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

Basics of Path Planning

The planning problem has been studied extensively in various fields like robotics, artificial intelligence, and control theory. In this chapter, we discuss the fundamentals of path planning.

1.1 Preliminaries

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.

1.1.1 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 [1] is shown in Algorithm 1.

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*.

Algorithm 1 Forward Search

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

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 Motion Planning Algorithms

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.

1.2.2 Potential field based Search

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.

At every instance, the robot tries to move in a direction along decreasing potential. This method gives fast results in many cases, especially when the obstacles are stationary. However, this method will fail if the robot encounters a local minima.

1.2.3 Sampling-based Search

Sampling-based search algorithms are the leading methods to tackle higher-dimensional path planning problems. They have been successful in dealing with problems whose dimension is of the order of hundreds. Two major sampling-based methods, namely *rapidly-exploring random trees* (RRT) and *probabilistic roadmaps*

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:     $q_{rand} \leftarrow$RAND_CONFIG()
4:     $q_{near} \leftarrow$NEAREST_VERTEX($q_{rand}, G$)
5:     $q_{new} \leftarrow$NEW_CONF($q_{near}, q_{rand}, \Delta q$)
6:     $G$.add_vertex($q_{new}$)
7:     $G$.add_edge($q_{near}, q_{new}$)
8: return $G$
```

Probabilistic roadmap method is preferred if the initial and goal states are not fixed. This method consists of phases, a preprocessing and query phase. 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 Planning with Differential Constraints

Chapter 2

Multi-Agent Path Planning

2.1 Graph Search Methods

2.1.1 M*

2.1.2 Preliminary Concepts

2.2 Continuous optimization schemes

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