Path Planning for Wheeled Mobile Robots using an Optimal Control Approach

by

Belinda Thembisa Matebese

Dissertation presented for the degree of
Doctor of Philosophy (Applied Mathematics)
in the Faculty of Science at Stellenbosch University

Supervisors: Prof M. K. Banda
Dr D. J. Withey
Dr W. Brink

December 2019

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2019 ....

Abstract

The capability and practical use of wheeled mobile robots in real-world applications have resulted in them being a topic of recent interest. These systems are most prevalent because of their simple design and ease to control. In many cases, they also have an ability to move around in an environment without any human intervention. A main stream of research for wheeled mobile robots is that of planning motions of the robot under nonholonomic constraints.

A typical motion planning problem is to find a feasible path in the configuration space of the mobile robot that starts at the given initial state and reaches the desired goal state while satisfying robot kinematic or dynamic constraints. A variety of methods have been used to solve various aspects of the motion planning problem. Depending on the desired quality of the solution, an optimal path is often sought.

In this dissertation, optimal control is employed to obtain optimal collision-free paths for two-wheeled mobile robots and manipulators mounted on wheeled mobile platforms from an initial state to a goal state while avoiding obstacles. Obstacle avoidance is mathematically modelled using the potential field technique. The optimal control problem is then solved using an indirect method approach. This approach employs Pontryagin's minimum principle where analytical solutions for optimality conditions are derived. Solving the optimality condition leads to two sets of differential equations that have to be solved simultaneously and whose conditions are given at different times. This set of equations is known as a two-point boundary value problem (TPBVP) and can be solved using numerical techniques.

An indirect method, namely Leapfrog, is then implemented to solve the TPBVP. The Leapfrog method begins with a feasible trajectory, which is divided into smaller subdivisions where the local optimal controls are solved. The locally optimal trajectories are added and following a certain scheme of updating the number of subdivisions, the algorithm ends with the generation of an optimal trajectory along with the corresponding cost. An advantage of using the Leapfrog method is that it does not depend on the provision of good initial guesses along a path. In addition, the solution provided by the method satisfies both boundary conditions at every step. Moreover, in each iteration the paths generated are feasible and their cost decreases asymptotically.

To illustrate the effectiveness of the algorithm numerically, a quadratic cost with the control objective of steering the mobile robot from an initial state to a final state while avoiding obstacles is minimized. Simulations and numerical results are presented for environments with and without obstacles. A comparison is made between the Leapfrog method and the BVP4C optimization algorithm, and also the kinodynamic-RRT* algorithm.

The Leapfrog method shows value for continued development as a path planning method since it initializes easily, finds kinematically feasible paths without the need of post processing and where other techniques may fail. To our knowledge the work presented here is the first application of the Leapfrog method to find optimal trajectories for motion planning on a two-wheeled mobile robot and mobile manipulator.

Uittreksel

Die bekwaamheid en praktiese gebruik van robotte met wiele in werklike toepassings maak dat dit ’n onderwerp van belang is vir navorsing. Hierdie stelsels is algemeen vanweë hul eenvoudige ontwerp en gemak van beheer. Hulle het ook die vermoë om in ’n omgewing rond te beweeg sonder menslike bemiddeling. ’n Hoofstroom van navorsing vir wiel-mobiele robotte is die bewegingsbeplanning van ’n robot onderhewig aan nie-holonomiese beperkings.

’n Tipiese bewegingsbeplanning probleem is om ’n haalbare pad in die konfigurasie-ruimte te vind, vir ’n mobiele robot wat in ’n gegewe aanvangstoestand begin en uiteindelik ’n bestemde doeltoestand moet bereik terwyl kinematiese of dinamiese beperkings bevredig word. ’n Verskeidenheid metodes is al gebruik om verskeie aspekte van die bewegingsbeplanning probleem op te los. Afhangende van die gewensde kwaliteit van die oplossing, word ’n optimale pad dikwels gesoek.

In hierdie proefskrif word die bewegingsbeplanning probleem vir ’n tweewiel mobiele robot en mobiele manipuleerder wat op ’n mobiele platform met wiele monteer is, beskou. Hindernis-vermyding word wiskundig met die potensiaalveld-tegniek modelleer. en bewegingsbeplanning word as ’n indirekte optimale beheerprobleem formuleer. Hierdie benadering gebruik Pontryagin se minimum-beginsel, waar analitiese oplossings vir optimaliteitsvoorwaardes afgelei word. Die oplossing van hierdie optimaliteitsvoorwaardes lei na twee stelle differensiaalvergelykings wat gelyktydig opgelos moet word, en waarvan die voorwaardes op verskillende tye gegee word. Hierdie vergelykings staan bekend as ’n tweepunt-randwaardeprobleem (TPRWP), en kan met numeriese tegnieke opgelos word.

’n Indirekte metode, naamlik Leapfrog, word dan implementeer om die probleem op te los. Die Leapfrog-metode begin met ’n haalbare trajek wat in kleiner onderafdelings verdeel word, waar die lokale optimale beheer opgelos word. Die lokaal-optimale trajekte word bymekaar gevoeg, en volgens ’n sekere skema om die aantal onderafdelings op te dateer, eindig die algoritme met die skep van ’n optimale trajek sowel as die ooreenstemmende koste. ’n Voordeel van die Leapfrog-metode is dat dit nie van die voorsiening van goeie aanvanklike skattings vir ’n pad afhang nie. Die paaie wat op elke iterasie deur die metode verskaf word, bevredig ook albei randvoorwaardes. Verder is die paaie wat op elke iterasie geskep word haalbaar, en hul koste neem asimptoties af.

Om die doeltreffendheid van die algoritme numeries te demonstreer word 'n kwadratiese koste minimeer, met die beheer-doel om die mobiele robot van 'n aanvanklike toestand tot by 'n finale toestand te bestuur terwyl hindernisse vermy word. Simulasies en numeriese resultate word vir omgewings met en sonder hindernisse aangebied. 'n Vergelyking met die BVP4C-optimeringsalgoritme word gemaak, asook met die kino-dinamiese RRT*-algoritme.

Die Leapfrog-metode toon waarde vir verdere ontwikkeling as 'n optimale padbeplanningsmetode, aangesien dit maklik inisialiseer, 'n haalbare pad op elke iterasie skep, en oplossings kan vind waar ander metodes misluk. Sover ons kennis strek is die werk wat hier aangebied word die eerste aanwending van die Leapfrog-metode om optimale trajekte te vind vir bewegingsbeplanning van 'n twee-wiel mobiele robot en mobiele manipuleerder.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisors, Prof Mapundi K. Banda and Dr Daniel Withey, for their enthusiasm, guidance and unrelenting support throughout my PhD research studies. They have routinely gone beyond their duties in helping me with my worries, concerns and anxieties, and have worked to instil great confidence in both myself and my research work. To Dr Willie Brink for his assistance on academic matters and always responding to my questions and queries so promptly. His guidance helped me in all the time of writing this dissertation. I could not have imagined having better supervisors for my studies.

I am grateful to my friends and family who have supported me along the way, I am blessed to have them in my life. To my partner, Terence Ratshidaho for his ceaseless encouragement. Through your love, patience and unwavering belief in me I have been able to complete this long dissertation journey.

Last but not the least, I would like to thank the Mobile Intelligent Autonomous Systems (MIAS) team for their help and support through out the years. Without their willingness to share knowledge, this research work would not have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, mobile robots have received a great deal of research attention and have been increasingly used in many real-world applications. Applications for mobile robots abound in mining, farming, security, package deliveries, factory automation, and space exploration. A primary task that the mobile robot needs to do in these applications is to maneuver itself to move from one point to another within its environment, without colliding with obstacles. For the robot to perform its task an efficient motion planning method is required. Motion planning is an essential topic in mobile robotics and can be defined informally as finding a feasible (collision-free) path between the given initial state and final goal state in an environment while satisfying the robot kinematic or dynamic (in short, differential) constraints and avoiding collision with static or moving obstacles [1].

From the literature, path and motion planning problems are two distinct parts of mobile robotics, but they are intimately related [6]. The task of path planning methods is to find a geometric path from an initial to a final point. The motion planning problem, on the other hand, finds a trajectory considering the robot geometry, motion model and time. The difference between the two is that a geometric path does not specify how fast a robot should traverse the path, while a trajectory prescribes how the configuration of the robot evolves over time [7]. The path planning problem is a subset of the general motion planning problem. This dissertation focuses on motion planning, i.e. generating trajectories. The motion and path planning terms, however, are used interchangeably for convenience.

Common approaches include graph-based methods, potential fields, and sampling-based methods. The A* algorithm [8] is an example of a graph-based method which uses a heuristic search that operates over a discretized search space. Potential fields [9], on the other hand, follow a gradient of a potential field that guides a robot to its goal. Each of these algorithms has its own advantages and shortcomings in finding the most efficient solution; these are discussed in Section 1.1.2.

Sampling-based methods are popular and have had remarkable success in solving motion planning problems in a high-dimensional environment. These algorithms include the Rapidly-exploring Random Tree (RRT) and its successor RRT*. Unfortunately, for RRT it was proved that the cost of the best trajectory returned by the algorithm converges almost surely to a non-optimal path [10]. Also, the RRT algorithm generates paths consisting of piecewise straight lines with jagged motion. Post-processing, such as path smoothing [11], is typically implemented to satisfy the motion constraints and parameterize the path so that the robot can follow it. The RRT* algorithm is an improvement of RRT in that the cost of the best path it returns converges to the optimum almost surely at almost the same order of computational cost [10]. Extensions of RRT* can be found in [12; 13].

Recent advances in mobile robotics require that the robots perform their tasks quicker, more accurately, safely, and smoothly. These additional requirements have shifted the focus of research from designing feasible and optimal solutions where the robot is modelled as a point (e.g. A* as well as RRT* in its simplest form) to optimal solutions where the robot model includes differential constraints. Describing robot motion by differential constraints is referred to as kinodynamic motion planning and was introduced in [14]. Kinodynamic planning led to the improvement of sampling-based methods to achieve asymptotically optimal solutions for systems with differential constraints [15; 16; 17; 18]. Despite the long history of robotic motion planning, the inclusion of differential constraints in the motion planning problem is currently considered an open challenge [19], especially when it comes to guaranteeing the quality of the resulting solution and class of dynamical systems (linear vs. nonlinear), that can be addressed.

Optimal control, often referred to as trajectory optimization, is another family of motion planning techniques. Optimal control can be considered as the process of finding control and state laws for a dynamic system over some time so that the performance of the system is optimal regarding some criterion, such as control effort, tracking error, energy consumption, or amount of time taken to reach a target [20]. Nowadays, the advantages of this method are well established and have been studied, including in the field of optimal motion planning for mobile robots [21; 22; 23].

Due to the dimensionality and complexity of mobile robot models, however, solving optimal control problems can generally lead to a much higher computational overhead and potential numerical instability. In addition, solving path planning problems using an optimal control approach in an environment with obstacles can be challenging as obstacles introduce non-convex constraints into the environment [1].

This dissertation aims to solve motion planning problems by undertaking mathematical investigations using the optimal control approach. The intention is to use methods that are also, as far as possible, mathematically sound rather than just heuristic. The aim is to find paths that are kinematically feasible, optimal and at the same time preserve mathematical properties like smoothness. The Leapfrog algorithm [24] for solving nonlinear optimal control problems is proposed as a tool to solve the motion planning problem for a two-wheeled mobile robot and mobile manipulator. These mobile robot platforms are commonly used in motion planning [25; 26; 27], and the implication is that since Leapfrog can be used for these then it should be useful on other systems with differential constraints, including those with more degrees of freedom. In previous work, the Leapfrog method has been used within the context of control systems, that is, to generate control inputs for a system, but not in motion planning, though its features suggest that it could be useful there. This dissertation presents the first study of applying the Leapfrog method to find optimal trajectories for mobile robot, and mobile manipulator, motion planning. The capability and effectiveness of the Leapfrog method are demonstrated through experimental studies.

1.1 Background and Motivation

In this section, an overview of kinematic constraints for a mobile robot is given. This is followed by a general description of motion planning methods and a discussion of some of their advantages and disadvantages. An overview of different approaches to solving the optimal control problem is given, and motivation is provided for the selection of the Leapfrog method. The fundamental problem formulation for motion planning based on optimal control is then defined.

1.1.1 Kinematic Constraints

Kinematic constraints impose a relationship between the configuration of a robot and its velocity and can be further classified into holonomic and non-holonomic constraints. In [28] holonomic is described as the relationship between the controllable and total degrees of freedom (DoF) of a robot where the controllable degrees of freedom are equal to the total degrees of freedom. Otherwise, if the number of controllable degrees of freedom is less than the total degrees of freedom, it is a nonholonomic constraint. In [1] nonholonomic constraints are referred to as differential constraints that cannot be completely integrated.

Differential drive robots are common nonholonomic robot systems. Such robot platforms are prevalent in motion planning because of simplicity, good maneuverability and the fact that they work well in an indoor environment. A differential drive comprises of two independent driven wheels and one (or two) caster wheel(s) to provide balance.

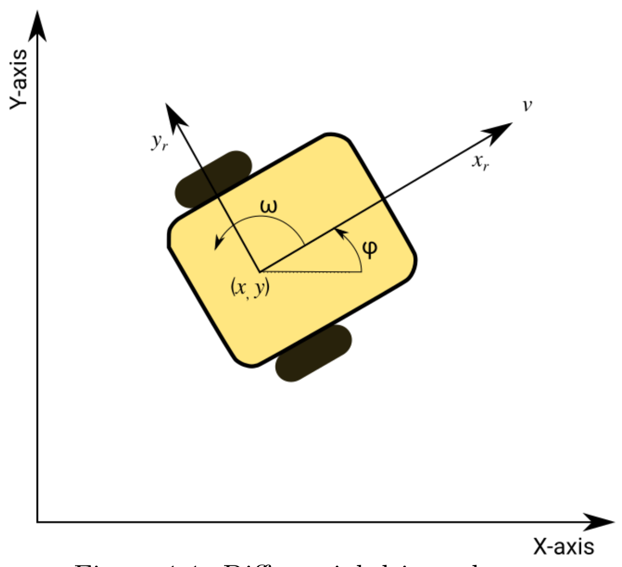


Figure 1.1: Differential drive robot.

The differential drive robot shown in Figure 1.1 has 3 DoF in its relative positioning which are represented by a pose $\mathbf{q} = [x\, y\, \varphi]^T$, where (x, y) is the position, and φ is the orientation of the robot. The global reference frame is given by the inertial basis $\{X\text{-axis}, Y\text{-axis}\}$ and $\{x_r, y_r\}$ is used as the robot's local reference frame. The differential drive, however, can only be controlled in two dimensions due to its nonholonomic constraint. This constraint exists due to the fact that, under conditions of no slip, the robot cannot have a velocity parallel to the direction of its axle. That is, the robot is capable of moving with a velocity in its body fixed x_r direction and rotating with an angular velocity ω, but it cannot have a velocity in its body fixed y_r direction. Furthermore, the non-slip constraint [29] is applied to the robot as

$$\dot{x}\sin\varphi - \dot{y}\cos\varphi = 0. \tag{1.1.1}$$

Assuming that the mobile robot model neglects external forces such as friction on the wheels or the possibility of wheel slippage, the mobile robot is said to satisfy the pure rolling constraint

$$\dot{x}\cos\varphi + \dot{y}\sin\varphi = v. \tag{1.1.2}$$

1.1.2 Motion Planning Methods

A motion planning problem is typically solved in the configuration space, $\mathcal{C}$, in which each configuration of the robot in a trajectory can be mapped to a point [30]. The position of a mobile robot in configuration space is denoted by $\mathbf{q} \in \mathcal{C}$. The obstacle region of the configuration space is $\mathcal{C}_{obs} \subseteq \mathcal{C}$ where the free space is then defined as $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$. In a configuration space, the path planning problem can be defined as follows:

Definition 1.1.1 (Path Planning Problem [19]). *Given the initial configuration* $\mathbf{q}_0 \in \mathcal{C}_{free}$*, goal configuration* $\mathbf{q}_f \in \mathcal{C}_{free}$ *and an obstacle set* $\mathcal{C}_{obs}$*, find a collision-free path* $\mu_z : [0, T] \rightarrow \mathcal{C}_{free}$ *such that* $\mu_z(0) = \mathbf{q}_0$*,* $\mu_z(T) = \mathbf{q}_f$ *and* $\mu_z(t) \in \mathcal{C}_{free}$ *for all* $t \in [0, T]$*.*

Figure 1.2 shows a basic path planning problem in a $\mathcal{C}$-space, where the shaded region represents obstacle space ($\mathcal{C}_{obs}$), the white region represents $\mathcal{C}_{free}$ and a path which connects the start and goal configurations, $\mathbf{q}_0$ and $\mathbf{q}_f$. Definition 1.1.1 is the simplest type of a path planning problem and is reasonably well understood in literature as finding an obstacle-free "geometric" path. Several methods have been proposed and applied to this path planning problem. Traditional approaches include sampling-based methods, graph-based methods, and artificial potential fields [1].

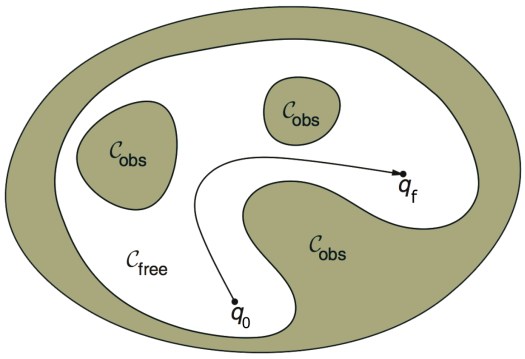
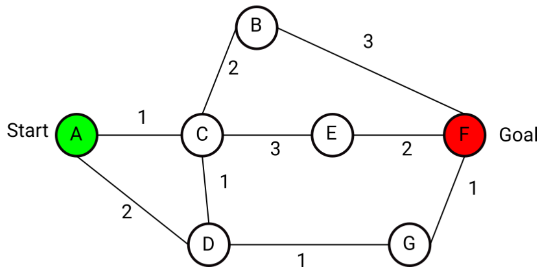


Figure 1.2: An example of the basic path planning problem [1].

The well-known Dijkstra's algorithm [31] and the A* algorithm [32] are examples of graph-based methods. Graph-based methods are relatively easy to implement and can return optimal paths. They discretize $\mathcal{C}$-space into a grid, and perform graph search on the grid vertices, discarding them if they are in a collision with obstacles. The desired path is then found by performing a search for a minimum-cost path in the graph. Dijkstra's algorithm creates a tree of shortest paths from the starting node to all other nodes in the graph. Figure 1.3(a) shows an example of the algorithm where the shortest path is obtained from a starting node (A) to a target node (F) in a weighted graph. From Figure 1.3(a), Dijkstra's algorithm will try to avoid edges with larger weights and take the shortest path $A \rightarrow C \rightarrow D \rightarrow G \rightarrow F$. A major disadvantage of the algorithm is the fact that it does a blind (uninformed) search thereby consuming a lot of time.

The A* algorithm can find an optimal path more efficiently by directing the search towards the goal using a heuristic function. Consider a 2-D grid as shown in Figure 1.3(b) with a starting cell (top left), a goal cell (bottom right) and obstacles (shaded region). To reach the target cell from the starting cell, A* would follow the path (solid line) as shown in Figure 1.3(b). Even though



(a) Dijkstra's algorithm.

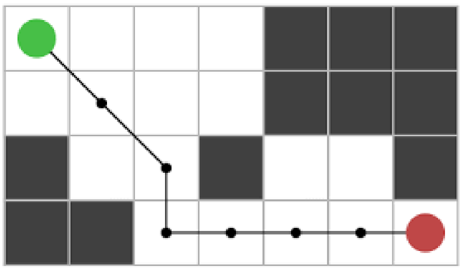
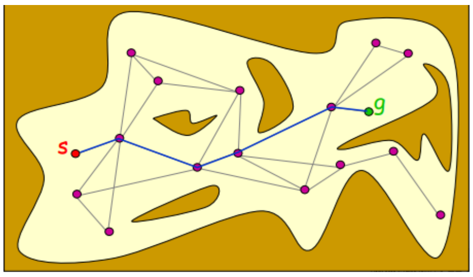

(b) A* algorithm.

Figure 1.3: Graph-search planners.

paths from graph-based methods are optimal in terms of transition costs in a graph (in discrete space), they are not necessarily optimal in terms of the robot operation in continuous space. Therefore these optimal paths need some post-processing to improve the quality of the path further. In addition, the memory required to store the grid and the time to search it grow exponentially with the number of dimensions of the space, which limits the algorithms to low-dimensional problems.

Sampling-based methods such as probabilistic roadmap (PRM) [33] and rapidly-exploring random tree (RRT) [19; 34] shown in Figure 1.4 are proven to be reliable in finding collision-free paths relatively quickly in high-dimensional space. These methods are efficient in practice and have probabilistic completeness guarantees, i.e., as the number of iterations grows, the probability of finding a solution path tends to unity, if a solution exists [19]. The PRM deals with preprocessing the configuration space of a kinematic system to generate a roadmap that can be used to answer multiple queries quickly. An example of how to solve a query with a roadmap is shown in Figure 1.4(a), where the solid (blue) line represents the shortest path obtained from the initial state (s) to the goal state (g) in the graph.



(a) PRM [33].

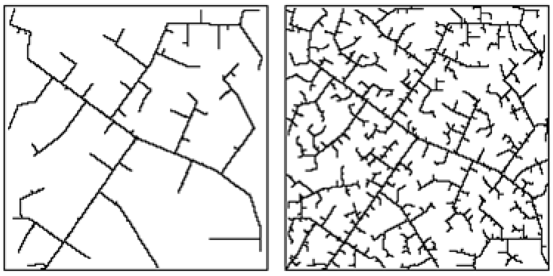

(b) RRT [1].

Figure 1.4: Sampling-based methods.

The RRT algorithm, on the other hand, is a single query algorithm that grows a tree from the initial to a goal state, finding a feasible path by adding a new edge or vertex in each iteration while avoiding obstacles. Hence it has an advantage of finding a feasible path relatively quickly, in high-dimensional and complex environments. The ability of RRT to explore free space is illustrated in Figure 1.4(b) where the RRT tree is shown to be growing denser with increasing iterations. Compared to PRM, RRT always maintains a connected structure using a small number of edges, while the PRM often suffers in performance because many extra edges are generated in an attempt to form a connected roadmap [35].

