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

Laboratory: Laboratoire des Sciences du Numérique de Nantes LS2N

Abstract

Acknowledgements

Notations

Abbreviations

List of Figures

List of Tables

Contents

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. 1.3 presents the applications of advanced planning methods for quadrotos. Finally, the control methods are shown in Section. 1.4.

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. 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.

1.1.2 Graph Search Algorithms

In this section, we cover the major graph search algorithms that will be useful in path planning. Forward search algorithms start at the initial state and explores the graph until encountering the goal state. The general template of such algorithms [5] 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 Motion Planning Methods

1.2.1 Grid Based Search

In grid based search, the configuration space is discretized into grids. Each grid point represents a configuration, and the robot may move to an adjacent grid, as long as the line between them is contained in $\mathcal{C}_{free}$. 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.

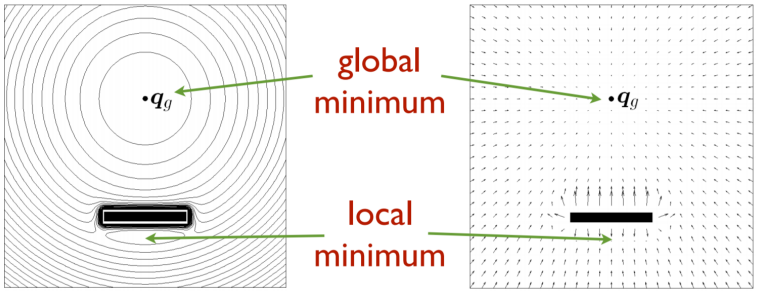


Figure 1.1: Local minima in potential field based search. [1]

1.2.2 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 a *repulsive potential*. This ensures that the robot moves towards the goal while avoiding obstacles.

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 method gives fast results in many cases, especially when the obstacles are stationary. However, this method may find a non-optimal path, or no path at all if the particle is stuck in a local minima (see Figure. 1.1).

1.2.3 Sampling-based Search

Sampling-based search algorithms are the leading methods to tackle higher-dimensional path planning problems. In this approach, the connectivity of $\mathcal{C}_{free}$ is explored by sampling random configurations and trying to connect in a network. Here, configurations and actions are represented by nodes and edges respectively. Two major sampling-based methods: *rapidly-exploring random trees* (RRT) [5] and *probabilistic roadmaps* [6], are discussed in this section.

The RRT algorithm explores the space by randomly sampling a space-filling tree. As shown in Algorithm 2, the tree is rooted at the initial configuration, and with each new sample, a vertex is added between it and the nearest state in tree. Once we obtain a connected graph between the initial state and the goal, search algorithms can be used to compute the best path.

Algorithm 2 Rapidly-exploring Random Trees

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

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. In the preprocessing stage, a random configuration is generated, and neighbours within a predefined distance are connected. This step is repeated until a dense graph is generated. 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.

1.3 Motion Planning for Quadrotors

This section is dedicated to motion planning methodologies for quadrotors. Our aim is to develop a planner that generates 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.

Until now, we assumed that the path between any two configurations can be easily determined. This is not always possible in the case of quadrotors, which have differential constraints that arise due to geometry and dynamics. *Kinodynamic motion planning* [7] 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 [8] 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.

1.3.1 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 [9], 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. [10] 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. [11] 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.

1.3.2 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. [8], 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:

$$\sigma_t = \begin{cases} \sum_{i=0}^{n} \sigma_{Ti1} t^i & t_0 \leq t \leq t_1 \\ \sum_{i=0}^{n} \sigma_{Ti2} t^i & t_1 \leq t \leq t_2 \\ & \vdots \\ \sum_{i=0}^{n} \sigma_{Tim} t^i & t_{m-1} \leq t \leq t_m \end{cases} \tag{1.3}$$

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:

$$\begin{aligned} \min_{\mathbf{c}} \quad & \int_{t_0}^{t_f} \left\| \frac{d^{k_r} \mathbf{r}_T}{dt^{k_r}}) \right\|^2 dt \\ \text{s.t.} \quad & \mathbf{r}_T = \mathbf{r}_w, \quad w = 0, \ldots, m \\ & \left. \frac{d^j x_T}{dt^j} \right|_{t=t_w} = 0 \text{ or free}, w = 0, \ldots, m; j = 1, \ldots, k_r \\ & \left. \frac{d^j y_T}{dt^j} \right|_{t=t_w} = 0 \text{ or free}, w = 0, \ldots, m; j = 1, \ldots, k_r \\ & \left. \frac{d^j z_T}{dt^j} \right|_{t=t_w} = 0 \text{ or free}, w = 0, \ldots, m; j = 1, \ldots, k_r \end{aligned} \tag{1.4}$$

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 program:

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{H} \mathbf{c} + \mathbf{f}^\top \mathbf{c} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{c} \leq \mathbf{b} \end{aligned} \tag{1.5}$$

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 [2] 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. 1.2, 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.

1.3.3 Search-Based Motion Planning

Recent work by Kumar et al. [3] 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. 1.3, 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 [3], 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.