The sampling-based algorithms mentioned above are only concerned with finding a geometric collision-free path and do not explicitly consider the optimality of the path against a desired performance index. It was proven in [10] that, as the number of samples increases, the probability that the RRT algorithm converges to a sub-optimal solution increases. This shifted the focus of research on sampling-based methods from producing feasible and collision-free solutions to achieving high-quality solutions with minimal cost.

Definition 1.1.2 (Optimal Path Planning Problem [10]). *Let $c : \sum_{\mathcal{C}_{free}} \to \mathbb{R}^+$ be a cost functional, which assigns a non-negative cost to all nontrivial, collision-free paths. Given an initial configuration $\mathbf{q}_0 \in \mathcal{C}_{free}$, a goal configuration $\mathbf{q}_f \in \mathcal{C}_{free}$, an obstacle set $\mathcal{C}_{obs}$ and a dynamic system, find a collision-free trajectory $\mu_z^* : [0, T] \to \mathcal{C}_{free}$, that solves the Path Planning Problem 1.1.1, and moreover minimizes the cost functional, c.*

The question of optimality was addressed in [34] where the RRT* algorithm was introduced as an improvement to RRT in that the cost of the best path it returns converges to the optimum almost surely as the number of iterations increases, and at almost the same order of computational cost [10]. The RRT* algorithm provides a significant improvement in the quality of the path discovered in the configuration space over its predecessor, the RRT. This line of work provided other asymptotically optimal solutions for motion planning algorithms such as PRM* [34].

Even though RRT* guarantees to find the optimal solution asymptotically, the algorithm suffers from slow convergence, that is, the algorithm has a longer execution time to produce best paths [36; 37]. This is due to the algorithm making many additional calls to the local steering procedure that associates to any pair of configurations a shortest path between them to improve the discovered paths continuously.

To deal with mobile robot differential constraints, an optimal kinodynamic motion planning problem is defined, as follows.

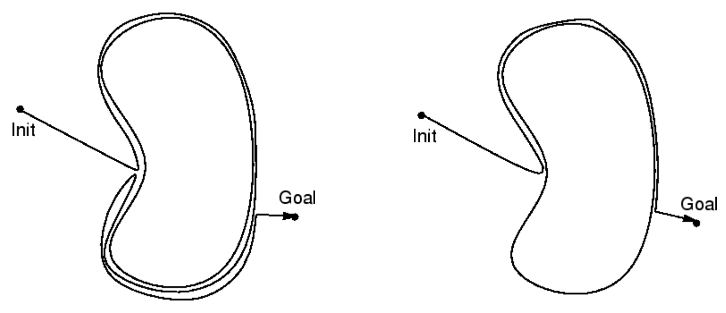
Definition 1.1.3 (Optimal Kinodynamic Planning Problem [15]). *Given an initial configuration $\mathbf{q}_0 \in \mathcal{C}_{free}$, a goal configuration $\mathbf{q}_f \in \mathcal{C}_{free}$, an obstacle set $\mathcal{C}_{obs}$ and a smooth function $\mathbf{f}$ that describes the system dynamics, find a control $\mathbf{u} \in \mathbb{R}^m$ (m stands for the number of controls) with domain $[0, T]$ such that a* *unique corresponding trajectory* $\mu_z^* : [0, T] \to \mathcal{C} \setminus \mathcal{C}_{obs}$, *with* $\dot{\mathbf{q}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t))$ *for all* $t \in [0, T]$, *avoids obstacles, reaches the goal region, and minimizes the cost functional* $J(\mathbf{q}(t), \mathbf{u}(t)) = \int_0^T g(\mathbf{q}(t), \mathbf{u}(t))dt$.

For RRT*, particularly, several extensions have been proposed to deal with mobile robot differential constraints [15; 16; 19; 38; 39], but despite this recent progress in solving kinodynamic-RRT* problems, the question of a systematic and efficient method to handle generic nonlinear dynamics, which inevitably involves two-point boundary value problem (TPBVP) solutions, in the RRT* process, remains [40].

1.1.3 Obstacle Avoidance

Obstacle avoidance is one of the vital aspects of motion planning for mobile robots as it is necessary for vehicle safety. A variety of obstacle avoidance algorithms have been designed and implemented for mobile robot path planning. These obstacles can be represented as either static or dynamic (moving) in an environment. In this section, however, only a brief discussion of the methods is given. An overview of these obstacle avoidance algorithms for mobile robots can be found in [29; 41].

According to literature, the first algorithms proposed for the discussion of obstacle avoidance are the Bug algorithms introduced in [42; 43]. The Bug algorithms are known as the simplest obstacle avoidance algorithms. These algorithms assume only local knowledge of the environment and a global goal. The basic idea is to follow the contour of each obstacle in the robot's way, and if the direction toward the goal is available again, the robot leaves the obstacle and goes toward the goal. An example of the Bug algorithms is given in Figure 1.5 where the robot starts from an initial point (Init) then departs from that point along the path with shortest distance, toward the goal. The Bug 2 algorithm has a shorter travel time compared to the Bug 1 algorithm and is more efficient especially in open spaces (see Figure 1.5). However, there are scenarios where the Bug 2 algorithm may be non-optimal [44].



(a) Bug 1 algorithm. (b) Bug 2 algorithm.

Figure 1.5: Paths generated by Bug algorithms [1].

Another approach is the artificial potential field (APF) technique introduced by Khatib [9]. The simple idea behind potential fields is to attract the robot towards the target while repelling it from the obstacles at the same time. The APF gained popularity in motion planning for mobile robots due to its mathematical simplicity and elegance. Figure 1.6 shows a representation of total potential field (U_{tot}) which consists of an attractive potential field (U_{att}) around the goal and a repulsive potential field (U_{rep}) around the obstacles.

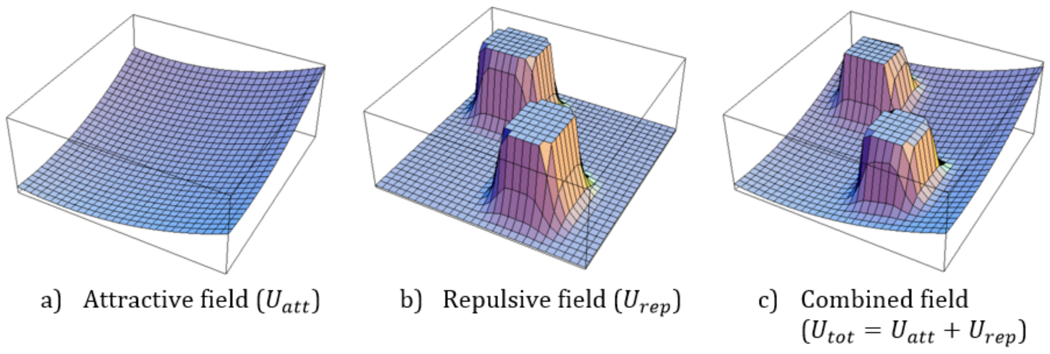

a) Attractive field (U_{att})
b) Repulsive field (U_{rep})
c) Combined field ($U_{tot} = U_{att} + U_{rep}$)

Figure 1.6: Representation of a potential field [2].

The main drawback of the method is that it is sensitive to local minima, i.e., the robot can become trapped at a point far from the target. Local minima have the effect of causing planning or navigation of the robot to fail, as the field will not produce any further progress towards the goal when it is encountered. Many works on artificial potential fields were developed to overcome this issue [45; 46]. Solving the local minimum problem has been a serious effort and an active research topic for potential field methods. Some other extensions to the potential field include the establishment of the navigation potential-based function [47] and harmonic potential functions [48] which are local minimum free.

To deal with the shortcoming of potential fields, Borenstein and Koren proposed the Virtual Force Field method [49]. This method, however, suffers from similar problems as the potential fields. Later the authors proposed the Vector Histogram Field (VHF) method [50] for fast obstacle avoidance. The VHF method is a real-time obstacle avoidance method that permits the detection of unknown obstacles and avoids collisions while simultaneously steering the mobile robot towards the target. The VHF solves the problem in two steps by computing a set of candidate motion directions and then selecting one of them. A disadvantage of using this method is that it may lead to the robot being driven far away from its target location. Further details on these steps can be found in [41].

In other instances where the motion planning problem is formulated as an optimal or trajectory optimization problem, obstacle avoidance can be a challenging task. Because these approaches do not naturally include obstacle avoidance, obstacles are typically added to the optimal control problem as constraints such as in [51] where constrained optimization techniques [52; 53] are then used to obtain the desired trajectories. Alternatively, the obstacles can be added in the cost function as in [25; 54; 55] where the optimal control problem is defined without constraints. Of particular interest in this dissertation is the use of Gaussian potential fields added to the cost functional to avoid obstacles and find optimal paths at the same time.

Sampling-based methods, on the other hand, do not rely on a particular collision detection algorithm. Because the role of a collision detection algorithm over the functioning of sampling-based algorithms is modular [56]. This approach provides information on whether a path collides with any obstacle as it is called several times over the course of planning. Collision checking is the most expensive operation in sampling-based planning.

1.1.4 Optimal Control

In motion planning, trajectory solutions are often required to be optimal with respect to some performance measure related to time, path length, or energy. Many researchers have studied the objective of designing optimal paths for wheeled mobile robots, the primary use being for minimum time paths which consist only of straight-line and circular-arc segments. This approach was used in [57; 58] for planning trajectories for wheeled mobile robots using analytical solutions. Later on, the problem was solved and refined by [59] and [60] with the help of optimal control techniques, which revealed scope for application of optimal control-based methods to mobile robot path planning.

The optimal control approach provides an excellent tool for finding optimal paths for mobile robot systems. It also offers more general numerical methods that are useful when solving problems with additional objective functions, constraints and boundary conditions. The optimal control approach has two roles in mobile robot motion planning. One is that optimal control defines a local method that can be used to find shortest paths between any two states, as is used, for example, in sampling-based algorithms such as kinodynamic-RRT* [41]. Secondly, it can be used in finding globally optimal trajectories directly from scratch, i.e., finding satisfactory solutions or a sequence of locally feasible solutions that converge to the globally optimal solution.

Optimal control problems can be solved either analytically or numerically. However, optimal control problems are generally nonlinear, and for most applications it is hard to find analytical solutions. As a result, it is often necessary to employ numerical methods to solve these problems. Numerical solutions to optimal control problems are classified as direct and indirect methods. The basic idea of direct optimization methods is to discretize the control problem, and then apply nonlinear programming (NLP) techniques [61] to the resulting finite-dimensional optimization problem. Compared to other local methods, direct methods have the advantage of being relatively robust and simple to implement. A drawback is that they provide sub-optimal approximate solutions. Moreover, their accuracy is less than that of indirect methods [62].

The indirect methods attempt to solve optimal control problems by seeking a solution to the necessary conditions of optimality derived through the Pontryagin minimum/maximum principle (PMP). The formulation of the necessary conditions of optimality leads to a TPBVP. Indirect methods are compelling, due to their rapid convergence to locally optimal solutions. However, deriving the necessary conditions of optimality is unique to each problem, so developing a tool for general optimal paths can be challenging. In addition, they may not find globally optimal solutions without an appropriate initial guess for costates [63].

The indirect shooting method is believed to be one of the simplest and easiest indirect methods to implement. The shooting method has been applied to motion planning for nonholonomic mobile robots in [64], [65] and [66]. In the shooting method, an initial guess is made of the unknown boundary conditions at one end of the interval. A TPBVP is then integrated from the initial time to final time or vice versa using the initial guess, together with the known initial conditions. Upon completion, the terminal conditions obtained from the numerical integration are compared to the known terminal conditions and the transversal conditions. If the integrated terminal conditions differ from the known terminal conditions by more than a specified tolerance, the unknown initial conditions are adjusted, and the process is repeated until the difference between the integrated terminal conditions and the necessary terminal conditions is less than some specified threshold.

The main drawback with the indirect shooting method is the need to find a good initial guess. Also, it suffers from slow convergence when the initial condition is far from the corresponding terminal condition [3]. An extension to the shooting method, namely the multiple shooting method [67], can be implemented to correct some of these challenges. In multiple shooting, a time interval is subdivided into small sub-intervals whose initial state is simultaneously updated at each iteration. Even though this method can be accurate, a good initial guess is required for each sub-interval. There are also constraints needed to guarantee that the initial and final states match the prescribed initial and final states. Since the trajectories for each sub-interval are discontinuous across interfaces of sub-intervals, constraints are used to make sure that the paths are connected.

The Leapfrog method proposed in [24] can be viewed as an indirect method for solving TPBVPs. The aim of the Leapfrog algorithm is to solve a global problem by building approximations from local solutions. First, an initial feasible path is constructed and subdivided into segments with the corresponding time interval. Optimal controls are then iteratively computed by building approximations from local solutions over each sub-interval. Following a midpoint scheme, the number of segments is updated, and the Leapfrog algorithm ends with the concatenated local paths as a trajectory from an initial to the desired final configuration. Under fairly general conditions, the method is convergent to a critical path, that is, a path that satisfies PMP [4]. It is also assumed that in each sub-interval local optimal control solutions exist and are unique.

The advantage of using the Leapfrog method over other indirect methods is that the method converges towards the optimal path, without the need for an initial guess for adjoint variables [4]. Instead, an initial feasible path is used. As part of the algorithm, once a feasible path is constructed, an affine approximation of the sub-problem in a subdivision (between nearby points) provides a good initial guess for adjoint variables. Also, the solution provided by the method satisfies both boundary conditions at every step, while in multiple shooting the boundary conditions are satisfied only after the whole iterative process is completed. Moreover, the paths generated by the Leapfrog method are feasible on each iteration.

The Leapfrog method also has a quality that is particularly useful for motion planning, in that it can be interpreted as an anytime computational framework for optimization problems with differential and geometric constraints. Evidence to this is that the Leapfrog method initially requires a feasible trajectory then incrementally improves it over time towards optimality. If the time allowed to find an optimal path for a system is not enough, the method can provide at least a feasible, sub-optimal path. This concept is also applicable to sampling-based methods such as RRT*, once an initial solution has been achieved.

In [68] it was demonstrated that although the affine approximation embedded in Leapfrog, and used to provide initial costates, reduces the possibility of making bad initial guesses for the costates, the approximation provides good guesses for only some of the costates, due to the nonlinearity characteristics of the mobile robot. With Leapfrog the initial costate guesses are used for the success of the local shooting method. In addition, the algorithm is convergent to a critical solution provided that these local shooting methods produce optimal trajectories. Thus in this dissertation, a gradient-based approach is used to estimate the initial costate values.

In the work done in [4], a simple shooting scheme was used to solve the local problem. The approach worked well for the example considered in [4], and also for the two-wheeled mobile robot planning problem without obstacles in [69]. In [70] a modified Newton's shooting method was proposed to improve the convergence of the Leapfrog method for solving mobile robot path planning problems in cluttered environments. The numerical simulations showed that the proposed approach took less time to compute the optimal path, as compared to the time required when simple shooting was used.

Overall, the Leapfrog algorithm shows value for continued development as a numerical method to find optimal paths for mobile robot planning since it initializes easily, generates kinematically feasible paths on each iteration and, as the number of iterations of Leapfrog increase, the cost decreases asymptotically.

1.2 Problem Statement

In this section, the path planning problems for a two-wheeled mobile robot and mobile manipulator are formulated as optimal control problems. These models are described below as Example 1.2.1 and Example 1.2.2.

Example 1.2.1 (Two-wheeled Mobile Robot)**.**

Consider the kinematic model of a differential drive mobile robot,

$$\dot{\mathbf{q}} = \begin{bmatrix} \cos\varphi & 0 \\ \sin\varphi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}, \tag{1.2.1}$$

where v and ω are control inputs representing the linear and angular velocities of the robot, respectively. The two-wheeled mobile robot system diagram is shown in Figure 1.1. Assuming no constraints on the states or the control variables, and that the initial and final times are fixed, an optimal control problem for the two-wheeled mobile robot model (1.2.1) can be defined as follows.

Problem 1.2.1. *Given an initial configuration $\mathbf{q}_0 \in \mathcal{C}_{free}$ and final configuration $\mathbf{q}_f \in \mathcal{C}_{free}$, find a set of control inputs $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^2$, for $t \in [t_0, t_f]$ to minimize the cost functional:*

$$\min_{\mathbf{u}\in\mathcal{U}} \quad \int_{t_0}^{t_f} (\mathbf{u}(t)^T R\mathbf{u}(t))\, dt, \tag{1.2.2}$$

subject to the kinematic system (1.2.1) described by a set of differential equations:

$$\dot{\mathbf{q}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)), \tag{1.2.3}$$

$$\mathbf{q}(t_0) = \mathbf{q}_0,\ \mathbf{q}(t_f) = \mathbf{q}_f, \tag{1.2.4}$$

where $[t_0, t_f]$ is the time interval, $\mathbf{q}$ is the state vector and R is a positive definite matrix.

As can be noted in Problem 1.2.1, the mobile robot planning is performed in the free configuration space without any obstacles in the workspace. Assuming that obstacles are represented by a potential field, the optimal control problem for the two-wheeled mobile robot in the presence of obstacles can be defined as follows.

Problem 1.2.2. *Given an initial configuration $\mathbf{q}_0 \in \mathcal{C}_{free}$, a final configuration $\mathbf{q}_f \in \mathcal{C}_{free}$, and a kinematic system described by a set of differential equations in Eq. (1.2.1), find a set of control inputs $\mathbf{u} \in \mathbb{R}^2$, for $t \in [t_0, t_f]$ that generates the mobile robot's collision-free path from an initial configuration to the desired final configuration by minimizing the cost functional:*

$$\min_{\mathbf{u}\in U} \quad \int_{t_0}^{t_f} \frac{1}{2}\Big(\mathbf{u}(t)^T R\mathbf{u}(t) + F_{rep_i}(\mathbf{q})\Big)dt. \tag{1.2.5}$$

The cost functional described in (1.2.5) includes the obstacle avoidance parameter which is defined by a Gaussian repulsive field [71]:

$$F_{rep_i}(\mathbf{q}) = A_{rep}\exp\left\{-\frac{1}{2}\left(\frac{(\rho^i)^2}{\sigma^2_{rep_i}}\right)^C\right\}, \tag{1.2.6}$$

where A_{rep} is a positive constant, ρ^i is the distance between the robot position and the center of the i-th obstacle, parameter C relates to the steepness of the potential field that represents the obstacle and σ_{rep_i} represents the size of the obstacle.

The optimal control problem is, therefore, modelled as:

$$\min_{\mathbf{u}\in U} \quad \int_{t_0}^{t_f} \frac{1}{2}\Big(\mathbf{u}(t)^T R\mathbf{u}(t) + \sum_{i=1}^{n} F_{rep_i}(\mathbf{q})\Big)dt \tag{1.2.7}$$

$$\dot{\mathbf{q}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)), \tag{1.2.8}$$

$$\mathbf{q}(t_0) = \mathbf{q}_0, \ \mathbf{q}(t_f) = \mathbf{q}_f. \tag{1.2.9}$$

Example 1.2.2 (Wheeled Mobile Manipulator)**.**

Consider the wheeled mobile manipulator (WMM) that consists of a differential drive mobile robot described above and a two-link manipulator arm mounted on top. The manipulator has two joints and two links and is mounted on the center of the mobile robot base. The joints of the manipulator are defined by angles θ_1 and θ_2 with the length of the first and second links represented as L_1 and L_2, respectively. The configuration of the WMM is represented by $\mathbf{q} = [x \ y \ \varphi \ \theta_1 \ \theta_2]^T$. Its kinematic model is given by

$$\dot{\mathbf{q}} = \begin{bmatrix} \cos\varphi & -d\sin\varphi & 0 & 0 \\ \sin\varphi & d\cos\varphi & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}, \tag{1.2.10}$$

where v and ω are the linear and angular velocities of the mobile platform, $\dot{\theta}_1$ and $\dot{\theta}_2$ represent the joint angle velocities, and d represents the offset distance from the center of the mobile wheels. Figure 1.7 clarifies.

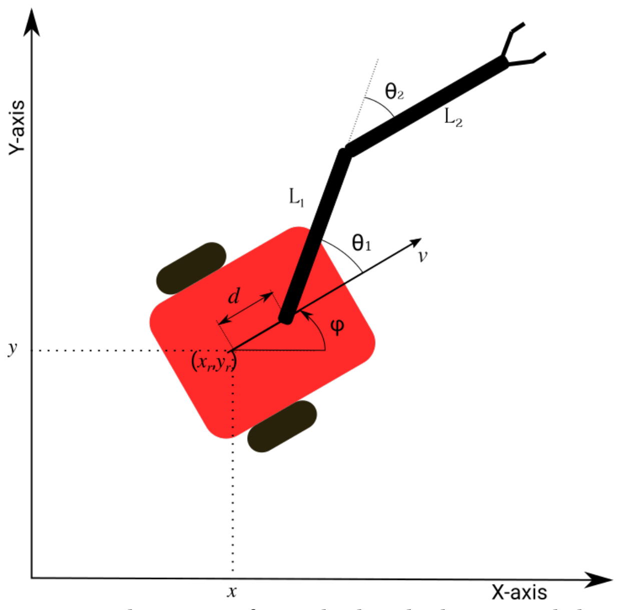


Figure 1.7: Systematic diagram of an idealized planar mobile manipulator.

Following Example 1.2.1, the mobile manipulator workspace is assumed to have static obstacles. The optimal control problem for the WMM is then defined as follows.

Problem 1.2.3. *Given an initial configuration $\mathbf{q}_0 \in \mathcal{C}_{free}$, a final configuration $\mathbf{q}_f \in \mathcal{C}_{free}$, a kinematic system of the mobile manipulator described in Eq. (1.2.10), and a cost functional described in Eq. (1.2.5), find a set of control inputs $\mathbf{u} \in \mathbb{R}^4$, for $t \in [t_0, t_f]$ that generates the mobile manipulator's collision-free path from the initial to the final configuration, avoiding $\mathbf{q} \in \mathcal{C}_{obs}$.*

The cost functional considered for this problem is similar to that in Problem 1.2.2 where it penalizes the energy effort and a Gaussian repulsive function. The difference is that additional inequality constraints are added to the Gaussian repulsive function to further restrict the mobile manipulator platform from colliding with the obstacles. Hence, the obstacle avoidance function is slightly different from Eq. (1.2.5); the formulation can be seen in Chapter 6.

1.3 Research Objectives

The aim of this dissertation is to present an efficient optimal control method to address the motion planning problem for mobile robots. To achieve that, the following objectives will be addressed in the dissertation:

(1) Model and design optimal control strategies for the two-wheeled mobile robot and mobile manipulator kinematics models.

(2) Apply the Leapfrog method to perform optimal path planning for both mobile robots, first in an environment without obstacles, and secondly in the presence of obstacles.

(3) Perform optimal kinodynamic motion planning for a two-wheeled mobile robot in the presence of obstacles using RRT* and compare with Leapfrog numerical solutions.

1.4 Contribution

Generating optimal movements can be achieved by minimizing a variety of quantities directly or indirectly involving some dynamic capacities of the robotic systems. Optimal control is a natural formalism for the representation of such problems. In this dissertation, a numerical optimal control technique, namely Leapfrog, is proposed to determine optimal paths for mobile robots in environments with obstacles. The work presented here is the first application of the Leapfrog method to find optimal trajectories for mobile robot motion planning.

The Leapfrog method relies on a solution of sub-problems through a given partition. It is assumed that in each sub-problem, there exists a locally unique optimal control. Also, the costate vector associated with a sub-problem is determined uniquely by the control input and the initial state values of the local optimal trajectory. Following these two assumptions, the Leapfrog method guarantees local optimality, because the underlying TPBVP is derived from the necessary conditions for optimality. The global solution is then achieved by building approximations from the local solutions.

To test the effectiveness of the Leapfrog method, the two-wheeled mobile robot and mobile manipulator platforms were chosen. The Leapfrog algorithm is not necessarily limited to these types of robotic systems, and the logical implication is that it can be applied to any dynamical system described by differential equations for which a TPBVP can be formulated.

In this dissertation, the Leapfrog method is evaluated in different environments cluttered with circular obstacles. It is noted that the obstacle set-up to test Leapfrog is not identical to the environments in which other motion planning techniques are tested. For example, sampling-based methods are usually tested in an environment with walls, narrow passages or using a parking scenario. However, the obstacle sets show examples at a level of complexity sufficient to demonstrate the usefulness of the Leapfrog algorithm for robot path planning.

The capability and effectiveness of the Leapfrog method are validated in numerical simulations, showing that the method initializes easily and can converge to a path that satisfies PMP. Real experiments conducted on a Pioneer 3-DX platform demonstrate that the solution obtained by the Leapfrog algorithm is efficient and can be deployed to a real robotic system without any need for post-processing.

Compared to general indirect methods, the Leapfrog method does not critically require an initial estimate for costates along a path. The Leapfrog method can naturally extend to general problems that involve robot differential constraints, while in existing motion planning algorithms the inclusion of differential constraints in the motion planning problem is considered an open challenge [16; 72]. In addition, Leapfrog finds the best path in less time as compared to the sampling-based kinodynamic-RRT* method. Furthermore, the path generated by Leapfrog is qualitatively less jagged as compared to that from kinodynamic-RRT*.

Even though the Leapfrog method shows promising results there are still issues with the implementation of the method, particularly when planning in a cluttered environment. The paths generated by Leapfrog may be optimal in some cases but optimality is not guaranteed, that is, the paths can be critical paths (local minima) but not guaranteed to be optimal (global minima). It is shown in simulations that Leapfrog does not always converge to a feasible path in some environments with obstacles when obstacle cost is relatively low, so selecting the parameters for obstacles is an important step. Another shortcoming of the method includes the reduction of partition points, which is important for efficient convergence of the algorithm. Algorithms for efficient partition point reduction will be considered as future work. In previous work [4; 73], the Leapfrog method used the affine approximation approach to help estimate initial costate values. For the mobile robot models considered in this dissertation, however, the approximation did not provide suitable guesses. As a result, a different, gradient-based approach is used.

The contributions of this work are as follows.

(i) In Chapter 4, the Leapfrog algorithm is presented as a method to find optimal trajectories for a two-wheeled mobile robot. It is demonstrated through simulations in [69] that the Leapfrog method successfully converges and produces kinematically feasible paths in each iteration. In addition, the cost along the path reduces throughout Leapfrog iterations with later paths showing improvements over the initial feasible path.

Evidence suggests that the Leapfrog algorithm iterates the feasible path toward an optimal path. Furthermore, path following experiments using a real robot are performed, demonstrating the utility of Leapfrog's paths.

(ii) In Chapter 5, the Leapfrog method is used to find trajectories for a mobile robot in an environment with obstacles. It is shown that obstacles in the environment can be represented as a potential field and included in the cost functional, allowing an optimal control problem to be solved using Leapfrog, for the required path planning. Though the Leapfrog method converges to optimal collision-free solutions for many scenarios, it is also shown that paths with collision may occur when the potential field parameters are not properly selected.

To improve the performance of the Leapfrog method, the following are also investigated.

- The Leapfrog algorithm is shown to be capable of converging towards an optimal path using initialization from other path planning techniques such as the A* and RRT algorithms. With Leapfrog, the initial feasible path is used to provide a good initial, sub-optimal guess needed for simple shooting in a subdivision. This work was published in [74].

- A gradient-based approach is implemented to compute the initial estimates of the costates. The approach is used due to the inherent nonlinear characteristics of the mobile robot system, in which the affine approximation embedded in Leapfrog failed to provide suitable costate initialization, causing the local solutions in the subdivisions to not converge.

(iii) In Chapter 6, the Leapfrog method is used to find paths for a mobile manipulator in the presence of obstacles. Here, the path planning problem is more complex (higher dimensionality) compared to that of the two-wheeled mobile robot considered in previous chapters. This is due to the higher DoF added by the two-link manipulator mounted on the mobile robot. Also, inequality constraints are added to the obstacle avoidance penalty to ensure that collision-free paths are generated. This work is presented in [75].

(iv) In Chapter 7, the kinodynamic-RRT* algorithm is used to find asymptotically optimal paths for the two-wheeled mobile robot. The numerical results are compared to Leapfrog results. The numerical results show that Leapfrog can improve the quality of the path in less time as compared to kinodynamic-RRT*.

(v) In another work, a modified Newton's shooting method to improve the convergence of the Leapfrog method for solving mobile robot path planning problems is proposed [70]. The numerical simulations show that the proposed approach takes less time to compute the optimal path, as compared to the time required when simple shooting is used.

The work presented in this dissertation has been published in the following peer-reviewed conference proceedings and a book chapter:

(1) B. T. Matebese, D. J. Withey and M. K. Banda, "Application of the Leapfrog method to robot path planning," in IEEE International Conference on Information and Automation, 2014. [69]

(2) B. Matebese, D. Withey and M. K. Banda, "Path planning with the Leapfrog method in the presence of obstacles," in IEEE International Conference on Robotics and Biomimetics, 2016. [76]

(3) B. Matebese, D. Withey and M. K. Banda, "Initialization of the Leapfrog algorithm for mobile robot path planning," in Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference, 2016. [74]

(4) B. Matebese, D. Withey and M. K. Banda, "Modified Newton's method in the Leapfrog method for mobile robot path planning", In: Dash S., Naidu P., Bayindir R., Das S. (eds) Artificial Intelligence and Evolutionary Computations in Engineering Systems. Advances in Intelligent Systems and Computing, vol 668. Springer, Singapore, 2018. [70]

(5) B. Matebese, D. Withey and M. K. Banda, "Optimal paths for a mobile manipulator using the Leapfrog method", in IEEE SAUPEC/RobMech/PRASA Conference, 2019. [75]

1.5 Thesis Outline

Chapter 2 gives an overview of existing motion planning algorithms, such as sampling-based methods, graph search methods, and potential fields. Discussions on some of their limitations together with approaches to overcome the issues are given. A general framework for kinodynamic motion planning with a sampling-based method is also given.

Chapter 3 The notation of methods and some preliminary material that is relevant to the dissertation are presented. First, the kinematic models of a two-wheeled mobile robot and mobile manipulator are derived. Secondly, the optimal control problem is formulated, and the necessary conditions of optimality are derived with the aid of calculus of variations and PMP. Lastly, the numerical methods, Leapfrog and BVP4C, that will be used to solve path planning as an optimal control problem in the indirect approach framework are discussed.

Chapter 4 Application of the Leapfrog algorithm is proposed to determine the optimal control solution in an unobstructed plane environment, for a two-wheeled mobile robot. The results of the numerical simulation are shown, and a comparison is made with the BVP4C optimal control algorithm. For the real experiments, a mobile robot platform is used to traverse a computed trajectory using path following control, to demonstrate that the computed path can be traversed by the robot.

Chapter 5 is an extension of the work done in Chapter 4. Firstly, the Leapfrog method is used to find optimal trajectories for mobile robot path planning in the presence of obstacles. Due to the obstacles present in the workspace, a feasible, collision-free, sub-optimal path is needed for the Leapfrog's first partition. Secondly, it is demonstrated that the Leapfrog algorithm is capable of finding optimal paths using different initialization approaches, such as using sub-optimal paths from RRT and A* planners, to form the initial partition.

Chapter 6 The optimal control problem in motion planning for a mobile manipulator in the presence of obstacles is presented. The obstacle avoidance is derived in terms of the distances between the obstacles and parts of the mobile manipulator. Then the optimal control problem is formulated for the mobile manipulator's kinematic system, and necessary conditions of optimality are derived. For evaluation, the paths generated by the Leapfrog method are compared to those generated by BVP4C.

Chapter 7 First, the optimal kinodynamic planning is done through the RRT* algorithm. Here the TPBVP for the two-wheeled mobile robot derived in Chapter 5 is considered. The kinodynamic-RRT* is tested using some of the numerical examples presented in Chapter 5. Secondly, the Leapfrog method is implemented using the path generated by Kinodynamic-RRT* as the initial partition. A performance comparison is made through simulations where optimality criteria such as path cost, path length, and runtime are evaluated.

Chapter 8 draws some concluding remarks, and provides a discussion involving the application of the Leapfrog method to mobile robot path planning and the prospective future work.

Chapter 2

Motion Planning Methods

The primary goal of motion planning algorithms is to provide a collision-free path from an initial state to the desired goal state within the configuration space of the robot. For a mobile robot to be able to plan a path from its starting state to a desired goal state, efficient motion planning algorithms are required.

An essential concept in motion planning algorithms is completeness. A motion planning algorithm is said to be complete if it finds a path whenever one exists or reports failure otherwise. The complete algorithms are called exact methods in the literature since they rely on the exact representation of the obstacles in the search space [1]. However, complete planning algorithms are difficult to implement and are computationally challenging. They are restricted to solving relatively simple problems and are not practical for complex, real-world applications. A simple version of the problem without dynamics is referred to as the Piano Mover's Problem, which was proven to be PSPACE-hard [77].

Research efforts have since shifted to algorithms that provide weaker completeness guarantees. The well-known graph-search methods such as Dijkstra's algorithm [31] and the A* algorithm [32] are examples of resolution complete algorithms. A planning algorithm is resolution complete if given a discretized problem, it will find a solution in finite time if one exists or correctly report that no solution exists within the specified parameters [78]. While popular graph-search algorithms like A* can guarantee convergence to optimal solutions, they typically suffer from exponentially increasing computational cost when applied in high-dimensional environments.

Sampling-based methods are popular and have had remarkable success in solving motion planning problems in high-dimensional environments. These methods compute candidate solutions based on samples drawn from a given distribution and usually produce a graph or tree. They are also efficient in practice and have probabilistic completeness guarantees, i.e. as the number of iterations increases, the probability of finding a solution path tends to unity if at least one solution exists.

RRT is one of the earliest sampling-based methods developed by Lavalle and Kuffner in [79]. RRT algorithms, however, return a solution once it is found, without considering the quality. As proven in [10], the cost of the best path returned by RRT converges almost surely to a non-optimal value. More recently, a sampling-based algorithm called RRT*, which provides asymptotically optimal solutions, was proposed in [12]. RRT* improves the path quality in terms of cost by introducing additional computational procedures. However, it obtains asymptotic optimality at a slower convergence rate.

Following these developments in optimal motion planning methods, several methods were proposed with the aim of providing asymptotically optimal solutions for systems with differential constraints [15], [38], [16]. This led to the improvement of sampling-based methods to achieve not only optimal solutions but also to include the robot system so that the solutions would be executable by the robot. However, the majority of the techniques used to solve optimal motion planning problems with differential constraints address specific classes of problems, such as linear or linearized systems. It was shown in [40] that the path obtained through linearization may not be realistic for actual systems and the path can be sub-optimal due to linearization error.

Also, considerable effort has been directed towards motion planning for wheeled mobile robots in the presence of obstacles, by employing a potential field. Artificial potential fields are used to direct the robot by providing a function over the state space, the gradient of which defines state-dependent motion vectors to be applied to the robot to move it past obstacles and towards the goal. In their simplest forms, however, potential fields are susceptible to local minima, that is the robot can become trapped at a point far from the target.

In this chapter, an overview of existing motion planning methods is given. Some of the advantages and challenges related to these methods are also discussed. A general framework for optimal motion planning for sampling-based methods with differential constraints is also given.

2.1 Sampling-based Methods

Sampling-based methods have proven to be extremely successful in addressing motion planning problems in high-dimensional spaces. Probabilistic Roadmaps (PRMs) [33] and RRT [34] are the main sampling-based methods and have been successfully used in solving motion planning problems in high-dimensional scenarios [80].

The PRM developed by Kavraki [33] is a multi-query method that first constructs a graph (roadmap) which represents a rich set of collision-free trajectories. It then answers queries by computing the shortest path that connects the initial state with the final state through a roadmap. PRMs are valuable in high-dimensional environments and are challenging when the environment changes, i.e., for online planning problems. With online planning the environment that the robot will operate in is unknown. In these applications, the initial calculation of the roadmap is often computationally expensive, or even infeasible.

The RRT, developed by Lavalle and Kuffner [34], represents another category of sampling-based methods, known as single-query methods. In this method, a tree is grown incrementally from the initial state to a goal state, finding a feasible path by adding a new edge or vertex in each iteration, while avoiding obstacles. Hence it has an advantage of finding a feasible path relatively quickly, in high-dimensional and complex scenarios. Compared to PRMs, RRT is faster because it does not need to sample the configuration space and then construct a roadmap. In addition, it always maintains a connected structure even if the number of edges is minimal, while PRMs often suffer in performance because many extra edges are generated in the attempt to form a connected roadmap.

A limitation of the RRT algorithm is that it suffers from the difficulty of determining or estimating a metric [35]. The algorithm depends on the existence of a metric because of the nearest neighbour operation. Given a target vertex, a distance metric is used to calculate a vertex that is near to the target in the existing tree. RRT-based methods consider path length in terms of Euclidean distance. In addition, the method remains ineffective when the configuration space has narrow passages [80]. This is caused by the number of samples covering the narrow passage, which may not allow the construction of a feasible path through it. Approaches to deal with very high dimensions can be found in [81] and approaches to deal with narrow passages can be found in [82].

Even though PRM and RRT have proved to be efficient, the cost of the best path returned by these methods converges almost surely to a non-optimal value. This shifted the focus of research on sampling-based method towards improving the quality of the paths. To address the issue, in [83] an anytime algorithm that runs RRT repeatedly to improve the quality of the solution was presented. In [84] an improved algorithm, namely RRT^{++}, was introduced. This algorithm returns feasible solutions as quickly as RRT and also approaches an optimal solution faster than the one presented in [83]. It is shown that as time progresses, its solution approaches a minimum cost.

To deal with optimality in sampling-based methods, in [12] important progress was achieved by using random graph theory [85] to show that sampling-based methods such as PRM* and RRT* can achieve asymptotic optimality.

2.1.1 RRT* Algorithm

Considering the literature on motion planning, the RRT* algorithm was a major breakthrough in optimal motion planning for high-dimensional problems in recent years, compared to other sampling-based methods. The RRT* algorithm inherits the key advantage of RRT, i.e. it explores the unexplored search space rapidly. In addition, it improves path quality by introducing additional computational procedures. Of particular interest in this dissertation is the RRT* algorithm. Hence in this section details of the algorithm are given. The RRT* algorithm essentially behaves identically to RRT except that RRT* considers a neighbourhood of near states when expanding the tree instead of choosing the nearest one. This procedure is performed within the area of a ball with radius defined as

$$r = \Upsilon \left(\frac{\log(n)}{n} \right)^{\frac{1}{d}}, \tag{2.1.1}$$

where d is the search space dimension, n is the size of the set of vertices, and Υ is a planning constant based on the environment [12]. Furthermore, adjusting the length of new connections in the rewiring procedure assures that RRT* improves the path and finds an optimal path as a comparison to RRT. As the number of iterations increases, RRT* improves its path cost gradually due to its asymptotic quality. On the other hand, this also slows down the convergence rate of RRT*.

Following [10], the RRT* algorithm is presented in Algorithm 1. It begins with initializing a graph $\mathcal{T}$ with vertices, $V = z_{init}$, and edges, $E = \emptyset$ (line 1). At each iteration, the algorithm randomly samples a state z_{rand} from $\mathcal{C}_{free}$ and finds the nearest node $z_{nearest}$ in the tree to this sampled state (lines 3-5). The algorithm then steers the system toward z_{rand} to determine z_{new} that is closest to z_{rand} and stays within some specified distance from $z_{nearest}$. The new node z_{new} is then added to the set of vertices V if the trajectory from $z_{nearest}$ to z_{new} is obstacle-free (lines 6-7).

Next, the best parent node for z_{new} is chosen from nearby nodes in the tree so that the trajectory from the parent to z_{new} is obstacle-free and of minimum cost (lines 8-18). After adding the trajectory segment from the parent to z_{new} (line 19), the algorithm rewires the nearby nodes in the tree so that the forward paths from z_{new} are of minimum cost (lines 20-26). Then the algorithm proceeds to the next iteration. The cost of the unique trajectory from the root vertex to a given vertex z is denoted as $\texttt{Cost}(z)$.

The major advantage that RRT* has over other planning methods is that it quickly finds a feasible path (not necessarily optimal) then incrementally improves it over time towards optimality. This indicates that the RRT* algorithm is a powerful tool that can be interpreted as an anytime computational

Algorithm 1 RRT* Algorithm

```
1: $\mathcal{T} \leftarrow \{z_{init}\}$ ; $E \leftarrow \emptyset$;
2: while $i < N$ do
3:     $z_{rand} \leftarrow$ Sample(i);
4:     $V' \leftarrow V; E' \leftarrow E$;
5:     $z_{nearest} \leftarrow$ Nearest$(\mathcal{T}, z_{rand})$;
6:     $z_{new} \leftarrow$ Steer$(z_{nearest}, z_{rand})$;
7:     $V \leftarrow V \cup \{z_{new}\}$;
8:     if CollisionFree$(z_{nearest}, z_{new})$ then
9:         $z_{min} \leftarrow z_{nearest}$;
10:        $Z_{nearby} \leftarrow$ Near$(\mathcal{T}, z_{new}, |V|)$;
11:        for $z_{near} \in Z_{nearby}$ do
12:            if CollisionFree$(z_{near}, z_{new})$ then
13:                $c' \leftarrow$ Cost$(z_{near})$ + c$(Line(z_{near}, z_{new}))$;
14:                if $c' <$ Cost$(z_{new})$ then
15:                    $z_{min} \leftarrow z_{near}$;
16:                end if
17:            end if
18:        end for
19:        $E \leftarrow E \cup (z_{min}, z_{new})$;
20:        for $z_{near} \in Z_{nearby} \backslash \{z_{min}\}$ do
21:            $z_{near} \leftarrow$ Steer$(z_{new}, z_{near})$;
22:            if CollisionFree$(z_{new}, z_{near})$ and
               Cost$(z_{near}) >$ Cost$(z_{new})$ + c$(Line(z_{new}, z_{near}))$ then
23:                $z_{parent} \leftarrow$ Parent$(z_{near})$;
24:                $E' \leftarrow E' \backslash \{(z_{parent}, z_{near})\}$;
25:                $E' \leftarrow E' \cup \{(z_{new}, z_{near})\}$;
26:            end if
27:        end for
28:    end if
29: end while
30: return $\mathcal{T}' = (V', E')$
```

framework for nonlinear optimization problems [66]. Thus, apart from having a probabilistic completeness guarantee it also ensures asymptotic optimality. Moreover, it was discussed in [12] that RRT* has advantages over other sampling-based planners in terms of time and space complexities.

2.1.2 Sampling-based Planning with Dynamical Systems

Motion planning with differential constraints is challenging because of the combinatorial complexity of the geometry of obstacles, and the nonlinear and possibly nonholonomic properties of the system. To deal with such challenges, the idea of kinodynamic planning was proposed in [14]. Kinodynamic planning refers to motion planning problems for which velocity and acceleration bounds must be satisfied. The first algorithms addressing the kinodynamic motion planning problem without a steering subroutine were the expansive space trees (EST) algorithm [86] and the RRT algorithm [19]. Both of these rely on a forward integration of the dynamics with random control inputs instead of a point-to-point local planning subroutine. The RRT algorithm is more robust with nonholonomic constraints and was specifically designed to operate in nonholonomic environments [79]. It was mentioned in [1] that kinodynamic planning is not necessarily a form of nonholonomic planning. A problem may even involve both nonholonomic and kinodynamic planning.

Incorporating differential constraints in RRT* is not as simple as the kinodynamic extension to RRT. In the two additional procedures of RRT*, i.e. choosing the parent and rewiring, two states need to be connected exactly and optimally. The development of optimal planning and RRT* has renewed interest in sampling-based methods. Preliminary work on extending the RRT* algorithm to handle systems with differential constraints was done in [15]. Minimum-time solutions were obtained for applications such as the double integrator and Dubin's vehicle dynamics, and it was proven that the optimality guarantee for RRT* in kinodynamic systems holds under certain conditions. One condition is that the trajectory found by the `Steer` procedure must be optimal in the absence of obstacles. Similarly, the cost estimate used by `Nearest` and `Near` must reflect the optimal cost. The work was also extended to generate optimal trajectories for minimum-time maneuvering of a high-speed race car [66].