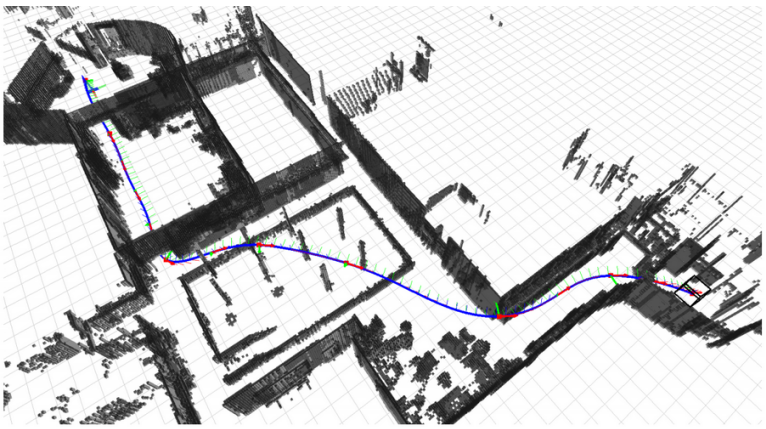


Figure 1.2: Minimum-snap trajectory generation using unconstrained quadratic program. [2]

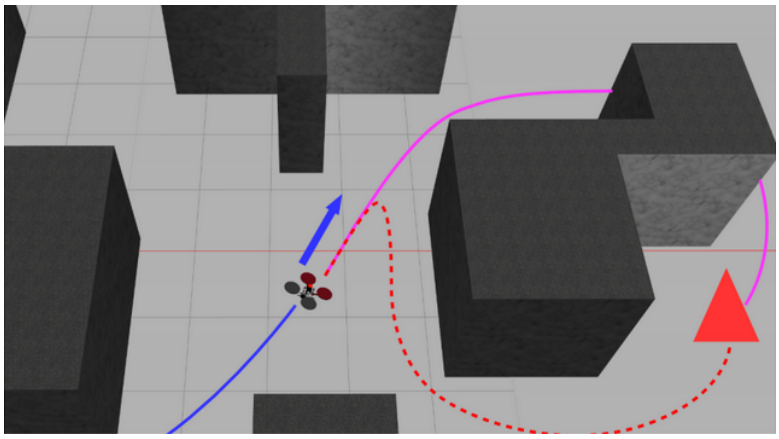


Figure 1.3: Path planning performed at non-zero inital velocity. [3] generates the more natural and time optimal magenta curve, where as other approaches gives rise to the red-dashed curve.

1.4 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 [12] have been used to control quadrotors by linearizing their dynamics around an operation point, typically the hover position. Better performance is obtained by non-linear control methods like backstepping, sliding mode [13] and feedback linearization [14].

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 [15] 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 [16] 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 [17], 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 [18] 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.

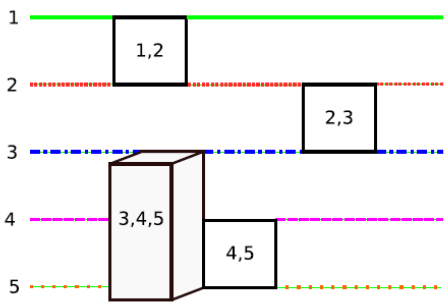
It is not necessary to plan in this space as the robots are usually well separated in the workspace, and collisions are infrequent. Some works generate feasible results by exploiting this decoupled property of the system. In this section, we discuss two such methods: M* [4] and conflict based search [19]. These methods have proven to produce remarkable results for cases with high congestion.

2.1.1 M* Search

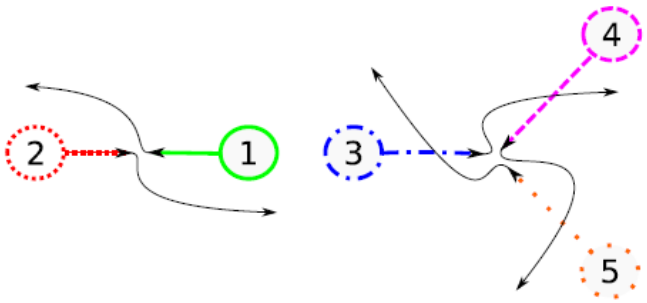
Initially, M* algorithm plans for each robot separately, without considering collisions. This is a good starting point for multi-robot planning as no path can be cheaper than this. When collisions occur, planning is performed in the joint configuration space (see Figure. 2.1) of the robots involved in 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* [20]. 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). [4]

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. [21] 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. [22] 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. [23], 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

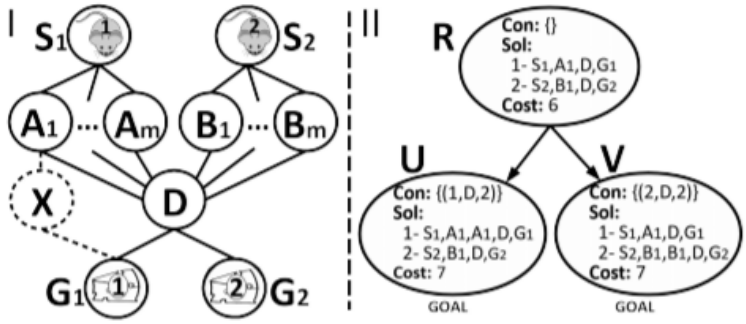


Figure 2.2: Example of CBS: (I) MAPF instance, and (II) its Constraint-Tree (CT)

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 as:

$$f_0(\chi) = \chi^\top P\chi + q^\top\chi + r \tag{2.4}$$

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:

$$\|p_i[k] - p_j[k]\|_2 \geq R \tag{2.5}$$

Using sequential convex programming, these constraints can be replaced with convex approximations around a previous solution χ^q using a first order Taylor expansion.

All these constraints are expressed in the form:

$$\begin{aligned} A_{eq}\chi &= b_{eq} \\ A_{in}\chi &\leq b_{in} \end{aligned} \tag{2.6}$$

and finally an optimum solution is found using quadratic programming.

2.3 Planning with time offsets

2.4 Velocity Profile Methods

2.5 Collision Avoidance based Methods

2.5.1 Nonlinear Model Predictive Control

2.5.2 Velocity Obstacle

2.6 Spline-based refinement of Waypoints

Chapter 3

Work Outline

Conclusion

Bibliography

[1] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

[2] 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.

[3] 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.

[4] 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.

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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

[22] 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.

[23] 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.