The original RRT* was developed for systems with simple dynamics, where any pair of states can be optimally connected by a straight line from $z_{nearest}$ to z_{rand} in order to generate z_{new}. However, when considering the robot's differential constraints the steering function is a challenge to solve. The steering function represents a local planning problem, defined as a two-point boundary value problem (TPBVP). Solutions to certain classes of TPBVP can be achieved through optimal control theory. In [66], time-optimal maneuvers for a high-speed off-road vehicle were successfully generated using the shooting method [87] to solve the TPBVP. Despite its generality, the shooting method does not guarantee an exact connection between two states and may lead to a suboptimal trajectory.

Several approaches have also been proposed to solve the resulting TPBVPs associated with the local optimal paths between two nodes in the RRT* process, in particular when the differential constraints are linear or linearized. In [38; 39] a linear quadratic regulator (LQR) is employed as a steering solution. However, this approach does not have a guarantee of reaching the exact final state, hence optimality is not guaranteed. In [16] a closed-form analytical solution of the optimization problem for a TPBVP with fixed final state and free final time is obtained. This extension method was proven to converge to optimal solutions for linear systems. However, this approach is only applicable to a limited class of cost functions.

As an extension to nonlinear systems, a successive approximation approach named SA-RRT* was proposed in [40]. The authors obtained TPBVP solutions for the same cost described in [16], however they presented a more general class of differential equations as compared to [16]. In their work, it was confirmed that the path found by the proposed method is more natural and near-optimal for the dynamics of the system than the trajectory from other RRT* algorithms. Despite this recent progress, optimal motion planning that involves generic nonlinear dynamics, which inevitably involves the computation of TPBVP solutions in the RRT* process, remains a challenge.

2.2 Graph-search Methods

The graph-search methods are known for solving shortest path problems. Traditional methods for solving this problem are Dijkstra's algorithm [31] and the A* algorithm [8].

Dijkstra's algorithm is one of the earliest graph-search algorithms for finding shortest paths. The algorithm performs a best first search to build a tree representing shortest paths from a given source vertex to all other vertices in the graph. When only a path to a single vertex is required, a heuristic can be used to guide the search process.

The A* algorithm, on the other hand, can find an optimal path more efficiently by directing a search towards the goal using heuristic functions. The heuristic function provides an approximation for the cost of the best route that goes through each node, and the algorithm employs this heuristic estimate when determining which node to visit next in its search process. Even though these paths are optimal on the graph, they are not equivalent to an optimal path in the continuous environment. Furthermore, the paths are not smooth and post-processing techniques are required to smooth the paths.

There are several extensions to the A* algorithm that have been developed, such as an extension to re-planning and anytime planning algorithms. The most important property of anytime algorithms is that they try to provide a feasible path, possibly sub-optimal, and continue to improve the solution while time is available. These algorithms include Anytime A* [88], Anytime Repairing A* (ARA*) [89] and Anytime Dynamic A* (ADA*) [90]. While anytime planning algorithms are useful when good models of the environment are known a priori, they are less beneficial when prior models are not very accurate or when the environment is dynamic [91].

2.3 Potential Field Methods

The potential field approach appears to be a bit different in nature as compared to other motion planning techniques mentioned above. The use of the artificial potential field was proposed by Khatib [9] and has since gained popularity in path planning for wheeled mobile robots due to its mathematical simplicity and elegance. In this approach, a potential field is defined in the configuration space such that it has a minimum at the goal configuration. In potential function-driven motion planners, the robot is attracted towards the goal and repelled from the obstacles at the same time. The resulting force on the robot, which is the sum of the attractive force due to the goal and repulsive forces due to the obstacles, represented by

$$F_{tot} = F_{att} + F_{rep}, \tag{2.3.1}$$

determines the direction and movement. In Eq. (2.3.1), F_{att} and F_{rep} represent the attractive and repulsive fields, respectively. The most commonly used attractive field is given by

$$F_{att}(\mathbf{q}) = \frac{1}{2}\zeta\rho^2(\mathbf{q}, \mathbf{q}_{goal}), \tag{2.3.2}$$

where ζ ia s positive scaling factor, and $\rho(\mathbf{q}, \mathbf{q}_{goal})$ is the distance between the robot position $\mathbf{q}$ and the goal. The repulsive function is described as

$$F_{rep}(\mathbf{q}) = \begin{cases} \frac{1}{2}\eta\Big(\frac{1}{\rho(\mathbf{q}, \mathbf{q}_{obs})} - \frac{1}{\rho_0}\Big)^2, & \text{if } \rho(\mathbf{q}) \leq \rho_0 \\ 0, & \text{if } \rho(\mathbf{q}) > \rho_0 \end{cases} \tag{2.3.3}$$

where η is a positive scaling factor, $\rho(\mathbf{q}, \mathbf{q}_{obs})$ represents the distance between the robot and the obstacle, and ρ_0 denotes the distance of influence around the obstacle.

2.3.1 Potential Fields for Obstacle Avoidance

Potential fields have also gained increased popularity in mobile robotics for obstacle avoidance. However, they are known to suffer from inherent limitations as discussed in [92]. Local minima represent one of the major issues in employing these methods, and occur when there is a collinear alignment between the robot, obstacle and goal position [92]. This would mean that the sum of the attractive and repulsive forces acting on the robot becomes zero. To overcome this issue, in [47] a global potential field approach was introduced using a special function called the navigation function. Unlike other potential field methods, the navigation function has only one minimum point (at the goal) so that there are no local minima, however significant calculations are required. Other extensions of potential fields were developed such as in [46] where a navigation potential field was introduced. The other potential field technique that is guaranteed to be local minimum free is the harmonic potential field [48] which is based on the Laplace equation. Even though the resulting potential field does not have local minima, offline computation is required to provide a solution to the Laplace equation.

The other issue with potential fields is when the goal is too close to an obstacle which generates a higher repulsive force than the attractive one, preventing the robot from reaching the goal. This problem is called Goal Non-reachable with Obstacles Nearby (GNRON) and was identified in [93]. To overcome this problem, the authors improved the repulsive potential function and later proposed a new potential field [94]. It is noted from the literature that most of these extensions to potential fields improve the repulsive potential function to overcome some limitations. This may cause the repulsive function to become complex while the attractive function remains unchanged.

In [92] it was demonstrated that there is a possibility that a robot cannot pass through closely spaced obstacles due to repulsive forces created by the obstacles. This can be experienced when a robot passes through a door frame or narrow passage. A Gaussian function can be implemented as a potential field to overcome some of the issues like GNRON, and problems of no passage between closely spaced obstacles. In [95] the Gaussian-based potential field was used to model the attractive force and a high order Gaussian-like function to model obstacles in order to avoid local minima. Later, [71] employed the Gaussian function and a modified simulated annealing method for obstacle avoidance on multi-link robots. In [96] the Gaussian function was used to overcome the GNRON limitation by adding it to a traditional repulsive function. These approaches modify the conventional repulsive function to improve its performance and also to generate fewer occurrences of local minima.

In mathematical terms, the attractive potential field can be formulated as a Gaussian function as follows:

$$F_{att}(\mathbf{q}) = 1 - \exp\left[-\frac{1}{2}\left(\frac{\rho_{goal}^2}{\sigma_{att}^2} \right) \right], \tag{2.3.4}$$

where σ_{att} is a constant that defines the width of the Gaussian function. The parameter ρ_{goal} is defined as the distance from the robot to the goal:

$$\rho_{goal} = \sqrt{(x_r - x_{goal})^2 + (y_r - y_{goal})^2}, \tag{2.3.5}$$

where (x_r, y_r) and (x_{goal}, y_{goal}) are the positions of the robot and the goal, respectively. Then the repulsive field F_{rep_i} for the i-th obstacle can be described as

$$F_{rep_i}(\mathbf{q}) = A_{rep} \exp\left[-\frac{1}{2}\left(\frac{(\rho^i)^2}{\sigma_{rep_i}^2}\right)^C \right] \quad \text{for } i = 1, \ldots, n, \tag{2.3.6}$$

where n is the number of obstacles, A_{rep} is a positive constant representing the height of an obstacle, C determines the steepness of the obstacle representation within the field, σ_{rep_i} is the size of the i-th obstacle and ρ^i is the distance between the robot and the i-th obstacle:

$$\rho^i = \sqrt{(x_r - x_{obs_i})^2 + (y_r - y_{obs_i})^2}, \tag{2.3.7}$$

where (x_{obs_i}, y_{obs_i}) represents the position of the obstacle. The repulsive field represented in Eq. (2.3.6) can also be expressed as

$$F_{rep}(\mathbf{q}) = \sum_{i=1}^{n} F_{rep_i}(\mathbf{q}). \tag{2.3.8}$$

The total potential field, F_{tot}, on the robot is then given by

$$F_{tot}(\mathbf{q}) = F_{att}(\mathbf{q}) + \sum_{i=1}^{n} F_{rep_i}(\mathbf{q}). \tag{2.3.9}$$

As an example, Figure 2.1 shows four obstacles and a goal position as a total potential field represented by Gaussian functions.

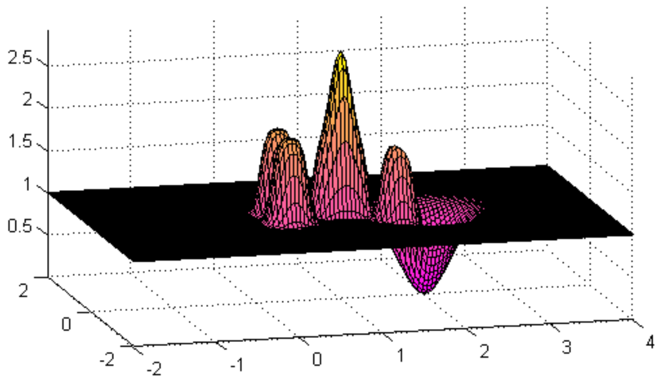
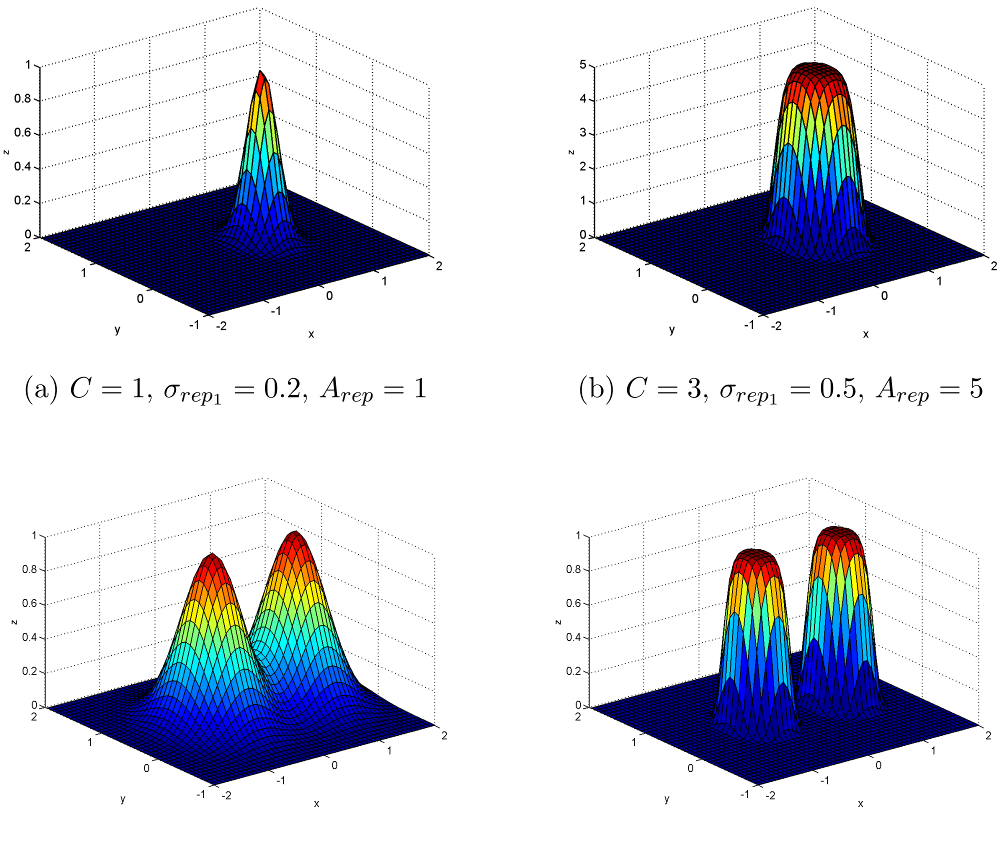


Figure 2.1: Potential field example showing four obstacles and the target generated by Gaussian functions.

Most potential fields found in the literature, for example the ones defined by Eq. (2.3.3), ensure that a robot never penetrates the boundaries of the obstacle and that the field becomes infinitely large at the center of the obstacle. This, however, can result in the robot getting stuck in local minima away from the goal. Gaussian functions, on the other hand, are smooth in nature as shown in Figure 2.2 and have finite height with a limited area of influence [97]. In addition, their derivatives are continuous and smooth functions, which makes this function suitable as a potential obstacle function for this dissertation.

With Gaussian potential fields, the behaviour of the robot around the obstacle area can be controlled by adjusting the parameters related to the height and steepness of the obstacle, that is, A_{rep} and C. By increasing the values of these parameters the allowed distance between the robot and the obstacle increases, leading to collision-free paths. Increasing parameter C on its own can create a minimum distance between obstacles (see Figure 2.2(d)) which in turn can reduce the occurrence of local minima. Figure 2.2 shows the effect of varying these parameters.



(a) $C = 1$, $\sigma_{rep_1} = 0.2$, $A_{rep} = 1$ (b) $C = 3$, $\sigma_{rep_1} = 0.5$, $A_{rep} = 5$

(c) $C = 1$, $\sigma_{rep_{1,2}} = 0.4$, $A_{rep} = 1$ (d) $C = 3$, $\sigma_{rep_{1,2}} = 0.4$, $A_{rep} = 1$

Figure 2.2: The effect of variance, σ_{rep_i}, height parameter, A_{rep}, and effective range parameter, C on the repulsive field shown in Eq. (2.3.6).

2.4 Conclusion

In this chapter motion planning methods were discussed. These included sampling-based methods, graph-search methods, and potential fields. A brief description of kinodynamic motion planning with sampling-based methods was given. It was noted that the current set of practical solutions using RRT* is limited mainly to linear or linearized systems. These systems, however, do not possess the generality of the original sampling-based method. Hence the motion planning problem with robot differential constraints has remained challenging for realistic systems, particularly with nonholonomic constraints. A brief discussion on graph-search methods together with their extension was given. Potential field methods for obstacle avoidance were also discussed along with their inherent limitations.

Chapter 3

Optimal Control Formulation

Mathematical modelling is essential when designing mobile robot systems. It generally simplifies a real-world problem and once a model is created, it can be used to approximate solutions of the system being modelled. In solving the model, various approaches can be employed including numerical methods for solving complex problems. An overview of the models implemented for this dissertation is given in Chapter 1. In this chapter, optimal control formulation and numerical techniques to solve the optimal control motion of the robots are described.

As mentioned before, the two-wheeled mobile robot platform is one of the simplest and most-used platforms in mobile robotics applications. Mobile manipulators, on the other hand, take advantage of the increased mobility and workspace provided by the mobile base. A mobile manipulator system is typically composed of a mobile base platform with one (or more) manipulators mounted on top of the base. Unlike the mobile base platforms, mobile manipulators have been recently researched and are now becoming commercially-available tools for industrial use [98; 99]. However, finding optimal trajectories for both systems is still a challenging problem. Even though there is a difference between the degrees of freedom of a manipulator and a mobile platform, most of the algorithms used to find optimal paths for mobile robots can be implemented for mobile manipulators as well.

Optimal control techniques have been widely studied and implemented as an approach to finding optimal paths in mobile robot path planning. Solutions to many optimal control problems for mobile robots cannot be found by analytical means. Over the years, numerical procedures have been developed to solve these systems. Numerical methods deal with the study of quantitative approximations to the solutions of mathematical problems, including consideration of and bounds on the errors involved. Methods for solving differential equations and integrating functions are required for all numerical methods in optimal control.

Choosing a method for solving an optimal control problem depends largely on the type of problem to be solved. Indirect approaches have the advantages of being simple to understand and producing highly accurate solutions when they converge. However, available indirect techniques usually face serious convergence difficulties because of the lack of a good initial guess, as discussed in Chapter 1. The Leapfrog method can be viewed as an indirect solution method. It is relatively simple to implement and works well in practice with no need for an initial guess.

In this chapter, the optimal control problem is described, and the resulting two-point boundary value problem (TPBVP) is derived using the first-order optimality conditions. After that, an overview of indirect methods for solving TPBVP, including an overview of the Leapfrog method, is done. The aim of deriving these systems is to formulate a path planning problem as an optimal control problem for mobile robots, in the chapters to follow.

3.1 Optimal Control

Optimal control theory concerns mathematical optimization methods for deriving control policies. The approach of optimal control allows for the solution of a large class of nonlinear control problems subject to complex state and control signal constraints. The formulation of the problem involves a set of differential equations describing the paths of the control variables that minimize a cost functional. There are various types of optimal control problems, depending on the mathematical description of the system to be controlled, the performance criterion, the different kinds of constraints and the statement of what variables are fixed or free.

3.1.1 Optimal Control Problem

A general optimal control problem consists of finding control state histories for a dynamic system over a period of time to minimize or maximize a cost or performance index. Consider a control system described by ordinary differential equations (ODEs) of the form

$$\dot{\mathbf{q}} = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)), \qquad \mathbf{q}(t_0) = \mathbf{q}_0, \tag{3.1.1}$$

with initial point $\mathbf{q}_0 \in \mathbb{R}^n$, $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and control represented by function $\mathbf{u} : [t_0, t_f] \to \mathbf{u}(t) \in \mathcal{U}$ where $\mathcal{U}$ is the set of all admissible controls. The function $\mathbf{f}$ now depends on the control $\mathbf{u}$ so that the solution $\mathbf{q}(\cdot) : [t_0, t_f] \to \mathbb{R}^n$ is predicated not only on $\mathbf{q}_0$ but on $\mathbf{u}(\cdot)$ as well.

Definition 3.1.1. *A functional J is a rule of correspondence that assigns to each function $\mathbf{q}(t)$ a unique real number [100].*

The cost functional can be written in the form

$$J(\mathbf{u}) = S(\mathbf{q}_f) + \int_{t_0}^{t_f} L(\mathbf{q}(t), \mathbf{u}(t))dt. \tag{3.1.2}$$

The function $S : \mathbb{R}^n \to \mathbb{R}$ represents the terminal cost and $L : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the running cost. The variable t_f is the final (or terminal) time which is either free or fixed and $\mathbf{q}_f = \mathbf{q}(t_f)$ is the final state which is either free or fixed or belongs to some given target set. The cost functional defined above is known as the Bolza problem. If $L(\mathbf{q}(t), \mathbf{u}(t)) = 0$, then the problem is known as the Mayer problem. If $S(\mathbf{q}_f) = 0$, it is known as the Lagrange problem.

The performance index may vary depending on the problem being solved; for example, it can measure control effort, fuel consumption, energy expenditure or the time for the system to reach a target. Examples of these different performance indices are discussed in [20].

Assuming that there are no path constraints on the states or control constraints, and the initial and final times are fixed, a general optimal control problem can be modelled by

$$\begin{aligned} \min_{\mathbf{u}\in\mathcal{U}} \quad & J(\mathbf{u}) \\ & \dot{\mathbf{q}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)), \\ & \mathbf{q}(t_0) = \mathbf{q}_0, \ \mathbf{q}(t_f) = \mathbf{q}_f. \end{aligned} \tag{3.1.3}$$

To solve the optimal control problem in Eq. (3.1.3), necessary conditions for optimality can be obtained through Pontryagin's principle.

3.1.2 Necessary Conditions of Optimality

In this section, a set of necessary conditions for the optimality of a solution of an optimal control problem is derived using the calculus of variations. This set of necessary conditions is also known as Pontryagin's minimum principle PMP)[101]. The method of Lagrange multipliers is used, and the Hamiltonian function and its boundary conditions are derived. The necessary conditions presented in this section are with unconstrained control inputs, meaning that the admissible state and control region are not bounded.

Pontryagin introduced the idea of adjoining functions to append the differential equations to the objective functional. Hence the following definition is considered:

Definition 3.1.1. *The Pontryagin Minimum Principle [101] states that if* $\mathbf{u}^*$ *and* $\mathbf{q}^*$ *are the state and optimal control inputs for Eq. (3.1.3) then there exists a function* $\boldsymbol{\lambda}$*, called the costate, that satisfies a certain minimum principle such that* $H(\mathbf{q}^*(t), \mathbf{u}^*(t), \boldsymbol{\lambda}^*(t)) \leq H(\mathbf{q}^*(t), \mathbf{u}(t), \boldsymbol{\lambda}^*(t))$.

The costate function, also known as the adjoint variable, is a Lagrange multiplier in multivariate calculus. Now considering the optimal control problem in Eq. (3.1.3), the function

$$H(\mathbf{q}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t)) = L(\mathbf{q}(t), \mathbf{u}(t)) + \boldsymbol{\lambda}(t)^T \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) \tag{3.1.4}$$

is called a Hamiltonian function.

Definition 3.1.2. *Any triple* $(\mathbf{q}(\cdot), \mathbf{u}(\cdot), \boldsymbol{\lambda}(\cdot))$ *satisfying the necessary conditions of optimality of Pontryagin's Minimum Principle is called an extremal.*

In literature, the Hamiltonian function in Eq. (3.1.4) is sometimes defined as $H(\mathbf{q}, \mathbf{u}, \boldsymbol{\lambda}) = \boldsymbol{\lambda}^T \mathbf{f} + \lambda_0 L$, where $\lambda_0 \geq 0$ is an additional constant to PMP. A case where $\lambda_0 > 0$ is called a normal extremal and when $\lambda_0 = 0$, the extremal is abnormal. If $\lambda_0 > 0$, then the Hamiltonian can be normalized so that $\lambda_0 = 1$. For the work presented in this dissertation, the normalized extremal is considered.

Assume that the initial time t_0 and final time t_f are fixed. The augmented functional, J_a, for this optimal control problem is obtained by adjoining the constraints imposed by Eq. (3.1.1) and the performance index in Eq. (3.1.3) through the Lagrange multipliers $\lambda_1(t), \dots, \lambda_n(t)$ as:

$$J_a(\mathbf{u}) \quad = \quad S(\mathbf{q}_f) + \int_{t_0}^{t_f} \left\{ H(\mathbf{q}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t)) - \boldsymbol{\lambda}^T(t)\dot{\mathbf{q}} \right\} dt. \tag{3.1.5}$$

For a minimum point to exist, the variation of J_a, δJ_a, must be zero on an extremal. Now consider a variation in $\mathbf{u}(t)$, denoted by $\delta\mathbf{u}$. Such a variation will produce variations in the state $\delta\mathbf{q}$, and a variation in the cost functional δJ_a. This can be expressed as

$$\begin{aligned} \delta J_a \quad = \quad & \left[\left(\frac{\partial S}{\partial \mathbf{q}} \right) \delta\mathbf{q} \right]_{t=t_f} + [\boldsymbol{\lambda}^T \delta\mathbf{q}]_{t=t_0} \qquad (3.1.6) \\ & + \int_{t_0}^{t_f} \left\{ \left(\frac{\partial H}{\partial \mathbf{q}} + \dot{\boldsymbol{\lambda}}^T \right) \delta\mathbf{q} + \left(\frac{\partial H}{\partial \boldsymbol{\lambda}} - \dot{\mathbf{q}}^T \right) \delta\boldsymbol{\lambda} \left(\frac{\partial H}{\partial \mathbf{u}} \right) \delta\mathbf{u} \right\} dt. \end{aligned}$$

From the expression in Eq. (3.1.6), it is observed that the constraints

$$\dot{\mathbf{q}}^*(t) = \mathbf{f}(\mathbf{q}^*(t), \mathbf{u}^*(t)) \tag{3.1.7}$$

must be satisfied by an extremal so that the coefficient of $\delta\boldsymbol{\lambda}(t)$ is zero. Since the Lagrange multipliers are arbitrary, they can be selected to make the coefficients of $\delta\mathbf{q}(t)$ and $\delta\mathbf{q}(t_f)$ equal to zero:

$$\dot{\boldsymbol{\lambda}}(t) \quad = \quad -\frac{\partial H}{\partial \mathbf{q}}, \tag{3.1.8}$$

$$\boldsymbol{\lambda}(t_f) \quad = \quad \left. \frac{\partial S}{\partial \mathbf{q}} \right|_{t=t_f}. \tag{3.1.9}$$

This choice of $\boldsymbol{\lambda}(t)$ results in the following expression for J_a, assuming that the initial state is fixed, so that $\delta\mathbf{q}(t_0) = 0$:

$$\delta J_a \;=\; \int_{t_0}^{t_f} \left\{ \frac{\partial H}{\partial \mathbf{u}} \delta \mathbf{u} \right\} dt. \tag{3.1.10}$$

For a minimum, it is necessary that $\delta J_a = 0$. Hence

$$\frac{\partial H}{\partial \mathbf{u}} = 0. \tag{3.1.11}$$

Equations (3.1.7), (3.1.8), (3.1.9) and (3.1.11) are the necessary conditions of optimality for the minimum of J we set out to determine. Equation (3.1.7) is equivalent to the state equation (3.1.1). Equation (3.1.8) represents the costate equation, and Eq. (3.1.9) and the initial state condition represent the boundary conditions or transversal conditions.

The necessary condition in Eq. (3.1.11) stands because the terminal time t_f is free. If the time t_f is fixed (prescribed), however, the condition in Eq. (3.1.11) is not part of the optimality system anymore. Then the Hamiltonian must be a constant, i.e., $H(\mathbf{q}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t)) = a$, but this condition is usually not needed for computations, resulting in a simpler optimality system.

3.1.3 Boundary Conditions

Using the necessary conditions of optimality derived above, a boundary condition can be derived from Eq. (3.1.9). For this research work, the fixed final time problem will be addressed, hence the following boundary condition is regarded.

Fixed Final Time: When the final time t_f is known.

Fixed Final State: In this case both $\mathbf{q}(t_f)$ and t_f are known, and $\delta\mathbf{q}_f$ and δt_f both equal zero. Applying this to Eq. (3.1.9) yields

$$\mathbf{q}(t_f) = \mathbf{q}_f. \tag{3.1.12}$$

For other cases of the boundary conditions, refer to [20].

Overall, in order to obtain a solution to the optimal control problem, two sets of differential equations whose conditions are given at different times have to be solved simultaneously. Substituting the computed control in Eq. (3.1.11) into the state equation (3.1.7) and costate equation (3.1.8), results in a set of ordinary differential equations where the functions $\mathbf{q}(t)$ and $\boldsymbol{\lambda}(t)$ must satisfy the boundary conditions:

$$\begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} H_{\boldsymbol{\lambda}} \\ -H_q \end{bmatrix}, \qquad (3.1.13)$$
$$\mathbf{q}(t_0) = \mathbf{q}_0, \; \mathbf{q}(t_f) = \mathbf{q}_f.$$

Equation (3.1.13) is known as a two-point boundary value problem (TPBVP) where $\mathbf{q}_0$ and $\mathbf{q}_f$ represent boundary conditions.

Now putting all the derivations done in this section together, it can be concluded that in order to apply the PMP the following need to be performed:

(i) Define the Hamiltonian function (H).

(ii) Satisfy the necessary conditions of optimality.

(iii) Solve the resulting TPBVP.

3.2 Indirect Methods For Optimal Control Problems

In most cases, numerical techniques must be employed to solve the TPBVP described above. When solving a TPBVP using numerical techniques, the goal is to iteratively solve the given differential equations while they conform to the set of boundary conditions. General methods to solve the TPBVP are discussed in Section 1.1.4 which include the indirect methods. In the indirect methods, the required solution for an optimal control problem is obtained by numerically solving equations (3.1.7) to (3.1.11) which are the optimality conditions. Advantages of indirect methods are that they converge to accurate solutions, they converge quickly near the optimal solution and they are reliable [62]. The disadvantage of indirect methods is the requirement of a good initial guess for the adjoint states. If the initial guess is too far away from the optimal solution, the numerical solution of the TPBVP will, in general, fail to converge. The indirect methods are known to converge to accurate solutions where numerical solutions of the TPBVP are performed by shooting or collocation techniques.

3.2.1 Shooting Methods

The shooting method [102] is based on the repeated solution of initial value problems for different initial values. The initial values are iteratively updated until the boundary conditions are satisfied. Consider a boundary value problem defined by the system of ordinary differential equations (ODEs):

$$\dot{\mathbf{y}}(t) = \mathbf{f}(\mathbf{y}(t), t), \quad t_0 \le t \le t_f \qquad (3.2.1)$$
$$\Phi(y(t_0), y(t_f)) = 0,$$

where $\mathbf{y} = [\mathbf{q}\ \boldsymbol{\lambda}]^T \in \mathbb{R}^n$ is the augmented vector of states and $\Phi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is the value of the solution at $t = t_f$. The shooting methods solve a boundary value problem in Eq. (3.2.1) by transforming it to an initial value problem (IVP):

$$\begin{aligned} \dot{\mathbf{y}}(t) &= \mathbf{f}(\mathbf{y}(t), t), \quad t_0 \le t \le t_f \\ \mathbf{y}(t_0) &= s \end{aligned} \tag{3.2.2}$$

by attempting to find a correct initial condition $y(t_0) = s$ where s is the prescribed initial vector which leads to an approximate value that satisfies the boundary conditions. If the solution to Eq. (3.2.2) with parameter s is denoted as $y(t; s)$, then solving Eq. (3.2.1) is reduced to finding a solution s of the nonlinear system of equations

$$F(s) = \Phi(s, y(t_f; s)) = 0. \tag{3.2.3}$$

This problem is solved iteratively, where on each iteration, $\Phi(s, y(t_f; s))$ must be evaluated for some s. In order to evaluate the shooting method, the steps presented in Algorithm 2 are used to update the initial condition for the costate variables.

Algorithm 2 Shooting Method

```
1: Choose an initial guess $s_0$ and set $k = 0$.
2: Solve the system in Eq. (3.2.2) with initial state $y(t_0) = s^{(k)}$. Compute
       $F(s^{(k)}) = \Phi(s^{(k)}, y(t_f; s^{(k)})),$
   and the Jacobian matrix
       $\dot{F}(s^{(k)}) = \Phi_{y_0}(s^{(k)}, y(t_f; s^{(k)})) + \dot{\Phi}_{y_f}(s^{(k)}, y(t_f; s^{(k)}))\, S(t_f),$
   where $S(t_f) = \frac{\partial y}{\partial s}(t_f; s^{(k)})$.
3: If $\| F(s^{(k)}) \| \approx 0$,   STOP.
4: Compute the Newton direction $D^{(k)}$ from the linear equation
       $\dot{F}(s^{(k)}) D^{(k)} = -F(s^{(k)}).$
5: Set $s^{(k+1)} = s^{(k)} + D^{(k)}$, $k = k + 1$, and go to Step 2.
```

Even though the shooting method is simple and reliable, it has some limitations. The main drawback with indirect shooting methods is finding an initial estimate of the initial costate variables $y(t_f; s^{(k)})$ that produce a solution reasonably close to the specified conditions $s^{(k)}$ at final time. The shooting method gives poor convergence if no good initial guess is available or if the IVP becomes unstable. Also, it suffers from slow convergence when the initial condition is far from the corresponding terminal condition. To overcome this problem, the trajectory must be split up into sub-intervals, and one must apply the same shooting method for each sub-interval, which results in the method of multiple-shooting.

The concept of the multiple-shooting method was first proposed in [103] and later promoted in [67]. The aim of the multiple-shooting method is to solve a nonlinear equation where the unknowns are not only the initial costate variables and the final time, as in the shooting method, but also the paired states. In multiple-shooting the interval $t \in [t_0, t_f]$ is divided into n sub-intervals $[t_i, t_{i+1}], i = 0, \dots, n-1$, such that

$$t_0 < t_1 < \cdots < t_n = t_f.$$

Let $y_i(t; s_i)$ be the solution of the IVP

$$\dot{\mathbf{y}}(t) = \mathbf{f}(\mathbf{y}(t), t), \quad \mathbf{y}(t_i) = s_i, \tag{3.2.4}$$

in each sub-interval $t \in [t_i, t_{i+1}]$, $i = 0, 1, \dots, n-1$. Then formulate the set of equations

$$F(s) = \begin{bmatrix} y_0(t_1; s_0) - s_1 \\ y_1(t_2; s_1) - s_2 \\ \cdot \\ \cdot \\ \cdot \\ y_{n-2}(t_{n-1}; s_{n-2}) - s_{n-1} \\ \Phi(s_0, y_{n-1}(t_n; s_{n-1})) \end{bmatrix} = 0, \tag{3.2.5}$$

and solve them by Newton's method. The outline of the multiple-shooting method is given in Algorithm 3. The method is also illustrated in Figure 3.1. The multiple-shooting method is a very accurate method and often performs better than the single shooting method [62]. However, it still requires good initial guesses in the sub-intervals taken to converge to an optimal solution. Also, the number of parameters to be updated in each iteration can be very large, leading to larger computation times when compared to the single shooting (provided that it converges).

A similar iterative method to multiple-shooting, namely the Leapfrog method, was developed to solve a special type of TPBVP arising in geodesics. In Leapfrog a feasible path with corresponding time interval is subdivided, and local optimal controls are found separately over each sub-interval. An optimal local trajectory is obtained in each sub-interval, where the junctions of these

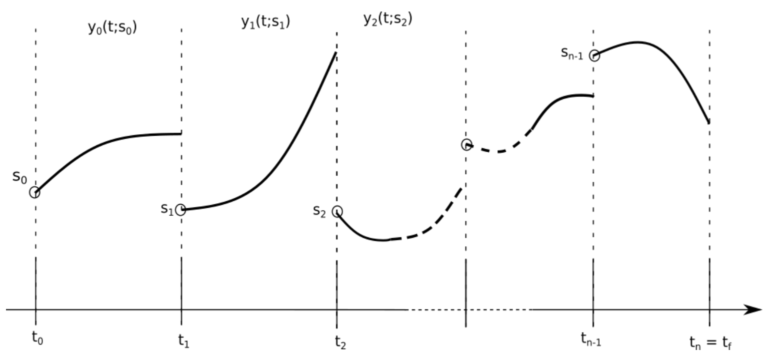


Figure 3.1: Multiple-shooting method [3].

Algorithm 3 Multiple-Shooting Method

```
1: Choose an initial guess $s^{(0)} = (s_0^{(0)}, \dots, s_{n-1}^{(0)})^T \in \mathbb{R}^{N \times n}$ and $k = 0$.
2: For $i = 0, \dots, n-1$ solve the IVPs
       $\dot{y}_i(t) = \mathbf{f}(\mathbf{y}_i(t), t), \quad y_i(t_i) = s_i^{(k)}$
   and the sensitivity differential equation
       $\dot{S}_i(t) = \dot{\Phi}_q(y_i(t), t)\, S_i(t), \quad S_i(t_i) = I \cdot D,$
   in $t \in [t_i, t_{i+1}]$, where the derivative of $\Phi$ is evaluated at $y_i$.
   Compute $F(s^{(k)})$ and $\dot{F}(s^{(k)})$.
3: If $\| F(s^{(k)}) \| \approx 0$, STOP.
4: Compute the Newton direction $D^{(k)}$ from the linear equation
       $\dot{F}(s^{(k)})\, D^{(k)} = -F(s^{(k)}).$
5: Set $s^{(k+1)} = s^{(k)} + D^{(k)}$, $k = k + 1$, and go to Step 2.
```

sub-trajectories are updated through a scheme of midpoint maps. The advantage of using the Leapfrog method is that it does not depend on the provision of good initial guesses along a path. Also, the optimal solution provided by the method satisfies both boundary conditions at every step, while in multiple-shooting the boundary conditions are satisfied after the whole iteration process is completed. Moreover, the Leapfrog paths are feasible in each iteration.

For the purpose of this research work, the Leapfrog method is chosen to solve the TPBVP for mobile robot path planning and is compared to BVP4C. The next section gives an overview of the Leapfrog method and the BVP4C solver.

3.2.2 The Leapfrog Method

In this section, the Leapfrog method is reviewed, and its convergence proof is given following the notation, terminology and lines of argument as in [4]. The Leapfrog method, proposed by Kaya and Noakes in [73], can be viewed as an indirect method for solving TPBVP. In the method, a feasible path is subdivided, and optimal local paths are found separately over each sub-interval. The method works in the following manner. Consider the following optimal control problem in Lagrange form:

$$\mathcal{X} = \begin{cases} \min\limits_{\mathbf{u}\in\mathcal{U}} \displaystyle\int_{t_0}^{t_f} L(\mathbf{q}(t), \mathbf{u}(t))\, dt, \\ \dot{\mathbf{q}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)), \\ \mathbf{q}(t_0) = \mathbf{q}_0,\ \mathbf{q}(t_f) = \mathbf{q}_f \end{cases}$$

with the state $\mathbf{q}(t) : \mathbb{R} \to \mathbb{R}^n$ and control $\mathbf{u}(t) : \mathbb{R} \to \mathbb{R}^m$. The function $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is C^1-continuous and t_0 and t_f represent the initial time and final time, respectively. A solution to $\mathcal{X}$ is given by an optimal state trajectory $\mathbf{q}(\cdot)$ and a corresponding control input $\mathbf{u}(\cdot)$. Here, we assume that the problem $\mathcal{X}$ has no constraints on the state and control, and the final time t_f is fixed.

Let μ_z be a piecewise path between the start position $z(t_0) = \mathbf{q}_0$ and terminal position $z(t_f) = \mathbf{q}_f$. The path μ_z is required to be feasible but not necessarily optimal, and the points $\mathbf{q}_0$ and $\mathbf{q}_f$ are not necessarily close to each other either. The aim is to find a critical trajectory between $\mathbf{q}_0$ and $\mathbf{q}_f$.

Definition 3.2.1. *If the trajectory pair* $(\mathbf{q}(\cdot), \boldsymbol{\lambda}(\cdot))$ *corresponding to control* $\mathbf{u}(\cdot)$ *satisfies the optimality system in Eq. (3.1.7)-(3.1.11), then it is said to be a critical trajectory.*

The Leapfrog method begins with partitioning the feasible path μ_z into p segments, with

$$z_0, z_1, \ldots, z_{p-1}, z_p$$

as subdivision points with corresponding partition time points

$$t_0 < t_1 < \cdots < t_p = t_f.$$

The partition points are chosen to be closer to each other such that a locally optimal solution exists from point z_{i-1} to z_{i+1}, $i = 1, \ldots, p-1$, which is easily computable.

Now, let us assume that the local solutions to the following sub-problem $\mathcal{X}_i$ can be obtained:

$$\mathcal{X}_i = \begin{cases} \min\limits_{u\in A} \displaystyle\int_{t_{i-1}}^{t_{i+1}} L_0(\mathbf{q}(t), \mathbf{u}(t))\, dt, \\ \dot{\mathbf{q}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)), \\ \mathbf{q}(t_{i-1}) = z_{i-1},\ \mathbf{q}(t_{i+1}) = z_{i+1}, \end{cases}$$

where t_{i-1} is fixed by the solution of $\mathcal{X}_{i-1}$ for $i > 1$ and t_{i+1} is fixed. The sub-problem $\mathcal{X}_i$ can be written as the optimality system associated with t for $t \in [t_{i-1}, t_{i+1}]$, with end points $q(t_{i-1}) = z_{i-1}$ and $q(t_{i+1}) = z_{i+1}$. Through the optimality conditions an initial value of the costate variable is given by $\phi_{i-1} = \lambda(t_{i-1})$ for problem $\mathcal{X}_i$.

The Leapfrog method relies on a solution of $\mathcal{X}_i$ through a given partition where optimal controls are obtained. Subsequently, the method follows an iterative process of updating the initial feasible path μ_z towards the optimal trajectory while solving the sub-problems $\mathcal{X}_i$. In each iteration, k, a local solution between partition points is solved for $i = 1, \dots, p-1$ and the partition point z_i is adjusted through a midpoint scheme.

Assumption 3.2.1. *There exists a (locally) unique optimal control* $\mathbf{u} : [t_{i-1}, t_{i+1}] \to \mathbb{R}^m$ *taking the system from* z_{i-1} *to* z_{i+1}.

Let $\mathcal{D}$ denote the set of all ordered pairs (z_{i-1}, z_{i+1}) such that z_{i+1} can be reached from z_{i-1} with cost less than 2δ,

$$\mathcal{D} = \{(z_{i-1}, z_{i+1}) \in \mathbb{R}^n \times \mathbb{R}^n : d(z_{i-1}, z_{i+1}) \leq 2\delta\}, \tag{3.2.6}$$

where d is the distance function and $\delta > 0$ is chosen such that there exists a local optimal control from z_{i-1} to z_{i+1}. Then, let the local cost function be denoted by

$$\tau : \mathcal{D} \to \mathbb{R}, \tag{3.2.7}$$

such that, given (z_{i-1}, z_{i+1}) in $\mathcal{D}$, $\tau(z_{i-1}, z_{i+1})$ is the infimum of the cost to get from z_{i-1} to z_{i+1}.

For $(z_{i-1}, z_{i+1}) \in \mathcal{D}$, using Assumption 3.2.1, let $\gamma_{z_{i-1}, z_{i+1}} : [t_{i-1}, t_{i+1}] \to \mathbb{R}^n$ be the unique local optimal trajectory such that $\gamma_{z_{i-1}, z_{i+1}}(t_{i-1}) = z_{i-1}$ and $\gamma_{z_{i-1}, z_{i+1}}(t_{i+1}) = z_{i+1}$. The midpoint $\hat{z}_i = \gamma_{z_{i-1}, z_{i+1}}(\hat{t}_i)$, between z_{i-1} and z_{i+1} is then defined by $\tau(z_{i-1}, \hat{z}_i) = \tau(z_{i-1}, z_{i+1})/2$.

To update each z_i, $i = 1, \dots, p-1$ and t_i, $i = 1, \dots, p$, set $z_0 = \mathbf{q}_0$ and $z_p = \mathbf{q}_f$. Then move z_i onto a local optimal path joining z_{i-1} and z_{i+1} obtained by solving problem $\mathcal{X}_i$. The updated point where z_i goes in the optimal path segment is taken to be the point such that the cost of getting from z_{i-1} to that point is half the cost of getting from z_{i-1} to z_{i+1} and is reached at the time average of t_{i-1} and t_{i+1}.

Remark 3.2.1. *A different position of* z_i^k *on the local optimal path is allowed.*

Remark 3.2.2. *For every update on* z_i^k, t_i^k, *the costates* ϕ_i *are also updated.*

The midpoint scheme is followed for all partition points so that all z_i^k and t_i^k are updated, meaning that one iteration, $k = 1$, of the Leapfrog is done and joining the optimal local paths between the partition points denotes $\mu_z^{(1)}$. Then the iterations are continued until the maximum number of iterations, N, is reached and Leapfrog convergences. The outline of the Leapfrog method is given in Algorithm 4.

Algorithm 4 Leapfrog Method

```
1: Initialize feasible trajectory $\mu_z^{(0)}$ with $z_0 = \mathbf{q}_0$ and $z_p = \mathbf{q}_f$.
2: Choose the number of subdivisions $p > 0$ and final time $t_f > t_0$.
3: Subdivide $\mu_z^{(0)}$ into $p$ segments with subdivision points
   $z_0^{(0)}, z_1^{(0)}, \dots, z_{p-1}^{(0)}, z_p^{(0)}$. Set $t_0^{(0)}$ and $k = 0$.
4: for $k = 1 : N$ do
5:     for $i = 1 : p - 1$ do
6:         Given $z_{i-1}^k$, $z_{i+1}^k$, $t_{i-1}^k$, and $t_{i+1}^k$. Solve the sub-problem $(\mathcal{X}_i)$ with
           $\mathbf{q}(t_{i-1}) = z_{i-1}^{(k)}$, $\mathbf{q}(t_{i+1}) = z_{i+1}^{(k)}$, $t_{i-1} = t_{i-1}^k$, and $t_{i+1} = t_{i+1}^k$.
7:         Let $z_i^{k+1} = \mathbf{q}(t_i)$ and $t_i^{k+1} = t_i$ such that $\tau(\mathbf{q}(t_{i-1}), \mathbf{q}(t_i)) = \frac{1}{2}\tau(\mathbf{q}(t_{i-1}), \mathbf{q}(t_{i+1}))$.
8:         Set $k = k + 1$
9:     end for
10: end for
11: return $\mu_z^k$.
```

During the Leapfrog iterations, the number of partition segments p is progressively decreased as the algorithm is executed. The aim is to reduce the segments to $p = 2$ for the final iteration. This means that the algorithm ends with solving problem $\mathcal{X}$ between the initial and final states z_0 and z_f. If p is decreased appropriately, the convergence to a critical trajectory is ensured. But, it might not be globally optimal, and could be a local minimum. The method also assures that the cost for trajectory $\mu_z^{(k)}$ decreases on each iteration, k. In addition, the trajectory satisfies both boundary conditions at every step of the iterations where the initial point z_0 and final point z_f are fixed as they denote the boundary conditions. Moreover, the trajectory $\mu_z^{(k)}$ generated in each iteration feasible, as illustrated in Figure 3.2.

3.2.2.1 Initial Feasible Path

Generally, it is far easier to obtain a feasible trajectory than an optimal trajectory. Starting with a feasible trajectory also ensures controllability between any two states considered along with the feasible trajectory after partitioning. In previous work [4; 74], the initial partition for the Leapfrog method is chosen

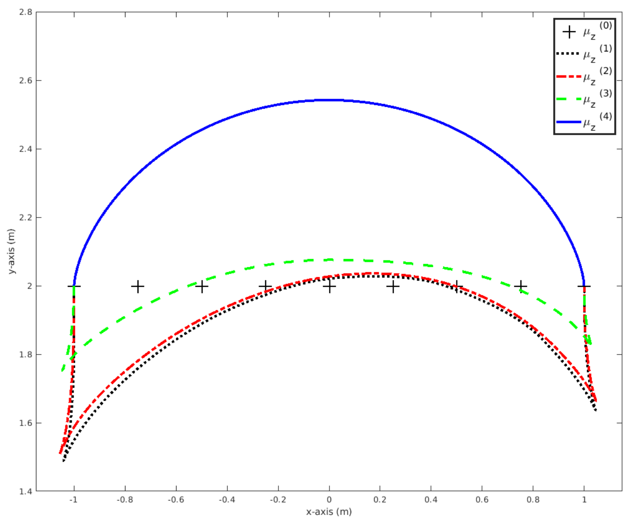


Figure 3.2: Feasible trajectories formed in each iteration of the Leapfrog method. The initial feasible path is represented by (+) and the final solution is represented by $\mu_z^{(4)}$.

as a straight line between the initial and final states. This approach is easy to implement, and will often work well, especially for simple boundary value problems. However, if one considers a path planning problem in the presence of obstacles, a different initialization technique is needed as Leapfrog requires a feasible (collision-free) initial path.

3.2.2.2 Initial Guess

In solving a TPBVP the physical information regarding the costates is not known, and only the boundary conditions for the state are known. Hence with indirect numerical methods, a good initial guess is required for a solution to be obtained. With the Leapfrog method, however, once a feasible path is constructed, the affine approximation of the sub-problem $\mathcal{X}_i$ in a subdivision may provide a good initial guess needed for simple shooting in that subdivision. Hence Leapfrog does not depend on the provision of a good initial guess. Even though an initial guess of the costates is not crucial in the Leapfrog method, the guesses are needed for the success of the local shooting method. Also, the algorithm is convergent to a critical trajectory provided that these local shooting solutions produce optimal trajectories.

3.2.2.3 Overview of Leapfrog Convergence

The main computation in the Leapfrog method depends on solving the sub-problem $\mathcal{X}_i$. In each subdivision, with the assumption that local optimal controls can easily be calculated, a piecewise optimal trajectory is obtained. Then the junctions of these smaller pieces of optimal control trajectories are updated through a scheme of midpoint maps. It was shown in [4] that under some assumptions the sequence of trajectories converges to a critical trajectory that satisfies the minimum principle. The proof of the Leapfrog convergence under such assumptions is also given. In this section, the Leapfrog convergence proof is reviewed by following the notation, terminology and lines of argument in [4].

As mentioned before, the aim of the Leapfrog method is to find a critical path from an initial state to a final state. For proving the convergence to a critical path, it is necessary to consider the local critical trajectory pair $(\mathbf{q}(\cdot), \boldsymbol{\lambda}(\cdot))$ obtained by solution of the sub-problem $\mathcal{X}_i$. What follows are the tools and definitions that lead to the convergence to a critical path.

(i) **Extremes and Leapfrog splicing**

Recall by Assumption 3.2.1 that $\gamma_{z_{i-1},z_{i+1}} : [t_{i-1}, t_{i+1}] \to \mathbb{R}^n$ is the unique optimal path from z_{i-1} to z_{i+1}. Now let $y_i = (z_i, \phi_i) = (\mathbf{q}(t_i), \boldsymbol{\lambda}(t_i))$ be between y_{i-1} and y_{i+1}, if z_i lies in $\gamma_{z_{i-1},z_{i+1}}(\cdot)$. Let Y be a set of all

$$y = (y_0, y_1, \dots, y_q),$$

then $y \in Y$ is extreme when y_i is between y_{i-1} and y_{i+1} for all $1 < i < p$. Let $\mu_i(\cdot) = (\gamma_{z_{i-1},z_i}(\cdot), \boldsymbol{\lambda}(\cdot))$ be the optimal trajectory pair from z_{i-1} to z_i, $i = 1, \dots, p$. Then μ_y is defined as the concatenation of $\mu_1, \mu_2, \dots, \mu_p$ in the given order such that $\mu_y(t_0) = \mu_1(t_0), \mu_y(t_i) = \mu_i(t_i) = \mu_{i+1}(t_i)$, $i = 1, \dots, p-1$, and $\mu_y(t_f) = \mu_p(t_p)$.

Lemma 3.2.1. *If y is extreme, then μ_y is a critical trajectory.*

Proof. Suppose that μ_y is a critical trajectory. Then by Assumption 3.2.1, the path from y_{i-1} to y_{i+1} is optimal which implies that y_i is between y_{i-1} and y_{i+1}. Therefore y is extreme.

Suppose that y is extreme, then using induction we prove that μ_y is critical. Suppose that $p = 2$, then $\mu_y = \gamma_1$. Assume that the lemma holds for $p \le k$ and μ_y^k is composed by concatenating $\mu_i, i = 1, \dots, k$. Let $p = k + 1$. The next Leapfrog step from y_{k-1} to y_{k+1} gives the local optimal path segment $\gamma_{z_{k-1},z_{k+1}}$. Note that μ_y^k is critical, i.e., it satisfies the PMP. Parts of the curves μ_k and μ_y^k between y_{k-1} and y_k overlap (see Figure 3.3), so both curves satisfy the optimality system associated with the optimal control problem $\mathcal{X}$ with the end conditions $q(t_0) = z_0$ and $q(t_{k+1}) = z_{k+1}$ for that portion of the curve and thus satisfy the PMP. Therefore μ_y^{k+1}, which is the concatenation of $\mu_i, i = 1, \dots, k + 1$ is a critical trajectory. This completes the proof. □

(ii) **Midpoint maps and total cost**

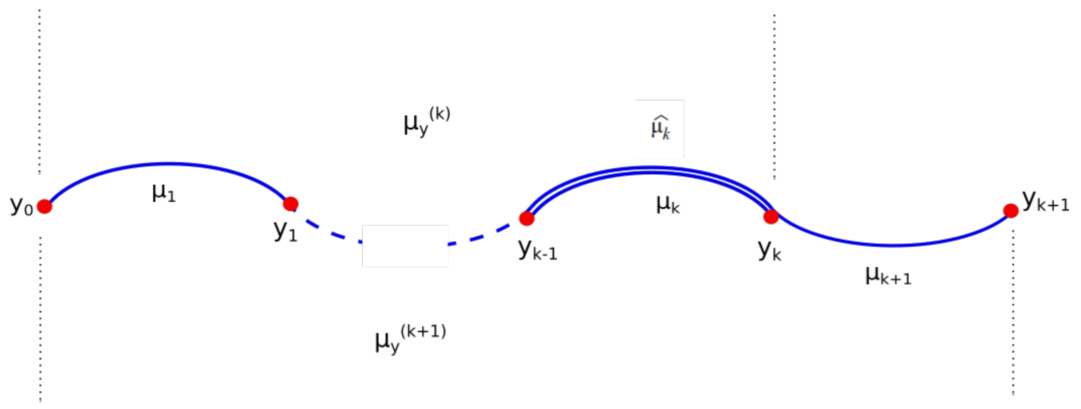


Figure 3.3: Concatenation of the local optimal path μ_i and the Leapfrog splicing [4].

Define a midpoint map to be any $\mathcal{M} : \mathcal{D} \to \mathbb{R}^2$ such that $\mathcal{M}(z_{i-1}, z_{i+1}) = \hat{z}_i$. Recall $\hat{z}_i$ is the midpoint between z_{i-1} and z_{i+1}.

For the state-costate space, let $X \subseteq \mathbb{R}^{2n}$ and Y be the set of all $(p+1)$-tuples $y = (y_0, y_1, \ldots, y_q) \in X^{q+1}$ such that $y_i = (z_i, \phi_i)$ and $\tau(z_i, z_{i+1}) \leq \delta$ for all $i = 1, \ldots, p-1$.

For $1 \leq m \leq p$ define the midpoint iteration $G_m : Y \to X^{p+1}$ such that

$$G_m(y) = (y_0, \ldots, y_{m-1}, \hat{y}_m, y_{m+1}, \ldots, y_p),$$

where $\hat{y}_m = (\hat{z}_m, \hat{\lambda}_m)$, $\hat{z}_m = \mathcal{M}(z_{m-1}, z_{m+1})$ and $\hat{\lambda}_m$ is given by one or two cases (see [4]).

Lemma 3.2.2. $G_g(y) \in Y$.

Thus $G_m : Y \to Y$ where $1 < m < p$. Define a function $F : Y \to Y$ as the composite

$$F = G_{p-1} \circ G_{p-2} \circ \cdots \circ G_1,$$

then $F(y) \in Y$.

The $(p+1)$-tuple z is said to have cost $\alpha(z)$ defined by

$$\alpha(z) = \sum_{i=1}^{p} \tau(z_{i-1}, z_i)$$

and hence

$$\tau(z_0, z_p) \leq \alpha(z).$$

The function α is continuous, because τ is continuous (refer to [4] for proof).

Lemma 3.2.3. *For $1 \leq m \leq p$ and all $y \in Y$,*

$$\alpha(G_m(y)) \leq \alpha(y).$$

If one considers the composition $F = G_{m-1} \circ G_{m-2} \circ \cdots \circ G_1$ in Lemma 3.2.3 instead of G_m, then the following lemma is obtained.

Lemma 3.2.4. *For all $y \in Y$,*

$$\alpha(F(y)) \leq \alpha(y).$$

The following lemma results from Lemma 3.2.4.

(iii) **Convergence to a critical trajectory**

Lemma 3.2.5. *(i) If μ_y is a critical trajectory then $\alpha(F(y)) = \alpha(y)$.*

(ii) If $\alpha(F(y)) = \alpha(y)$ then μ_y is a critical trajectory.

The following theorem follows from Lemma 3.2.4 and Lemma 3.2.5.

Theorem 3.2.1. *The Leapfrog iterations converge to a critical trajectory $\mu_{y^{(\infty)}}$.*

Proof. Note that $F(y^{(\infty)}) = (y^{(\infty)})$. The theorem follows Lemma 3.2.5, then

$$\alpha(F(y^{(\infty)})) = \alpha(y^{(\infty)}).$$

□

A detailed proof of the lemmas described in this section can be found in [4].

3.2.3 Collocation Method - BVP4C

Collocation methods, on the other hand, use cubic polynomials to obtain an approximate solution of TPBVP. Collocation methods are usually applied to handle boundary value problems which give unstable initial value problems [1]. A commonly used implementation of a collocation method is the MATLAB solver BVP4C. The function BVP4C divides the time interval $[t_0, t_f]$ in sub-intervals and discretizes differential equations along the time mesh. To initialize the solver, an initial mesh and a guess of the solution at the mesh points are required. The mesh is adapted to obtain the specified accuracy of the solution for an almost minimal number of mesh points.

The MATLAB solver BVP4C is a built-in function which provides a means for rapid construction and solution of TPBVP. The solver uses a three-point Lobatto method of order four called the Simpson formula because it reduces to Simpson's quadrature rule [104]. This method uses a collocation formula, and the collocation polynomial provides a C^1-continuous solution that is fourth-order accurate uniformly over the interval. In [104], BVP4C is described to solve boundary value problems for ordinary differential equations efficiently. However, the higher the complexity and nonlinearity of the dynamics, the more the difficulty with the solver. Also, the approach requires considerable assistance in selecting a good initial guess for the states and costate trajectories in addition to the terminal time for the problem. In motion planning for wheeled mobile robots, BVP4C solves a TPBVP resulting from deriving the necessary conditions for the optimal point-to-point motion planning for the robot.

This approach has been used as an indirect method to obtain optimal trajectories for two-wheeled mobile robots and mobile manipulators [25; 105; 106]. In [25] initial guesses for the angular positions and velocities were considered to be zero and a uniform time mesh with 200 points was considered. The capability of the method was demonstrated through simulation and experimental results. The method was seen to be efficient in finding paths for the mobile robot in the presence of obstacles. To understand how BVP4C can be used, the following describes the method as it can be used in MATLAB.

The function BVP4C integrates a system of ODEs of the form

$$\frac{d\mathbf{y}}{dt} = f(\mathbf{q}, \mathbf{y}, \mathbf{p}) \tag{3.2.8}$$

subject to a general nonlinear TPBVP condition

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b), \mathbf{p}) = 0, \tag{3.2.9}$$

where $\mathbf{q}$ is the independent variable, $\mathbf{y}$ is a vector containing the dependent variables, and $\mathbf{p}$ is a vector of unknown parameters.

For this research work, the solver operates on the assumption that the time period in the problem is fixed, $t \in [a, b]$. The boundary value solver BVP4C requires three pieces of information: the equation to be solved, its associated boundary conditions, and an initial guess for the solution.

Remark 3.2.3. *The number of ordinary differential equations must equal the number of boundary conditions such that the problem is solvable.*

The MATLAB syntax of the BVP4C function is as follows:

```
sol = bvp4c(@odefun,@bcfun,solinit,options);        (3.2.10)
```

From above,

- `(odefun)` is a function that evaluates the TPBVP in the form of a system of first-order differential equations which can be expressed as

  ```
  function dydx = odefun(q,y);        (3.2.11)
  ```

  where `q` is a scalar, and `dydx` and `y` are column vectors.

- `(bcfun)` computes the residual in the boundary conditions and takes the form

  `function res = bcfun(y(a),y(b));` (3.2.12)

- `solinit = bvpinit(t,xinit,params);` is a structure that contains the initial guesses for the solution, `t` represents the points at which the boundary conditions are imposed, `xinit` provides the initial guesses to the solver, and `params` gives an initial guess for unknown parameters.

- A list of `options` such as error tolerance, maximum number of mesh points allowed `(Nmax)` and display of computation cost statistics is offered. The list can be supplied to the solver using `bvpset` function

  `options = bvpset('RelTol',10`$^{-6}$`,'Nmax',5000,'Stats','off');`

What is produced by BVP4C is a data structure `sol` containing the solution and has the following fields:

`sol.q` is a mesh selected by BVP4C.

`sol.y` is a solution computed at the mesh points of `sol.q`.

For more detailed information, refer to the MATLAB help file for BVP4C.

Remark 3.2.4. *The convergence of a solution for BVP4C relies on a good initial guess. A bad initial guess may result in inaccurate solutions, or no solutions, or sub-optimal solutions.*

3.3 Conclusion

In this chapter, a general optimal control problem was defined, and optimality conditions were derived using the help of variation techniques and PMP. The optimality conditions resulted in a TPBVP. An overview of indirect methods which generally solve the TPBVP was given which included the shooting-based methods, the Leapfrog method and BVP4C.

Chapter 4

Leapfrog Paths for a Two-wheeled Mobile Robot

In this chapter, an optimal control strategy for optimal motion planning for a two-wheeled mobile robot is presented. This is achieved by formulating the path planning problem of the mobile robot as an optimal control problem. The kinematic model for two-wheeled mobile robots described in Chapter 1 is considered. Its necessary conditions of optimality are derived through PMP and the resulting TPBVP is solved using the Leapfrog method. In previous work, the Leapfrog method has been used within the context of control systems, that is, to generate control inputs for a system. The work presented in this chapter is the first application of the Leapfrog method to path planning for mobile robots, which can also be found in [69]. To evaluate the effectiveness and capability of the Leapfrog method, four different test cases are numerically solved and compared to BVP4C solutions. The numerical simulations are then used for experimental studies through path following control.

4.1 Optimal Control Problem Formulation

Given the mobile robot kinematic model in Eq. (1.2.1) and boundary conditions $\mathbf{q}_0 = (\mathbf{x}_0, \mathbf{y}_0, \varphi_0)$ and $\mathbf{q}_f = (\mathbf{x}_f, \mathbf{y}_f, \varphi_f)$, find a control $\boldsymbol{u}(t)$ that generates the mobile robot's trajectory by minimizing the following objective function:

$$J = \frac{1}{2}\int_{t_0}^{t_f} \boldsymbol{u}^T \boldsymbol{u}\, dt, \tag{4.1.1}$$

where $\boldsymbol{u} = [v, \omega]^T$ is a control input represented by the linear and angular velocities of the robot, respectively.

In order to minimize the cost function, the necessary conditions must be satisfied. Firstly, the Hamiltonian function is defined as

$$H = \frac{1}{2}(v^2 + \omega^2) + \lambda_1 v \cos\varphi + \lambda_2 v \sin\varphi + \lambda_3 \omega, \tag{4.1.2}$$

where λ_1, λ_2, and λ_3 represent the costate variables.

Using $\dot{\boldsymbol{\lambda}} = -\dfrac{\partial H}{\partial \mathbf{q}}$, the co-state equations are obtained as

$$\dot{\boldsymbol{\lambda}} \;=\; \begin{bmatrix} 0 \\ 0 \\ \lambda_1 v \sin\varphi - \lambda_2 v \cos\varphi \end{bmatrix}, \tag{4.1.3}$$

and the state equations from $\dot{\mathbf{q}} = \dfrac{\partial H}{\partial \lambda}$, as

$$\dot{\mathbf{q}} \;=\; \begin{bmatrix} v\cos\varphi \\ v\sin\varphi \\ \omega \end{bmatrix}. \tag{4.1.4}$$

For the control to be optimal, the necessary condition $\dfrac{\partial H}{\partial \mathbf{u}} = 0$ leads to

$$0 = \begin{bmatrix} H_v \\ H_\omega \end{bmatrix} = \begin{bmatrix} v + \lambda_1\cos\varphi + \lambda_2\sin\varphi \\ \omega + \lambda_3 \end{bmatrix}, \tag{4.1.5}$$

where H_v and H_ω are derivatives with respect to control inputs v and ω. Solving for control inputs v and ω, Eq. (4.1.5) becomes

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} -(\lambda_1\cos\varphi + \lambda_2\sin\varphi) \\ -\lambda_3 \end{bmatrix}. \tag{4.1.6}$$

Substituting Eq. (4.1.6) into the state in Eq. (4.1.4) and costate in Eq. (4.1.3), gives the following two-point boundary value problem (TPBVP):

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \\ \dot{\lambda}_1 \\ \dot{\lambda}_2 \\ \dot{\lambda}_3 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2}\lambda_1(1+\cos 2\varphi) - \frac{1}{2}\lambda_2\sin 2\varphi \\ -\frac{1}{2}\lambda_1\sin 2\varphi - \frac{1}{2}\lambda_2(1-\cos 2\varphi) \\ -\lambda_3 \\ 0 \\ 0 \\ \frac{1}{2}(\lambda_2^2 - \lambda_1^2)\sin 2\varphi + \lambda_1\lambda_2\cos 2\varphi \end{bmatrix}, \tag{4.1.7}$$

with the following initial and boundary conditions imposed on the position and orientation of the robot for $t_0 = 0$ and $t_f = 1$:

$$\mathbf{x}(0) = x_0, \qquad \mathbf{x}(1) = \mathbf{x}_f, \tag{4.1.8}$$
$$\mathbf{y}(0) = y_0, \qquad \mathbf{y}(1) = \mathbf{y}_f, \tag{4.1.9}$$
$$\varphi(0) = \varphi_0, \qquad \varphi(1) = \varphi_f. \tag{4.1.10}$$

4.2 Numerical Solutions

In this section, the Leapfrog algorithm is applied to find numerical solutions to the TPBVP derived in Section 4.1. The simulations were performed using MATLAB [107] and a MATLAB code [108] by C.Y. Kaya was modified to suit the mobile robot problem. Four different examples are considered in order to evaluate the Leapfrog algorithm and are compared to numerical solutions using the MATLAB solver BVP4C. This solver which comes standard with MATLAB is commonly used to solve optimal control problems for mobile robots such as in [109], [106], and [110], and it has shown to solve these systems efficiently. Here, the BVP4C solution is used as a reference or ground truth and compared to the Leapfrog solution.

To implement the Leapfrog method, first a feasible path defined as $\mathbf{z}$, is chosen between the robot's initial and final configurations $\mathbf{q}_0$ and $\mathbf{q}_f$. The feasible path is then subdivided into p equally spaced partition points along a straight line. In this case, the feasible path was easy to generate because there are no obstacles in the robot workspace. An affine approximation, which is part of the Leapfrog method, then uses the information from the initial feasible path to compute initial conditions for the costates. For the Leapfrog simulations, four maximum iterations, i.e., $k = 4$, are considered with the feasible path $\mathbf{z}_i^{(k)}$ for $i = 0, 1, \ldots, p$ where $p = 8$.

For the BVP4C implementation the initial mesh between the time interval $[t_0, t_f]$ was chosen to be: `solinit = bvpinit(linspace(0,`t_f`,N),yinit)` with $N = 8$ equally spaced points, where the final time is set to $t_f = 1$ for all cases which is the same as in the Leapfrog method. The initial conditions for the states and costates are stored in `yinit`. If there are no guesses generated for the costates then values in `yinit` are taken as a guess. In the simulations done here, `yinit` is chosen as a default setting for initial condition guess. Once the initial mesh and guesses are selected, the TPBVP and boundary conditions are called and solved numerically.

Even though the examples chosen for this section look simple, the computation for the differential equations for the mobile robot is numerically challenging. The aim for the following cases is to illustrate that the Leapfrog method can find optimal solutions given the robot final configuration to be in an opposite direction as the initial configuration, as demonstrated in the first three examples.

Case 1: In this simulation, the robot is considered to move from the initial state at $[-1, 2, \frac{\pi}{2}]$ to the state at $[1, 2, -\frac{\pi}{2}]$. Figure 4.1 shows the trajectories generated by both methods with arrows indicating computed orientations at key positions. As can be seen from Figure 4.1, the orientation requirement is resolved by travelling in the forward direction.

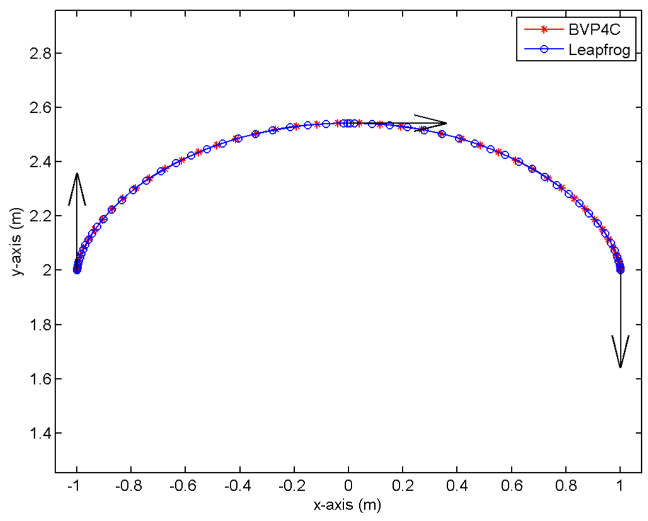


Figure 4.1: The BVP4C and Leapfrog trajectories for Case 1.

Case 2: In this simulation the robot is given the initial state $[-1, 0, 0]$ and desired final state $[1, 0, \pi]$. The solution trajectories can be seen in Figure 4.2 and contain two 90° turns. The first 90° turn is performed in the forward direction while the second turn is in reverse, completing a direction reversal, as desired.

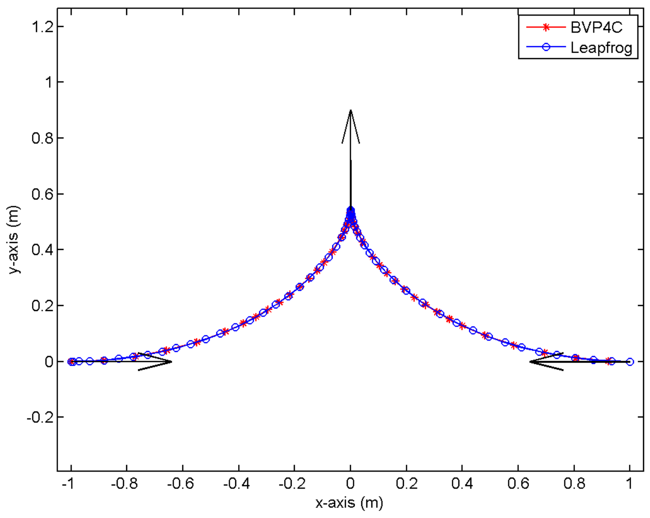


Figure 4.2: The BVP4C and Leapfrog trajectories for Case 2.

Case 3: This case is similar to Case 1, the difference being that the robot is desired to move in the y-direction with orientation reversal. The initial state of the robot is $[0, 3, 0]$ and the final state is $[0, 1, \pi]$. An optimal solution was found using the Leapfrog algorithm and the trajectory can be seen in

Figure 4.3. The orientation requirement is resolved by travelling in the reverse direction.

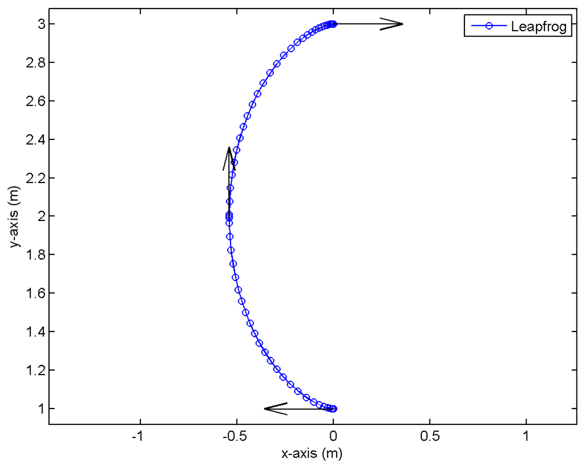


Figure 4.3: The path generated by Leapfrog for Case 3.

On the other hand, the BVP4C solver failed to find a solution for Case 3 and resulted in a state singularity error. To overcome this, a different initial guess for the costate was chosen as $\lambda_{t_0} = [1, 1, 1]^T$. The change resulted in an optimal solution similar to Leapfrog as shown in Figure 4.4. From this example, it was confirmed that the success of BVP4C depends on a good initial guess for the costates. Without a good guess, there is a possibility of getting no solution or a sub-optimal solution.

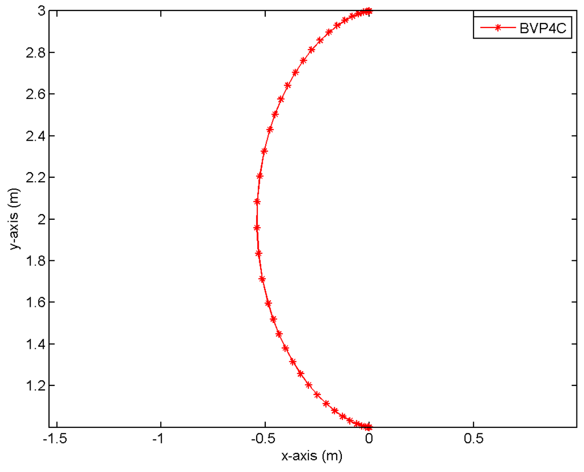


Figure 4.4: The BVP4C trajectory for Case 3 given different values for the initial costates.

Case 4: In this case, the initial state is $[2, 2, 0]$ and the final state $[2, 4, \frac{\pi}{3}]$. The solution trajectories can be seen in Figure 4.5 where the orientation requirement is resolved by travelling in a reverse direction, then a forward direction.

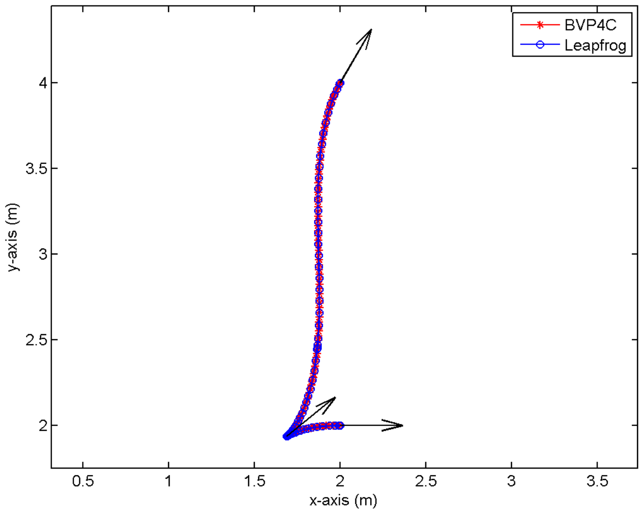


Figure 4.5: BVP4C and Leapfrog trajectories for Case 4.

For comparison, the time of producing the optimal paths from both Leapfrog and BVP4C is demonstrated in Table 4.1 together with the path cost which is noted to be the same for both methods.

Table 4.1: Path cost and CPU time for the simulation results.

Case	Cost	CPU time(s)	
		Leapfrog	BVP4C
1	0.502	1728.62	1.717
2	0.604	35.850	1.775
3	0.502	163.14	1.292
4	0.702	3316.63	2.069

Mean Square Error (MSE) was then calculated for the position coordinates, where Leapfrog results were interpolated using the BVP4C coordinates. The values for the test cases are shown in Table 4.2.

From Table 4.2 it can be noted that the small MSE values indicate high similarity between the trajectories from the BVP4C and Leapfrog methods.

Table 4.2: MSE values for simulation results.

Case	MSE (m^2)
1	1.3895×10^{-9}
2	1.2571×10^{-9}
3	-
4	2.7536×10^{-10}

4.3 Mobile Robot Experiments

For the experimental investigations, a Pioneer 3-DX mobile robot shown in Figure 4.6 is used. This type of mobile robot is a commercial product which is commonly used for research purposes and has shown to have good maneuvering and work well indoors on flat surfaces. The robot has two wheels each attached to a motor, and a caster wheel for balance. The Pioneer 3-DX platform weight

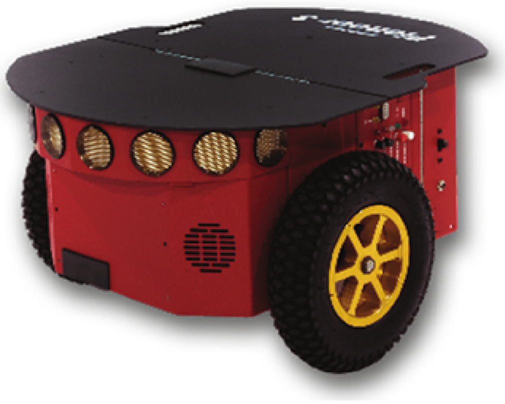


Figure 4.6: Pioneer 3-DX mobile robot [5].

is 9kg, can reach maximum forward or backward speed of 1.6 meters per second and carry a payload of up to 23 kg. In order for the user to be able to control the robot, it has an onboard PC, WiFi adapter for wireless connection and also has a standard serial cable. More information on the specifications of the mobile platform is available on the Gènèration robots website [5].

4.3.1 Path Following Control

In path following, the robot is required to converge to and follow a geometric path. In this work, the optimal trajectories generated by the Leapfrog method in the previous section are considered as the reference path. It would be possible to use Leapfrog velocities as control inputs to the robot, however it is important to provide feedback control for realistic path following in situations where the robot experiences slip or the robot control is not accurate enough. Therefore, a proportional plus derivative (PD) controller was used on the robot.

The inputs to the PD controller are chosen to be Leapfrog-computed position information say, $P_z(t) = [x_z, y_z, \varphi_z]^T$ at time t, and the sign of Leapfrog-computed linear velocities, to allow the PD controller to distinguish forward and reverse directions. Figure 4.7 illustrates how the control inputs to the robot are computed at each time step.

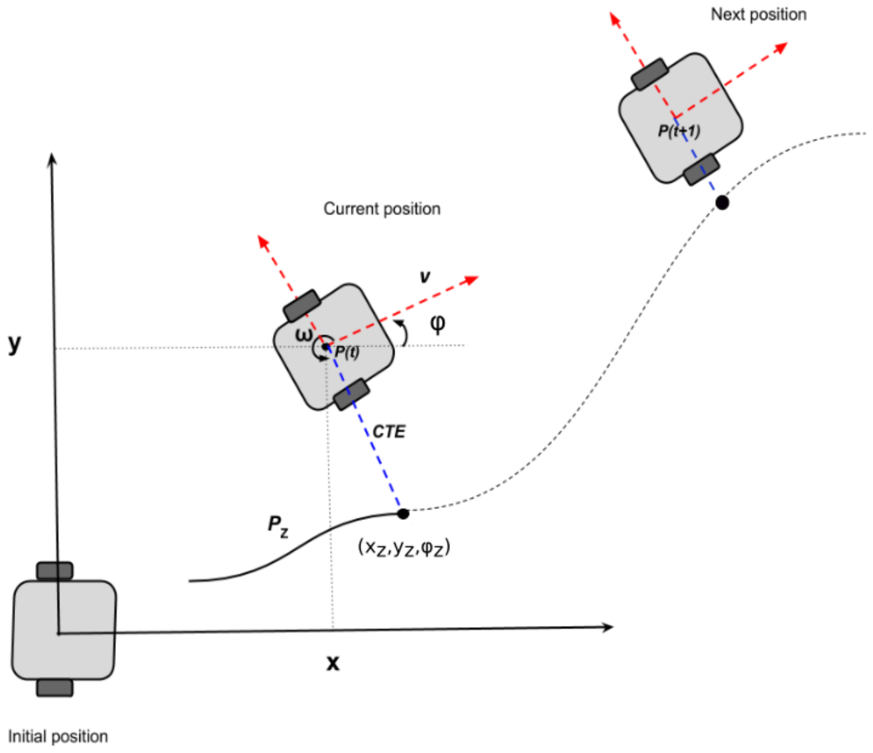


Figure 4.7: Steps on how the position error is computed for the PD controller.

The positional error which is the distance from the current robot location P to the current point on the path P_z with respect to the current robot pose is computed as follows:

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = T_t \begin{bmatrix} x_z \\ y_z \\ 1 \end{bmatrix}, \tag{4.3.1}$$

where

$$T_t = \begin{bmatrix} \cos\varphi & -\sin\varphi & x \\ \sin\varphi & \cos\varphi & y \\ 0 & 0 & 1 \end{bmatrix}^{-1}, \tag{4.3.2}$$

(x_z, y_z, φ_z) are the coordinates for the reference path $\mathbf{P}_z$, which is where the robot should be on the path and (x, y, φ) are the current coordinates for the controlled robot. The Cross Track Error (CTE) is then calculated as the y-value of the positional error, CTE $= e_2$.

Using the PD controller, the steering command is computed as:

$$a = -k_P\, e_2 - k_D(e_2 - e_2(t-1))/dt, \tag{4.3.3}$$

where $e_2(t-1)$ is the previous CTE, and k_P and k_D are the positive constant proportional and derivative gains, respectively. The next control point is then computed as

$$\mathcal{C}(t) \quad = \quad R(a) \cdot T(P(t)) \cdot P_z(t+1). \tag{4.3.4}$$

Therefore, the next or desired velocities to send to the robot are calculated as follows:

$$v(t+1) \quad = \quad \mathcal{C}(t) \cdot x/dt, \tag{4.3.5}$$

$$\omega(t+1) \quad = \quad a/dt. \quad (a \text{ is negated when } v \text{ is negative}) \tag{4.3.6}$$

To demonstrate the performance of the control strategy, Case 2 in Section 4.2 is considered.

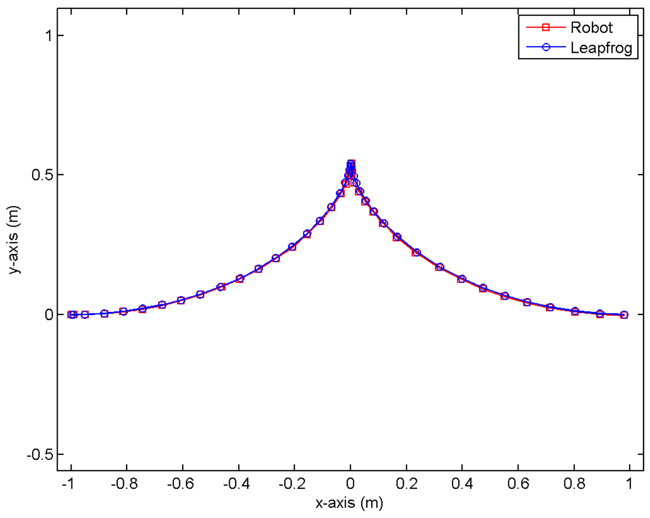


Figure 4.8: Mobile robot and Leapfrog trajectories for Case 1.

From Figure 4.8 it can be seen that the proposed control law shows good convergence. Table 4.3 shows the MSE for robot position relative to the Leapfrog method's position for Cases 1-4 outlined in Section 4.2.

From the MSE values it can be noted that there are only small errors between the trajectories from the Leapfrog algorithm and those from the robot.

Table 4.3: MSE values for real experiments.

Case	MSE (m^2)
1	1.8538×10^{-5}
2	1.5585×10^{-5}
3	2.0727×10^{-5}
4	9.5429×10^{-6}

4.4 Conclusion

In this chapter, an application of the Leapfrog method to find optimal paths for a two-wheeled mobile robot was introduced. First, using the mobile robot kinematic model and a cost functional, an optimal control problem was formulated. Then, the minimum principle of Pontryagin was employed and the optimality conditions were derived. The TPBVP resulting from the optimality conditions was then solved using the Leapfrog method. Finally, simulations were performed for the mobile robot given different positions in each example and optimal paths were generated. The simulation results show that the Leapfrog method can find solutions in cases where the BVP4C approach failed. This was shown in Case 3 where BVP4C resulted in the generation of a singular Jacobian while attempting to solve the collocation equations. This was traced to a problem with the initial guess of the solution. Path following experiments using a real robot were also performed, where it was shown that the robot was able to follow the paths generated by the Leapfrog method.

Chapter 5

Leapfrog Paths for a Two-wheeled Mobile Robot in the Presence of Obstacles

In this chapter, the optimal control problem is employed to generate collision-free paths for the two-wheeled mobile robot in an environment with obstacles. The problem of planning an optimal path through the optimal control approach in an environment with obstacles can be challenging to solve, since conventional optimal control approaches cannot directly be used to deal with the obstacle avoidance problem. A simplified approach in handling obstacle avoidance is adding potential fields to the cost functional to be minimized; in that case, the problem is defined as an optimal control problem without constraints [9; 23; 111]. This formulation ensures that the robot avoids obstacles and can find optimal paths at the same time. Therefore, in this work obstacle avoidance is mathematically modelled using a Gaussian potential field and added to the cost functional.

Similarly to Chapter 4, the optimal control problem is formulated using the necessary conditions for optimality derived through PMP, and the resulting TPBVP is numerically solved using the Leapfrog method. In order to execute the Leapfrog method an initial feasible path is prescribed. One of the simplest initialization approaches is to assume that the path is a straight line between the initial and final states, as in Chapter 4. With obstacles present, however, a feasible (but not necessarily optimal) path is required. In this chapter, the trajectories produced by the A* and RRT algorithms are used as initial feasible paths for Leapfrog. Hence the second part of this chapter is an evaluation to check if the Leapfrog algorithm is capable of finding a unique path given different initialization approaches. It will also be checked whether Leapfrog is unaffected by the specific route taken by the initial path, or if the Leapfrog result does depend on the method used to generate the initial path.

To evaluate the efficiency of the Leapfrog method, numerical simulations are done, and a comparison is made with the BVP4C algorithm. The simulation results show that a similar path cost can be obtained with the Leapfrog approach which forms part of the first part of the chapter.

5.1 Obstacle Avoidance Formulation

Obstacle avoidance is one of the important aspects of solving motion planning problems. It addresses the problem of how to drive a robot from one state to another while avoiding collisions with obstacles. There are different methods that can be used for obstacle avoidance and each has its own advantages and disadvantages, as mentioned in Chapter 2. Apart from alleviating the known issues of getting trapped in local minima, GNRON and an inability to deal with closely spaced obstacles, the potential field methods are known to provide simple and elegant solutions for obstacle avoidance for mobile robot planning [9]. For this reason the potential field method is used as an obstacle avoidance technique in this chapter.

To represent the obstacles, a repulsive field which imposes an area of high potential near obstacles is defined by a Gaussian function. Obstacles represented by the Gaussian potential field provide a region with finite height surrounding the obstacle. Parameters defining this region can be chosen to prevent collision between the robot path and the obstacle. In addition, the Gaussian function derivatives are continuous and smooth functions [97] which makes them suitable as a potential obstacle representation function.

From the literature presented in Chapter 2 it can be noted that Gaussian potential fields are mostly used to avoid the GNRON problem [96] and to create a passage between closely spaced obstacles which in turn reduces the number of local minima [95]. In this work, however, challenges of local minima are not considered. In the literature it is presented that one can implement global solutions such as navigation functions [47], harmonic functions [48] and Simulated Annealing [71] in order to avoid local minima.

Assume that the robot is a point mass in a 2-D workspace with the robot position $\mathbf{q} = [x_r,\, y_r]^T$. Each circular obstacle in the workspace is located at center point (x^i_{obs}, y^i_{obs}) with radius r^i_{obs}. The repulsive potential function is therefore given by

$$F_{rep_i}(\mathbf{q}) = A_{rep} \exp\left[-\frac{1}{2}\left(\frac{(\rho^i)^2}{\sigma^2_{rep_i}}\right)^C \right], \tag{5.1.1}$$

where A_{rep} is a positive constant, parameter C relates to the steepness of the obstacle and σ_{rep_i} is the size of the obstacle which is related to r^i_{obs}. The distance ρ^i between the center of the mobile base (x_r, y_r) and the center of the

i-th obstacle is given by

$$\rho^i = \sqrt{(x_r - x^i_{obs})^2 + (y_r - y^i_{obs})^2}. \tag{5.1.2}$$

By computing the repulsive force with respect to a sum over all obstacles in the workspace, Eq. (5.1.1) becomes

$$F_{rep}(\mathbf{q}) = \sum_{i=1}^{n} F_{rep_i}(\mathbf{q}), \tag{5.1.3}$$

where n is the number of obstacles.

The Gaussian potential function defined in Eq. (5.1.3) is used to avoid collisions between the robot and the obstacles. With the Gaussian function, the behaviour of the robot around obstacles can be modified by tuning the parameters σ_{rep_i}, A_{rep} and C. By tuning these parameters properly, it is possible to guarantee collision-free paths in the presence of obstacles [112].

5.2 Optimal Control Formulation in the Presence of Obstacles

In this section, the optimal control problem for the mobile robot in the presence of obstacles is formulated through the minimum principle where the optimality conditions will be derived. The resulting TPBVP is then numerically solved using the Leapfrog method, yielding a critical path for the mobile robot.

The cost function is defined to minimize the energy control inputs and includes the Gaussian potential function. The cost function of the optimal control problem is therefore given by

$$J = \frac{1}{2}\int_{t_0}^{t_f} [\mathbf{u}^T R \mathbf{u} + \sum_{i=1}^{n} F_{rep_i}(\mathbf{q})]\, dt, \tag{5.2.1}$$

where $\mathbf{u} = [v, \omega]^T$ is the control input vector and R is the control weighting matrix.

Given the robot kinematic model in Eq. (1.2.1) and adjoining the constraints to the cost functional in Eq. (5.2.1) with Lagrange multiplier, $\boldsymbol{\lambda}$, the Hamiltonian function is given by

$$H = \frac{1}{2}\left(R_{11}v^2 + R_{22}w^2 + \sum_{i=1}^{n} F_{rep_i}(\mathbf{q})\right) + \lambda_1 v\cos\varphi + \lambda_2 v \sin\varphi + \lambda_3\omega. \tag{5.2.2}$$

Using the control optimality condition $\dfrac{\partial H}{\partial \mathbf{u}} = 0$, the optimal velocities are

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} -(\lambda_1\cos\varphi + \lambda_2\sin\varphi) \\ -\lambda_3 \end{bmatrix}. \tag{5.2.3}$$

The optimality condition for the costate equations is given by

$$-\frac{\partial H}{\partial \mathbf{q}} = \begin{bmatrix} -\sum_{i=1}^{n} A_{rep} \frac{(x - x^i_{obs})}{2\sigma^2_{rep_i}} C \exp\left(-\frac{(\rho^i)^2}{2\,\sigma^2_{rep_i}}\right)\left(\frac{(\rho^i)^2}{\sigma^2_{rep_i}}\right)^{C-1} \\ -\sum_{i=1}^{n} A_{rep} \frac{(y - y^i_{obs})}{2\sigma^2_{rep_i}} C \exp\left(-\frac{(\rho^i)^2}{2\,\sigma^2_{rep_i}}\right)\left(\frac{(\rho^i)^2}{\sigma^2_{rep_i}}\right)^{C-1} \\ \frac{1}{2}(\lambda_2^2 - \lambda_1^2)\sin 2\varphi + \frac{1}{2}\lambda_1\lambda_2\cos 2\varphi \end{bmatrix}. \tag{5.2.4}$$

The control input $\mathbf{u}$ can be solved in terms of the state $\mathbf{q}$ and costate $\boldsymbol{\lambda}$. Substituting the optimal velocities from Eq. (5.2.3) in the state optimality condition yields

$$\dot{\mathbf{q}} = \frac{\partial H}{\partial \boldsymbol{\lambda}} = \begin{bmatrix} -\frac{1}{2}\lambda_1(1+\cos 2\varphi) - \frac{1}{2}\lambda_2 \sin 2\varphi \\ -\frac{1}{2}\lambda_1 \sin 2\varphi - \frac{1}{2}\lambda_2(1-\cos 2\varphi) \\ -\lambda_3 \end{bmatrix}. \tag{5.2.5}$$

Hence the TPBVP considered becomes:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{\lambda}_1 \\ \dot{\lambda}_2 \\ \dot{\lambda}_3 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2}\lambda_1(1+\cos 2\varphi) - \frac{1}{2}\lambda_2 \sin 2\varphi \\ -\frac{1}{2}\lambda_1 \sin 2\varphi - \frac{1}{2}\lambda_2(1-\cos 2\varphi) \\ -\lambda_3 \\ -\sum_{i=1}^{n} A_{rep} \frac{(x - x^i_{obs})}{2\sigma^2_{rep_i}} C \exp\left(-\frac{(\rho^i)^2}{2\,\sigma^2_{rep_i}}\right)\left(\frac{(\rho^i)^2}{\sigma^2_{rep_i}}\right)^{C-1} \\ -\sum_{i=1}^{n} A_{rep} \frac{(y - y^i_{obs})}{2\sigma^2_{rep_i}} C \exp\left(-\frac{(\rho^i)^2}{2\,\sigma^2_{rep_i}}\right)\left(\frac{(\rho^i)^2}{\sigma^2_{rep_i}}\right)^{C-1} \\ \frac{1}{2}(\lambda_2^2 - \lambda_1^2)\sin 2\varphi + \frac{1}{2}\lambda_1\lambda_2\cos 2\varphi \end{bmatrix},$$

with boundary conditions $\mathbf{q}(t_0) = \mathbf{q}_0$ and $\mathbf{q}(t_f) = \mathbf{q}_f$.

5.3 Experiments in the Presence of Obstacles

In this section, the optimal control problem for the mobile robot in the presence of obstacles is numerically solved using the Leapfrog method. For the simulations, the robot is given an initial position and orientation, and a goal position and orientation. The robot will plan its path in a workspace with circular obstacles each located at center point (x^i_{obs}, y^i_{obs}) with radius r^i_{obs} for $i = 1, \ldots, n$. The positions of the obstacles in the workspace are known to the robot. The control weighting matrix in the cost function in Eq. (5.2.1) is given by $R = diag(1, 1)$, the height of each obstacle is $A_{rep} = 1$, the steepness of each obstacle is $C = 1$ and the width of each obstacle is represented by $\sigma_{rep_i} = r^i_{obs}$. Since the values for the Gaussian parameters need to be chosen before Leapfrog is executed, several simulations were done and the value of 1 was selected for these parameters.

To evaluate the Leapfrog method, first a path generated by the RRT algorithm is chosen as an initial feasible partition represented by $\mu^{(0)}_{\mathbf{z}}$. To achieve a faster convergence in the Leapfrog method, a few sets of way-points are chosen as partition points ($p = 8$) from the RRT path. Once the initial feasible path is chosen, the Leapfrog method uses an affine approximation to initialize values for the costates. However, for the mobile robot model presented in this section, the approximation did not yield reasonable guesses. In [68] it is demonstrated that the affine approximation can provide good guesses for some of the costates. Therefore in this chapter, a gradient approach is used to calculate initial conditions for the costates as follows:

$$\lambda_{1,k-1} = \frac{x_{k+1} - x_{k-1}}{t_{k+1} - t_{k-1}}, \tag{5.3.1}$$

$$\lambda_{2,k-1} = \frac{y_{k+1} - y_{k-1}}{t_{k+1} - t_{k-1}}, \tag{5.3.2}$$

$$\lambda_{3,k-1} = \frac{\varphi_{k+1} - \varphi_{k-1}}{t_{k+1} - t_{k-1}}. \tag{5.3.3}$$

The numerical simulations are carried out using MATLAB. For comparison, the BVP4C method which has shown to be effective in the previous chapter is used. Both the Leapfrog and BVP4C methods use the same Gaussian parameters and final time (t_f) per problem.

Case 1: In this simulation, the mobile robot is to move from a given initial state $[0, 0, \pi/4]$ to a final state $[1, 1, \pi/4]$ at total time $t_f = 4s$. The obstacles are located at center points $x^1_{obs} = 0.35$, $y^1_{obs} = 0.45$ and $x^2_{obs} = 0.55$, $y^2_{obs} = 0.7$, respectively. The radii of the obstacles are $r^1_{obs} = 0.1$ and $r^2_{obs} = 0.1$. As it can be seen in Figure 5.1, the Leapfrog method is able to plan a critical collision-free path. It can also be noted that the state trajectory generated by the BVP4C solver is very similar to the Leapfrog trajectory.

Case 2: For this simulation, the robot is considered to move from an initial state $[0, 0, 0]$ to a final state $[2, 0, 0]$ at total time $t_f = 5s$. The center coordinates of the obstacles are $x^1_{obs} = 0.9$, $y^1_{obs} = 0$, $x^2_{obs} = 1.1$, $y^2_{obs} = 0$, $x^3_{obs} = 0.3$, $y^3_{obs} = -0.35$, $x^4_{obs} = 1.6$, $y^4_{obs} = 0.35$ and $x^5_{obs} = 0.4$, $y^5_{obs} = 0.3$ with the radii $r^i_{obs} = 0.08$, for $i = 1, \ldots, 5$. Figure 5.2 shows the paths generated by the Leapfrog method and BVP4C.

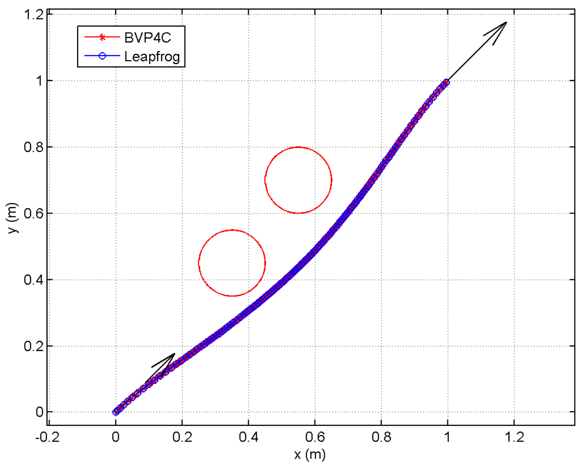


Figure 5.1: The BVP4C and Leapfrog trajectories in the presence of two obstacles for Case 1.

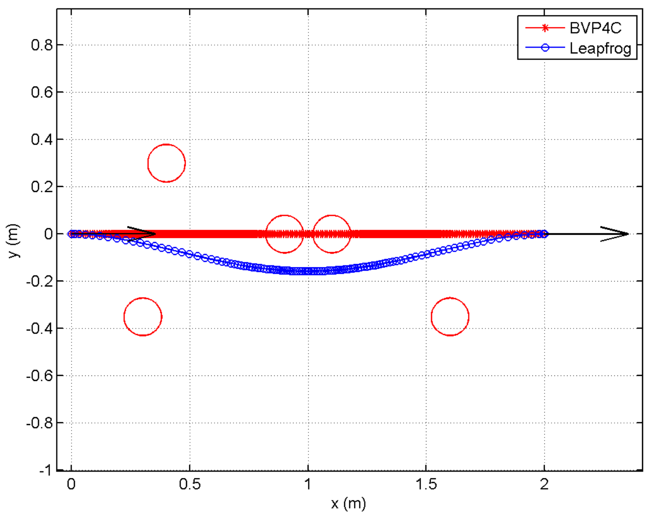


Figure 5.2: The trajectories obtained from BVP4C and Leapfrog in the presence of five obstacles for Case 2.

It is also depicted in Figure 5.2 where there are obstacles between the start state and final state that the Leapfrog algorithm managed to find a collision-free path whereas the BVP4C path went over the obstacles resulting in a higher path cost (see Table 5.1). Given a better initial guess, however, the BVP4C can produce a collision-free path. Hence Case 2 was executed again with a different initial guess for the costates, $\lambda(t_0) = [1\,1\,1]^T$, and the results for BVP4C are shown in Figure 5.3 where the cost is now matching the Leapfrog result.

Path costs for the Leapfrog and BVP4C solutions are computed where the

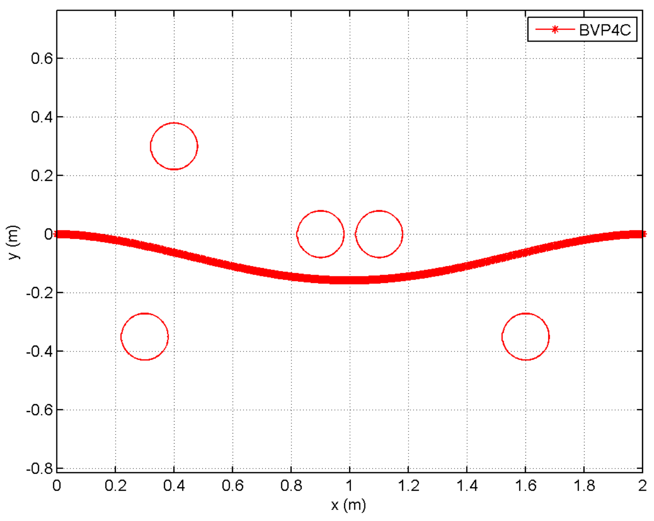


Figure 5.3: The trajectory obtained from BVP4C using different initial costate values for Case 2.

forward and angular velocities together with the Gaussian function for each time point are used to calculate the integrand of Eq. (5.2.1). The resulting discrete-time function is then integrated numerically using `trapz` in MATLAB. Table 5.1 shows the path costs for the cases above together with the execution times for both the Leapfrog method and BVP4C.

Table 5.1: Numerical evaluation

Method	Case	Cost	CPU time(s)
Leapfrog	1	0.502	81.22
	2	0.604	135.55
BVP4C	1	0.502	5.29
	2 (Fig. 5.2)	0.702	9.35
	2 (Fig. 5.3)	0.604	8.47

5.3.1 Path Following Results

Path following experiments are conducted using a Pioneer 3-DX mobile robot. A simple proportional plus derivative (PD) controller is implemented where the paths generated by Leapfrog are taken as inputs. The PD controller then sends approximate velocities to the robot so that it follows the Leapfrog-computed position (x, y), and the sign of the Leapfrog-computed forward velocity. The trajectories obtained from the Pioneer robot and Leapfrog simulation in the presence of two obstacles (Case 2) are shown in Figure 5.4. From this simulation, it can be seen that the robot managed to follow the path generated by the Leapfrog method.

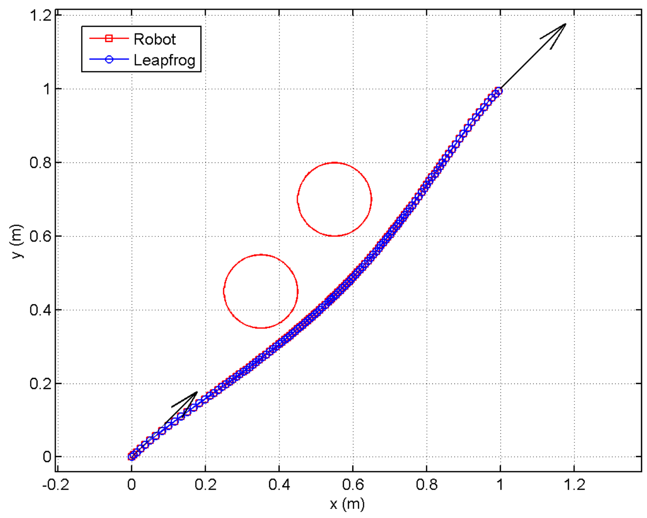


Figure 5.4: The trajectories obtained from the Pioneer robot and Leapfrog simulation in the presence of obstacles.

Table 5.2 shows the MSE for robot position relative to the Leapfrog position for all the cases. In this table it can be noted that the results obtained from the robot experiments are close to those obtained from the Leapfrog simulations.

Table 5.2: MSE values for real experiments.

Case	MSE (m^2)
1	2.1076×10^{-6}
2	$1.03979\ \times10^{-6}$
3	$3.2580\ \times10^{-6}$
4	$1.9094\ \times10^{-6}$

Concluding the numerical results for this section, it is noted from Table 5.1 that the cost values for the Leapfrog method are very close to the BVP4C cost values. This shows that the Leapfrog method has the capability to perform well even in the presence of obstacles. In [69] it was shown that the Leapfrog method can produce solutions in cases where BVP4C encountered singularities.

5.4 Leapfrog Initial Feasible Path

In Chapter 4, the first partition for the Leapfrog method was selected as a straight line between the initial and final states. However, due to obstacles present in the robot workspace, a different approach is required to generate a collision-free initial partition for the Leapfrog method. The main focus of this section is to demonstrate that the Leapfrog algorithm is capable of finding a unique critical path using initialization from different path planners. This is illustrated by using paths generated by the A* and RRT algorithms. It should be noted that the path produced by these planners does not necessarily need to be optimal, only feasible. In addition, paths generated by these planners are not refined by Leapfrog, they provide the initialization trajectories for the Leapfrog method. The optimal control problem for mobile robot motion planning in the presence of obstacles is defined as in the previous section.

5.4.1 Simulation Results

To demonstrate the performance of the Leapfrog method, four different sets of examples are considered. Both A* and RRT build a path in (x, y) coordinates. To get the orientation φ, the MATLAB function `atan2` is used so that the resulting orientation is aligned to the path. Since the path generated by A* is in discrete space, it is first converted to continuous space before its use in forming the initial partition for Leapfrog. Furthermore, the A* and RRT methods may produce many way-points which form the initial path. These points are reduced using B-spline interpolation for faster Leapfrog convergence. The Leapfrog method is executed for each initialization and the results are illustrated in the following cases. For each case, the path produced by Leapfrog together with the paths generated by A* and RRT, are plotted.

Case 3: In the first simulation of this section, the mobile robot is required to move from initial state $[0, 0, \frac{\pi}{4}]$ to final state $[1, 1, \frac{\pi}{4}]$ during the overall time $t_f = 4s$ while avoiding two obstacles. The obstacle locations are given in Table 5.3 and the results are shown in Figure 5.5.

Table 5.3: Obstacle coordinates for Case 3.

Obstacle, i	(x^i_{obs}, y^i_{obs})	r^i_{obs}
1	(0.35, 0.45)	0.1
2	(0.55, 0.7)	0.1

Case 4: In this simulation, three obstacles are considered where their location coordinates can be seen in Table 5.4. The mobile robot is expected to find

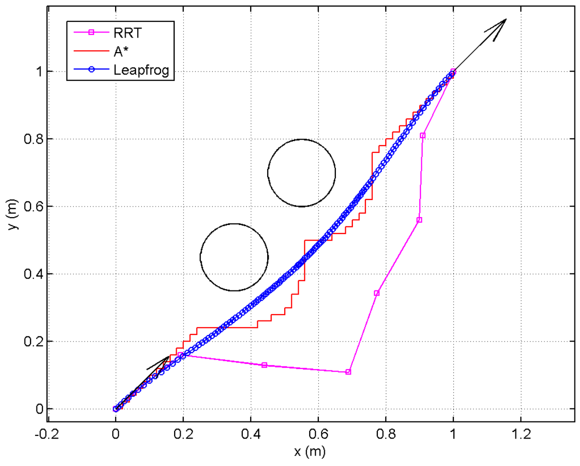


Figure 5.5: Paths generated by A* and RRT and the path resulting from the Leapfrog method for Case 3.

a path from initial state $[0, 0, \frac{\pi}{4}]$ to final state $[4, 4, \frac{\pi}{4}]$ within the final time $t_f = 5s$. The results are shown in Figure 5.6.

Table 5.4: Obstacles coordinates for Case 4.

Obstacle, i	(x^i_{obs}, y^i_{obs})	r^i_{obs}
1	(1, 2)	0.3
2	(2.5, 1)	0.3
3	(2.5, 2.6)	0.3

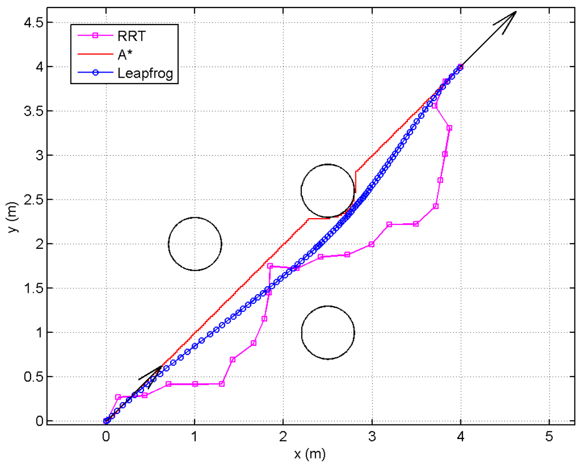


Figure 5.6: Paths generated by A* and RRT and the resulting path resulting from the Leapfrog method for Case 4.

For the simulations shown in Figure 5.5 and Figure 5.6 it can be noted that if there were no obstacles, the resulting optimal path would be a straight line.

Case 5: To intensify the problem in this simulation, the mobile robot is expected to move from initial state $[0, 0, 0]$ to final state $[2, 0, \pi]$ while avoiding seven obstacles (as defined in Table 5.5) within the final time $t_f = 4s$. For this example, the robot is expected to complete a direction reversal.

Table 5.5: Obstacles coordinates for Case 5.

Obstacle, i	(x^i_{obs}, y^i_{obs})	r^i_{obs}
1	(0.9, 0)	0.08
2	(1.1, 0)	0.08
3	(0.3, -0.35)	0.08
4	(1.6, -0.35)	0.08
5	(0.3, 0.2)	0.08
6	(1.6, 0.2)	0.08
7	(1, 0.6)	0.08

As can be seen in Figure 5.7 the Leapfrog method produced a collision-free path that satisfies the initial and final configuration.

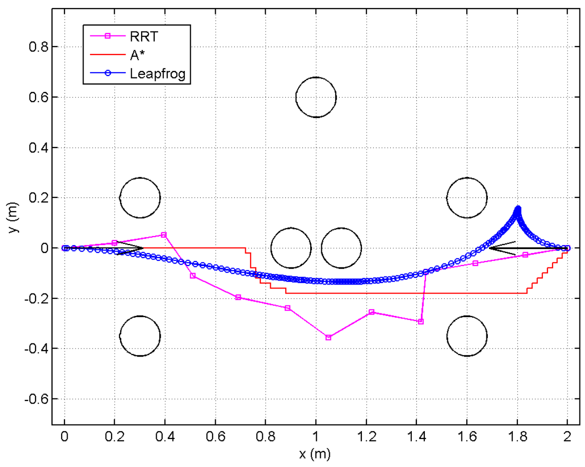


Figure 5.7: Paths generated by A* and RRT and the resulting path resulting from the Leapfrog method for Case 5.

It is known through literature that RRT produces different paths every time the algorithm is executed. Due to this, choosing which path to take for the Leapfrog initialization can be a challenge. Hence in the following example, two paths generated by the RRT are used to further test if the Leapfrog method will find a unique solution.

Case 6: Given the robot initial state $[-0.5, -3, 0]$ and final state $[1.5, 2.5, 0]$ a critical path in the presence of obstacles is found (see Figure 5.8).

Table 5.6: Obstacles coordinates for Case 6.

Obstacle, i	(x^i_{obs}, y^i_{obs})	r^i_{obs}
1	(1.5, 1)	0.5
2	(-0.5, -1)	0.8

For this case, the initial paths considered are generated by RRT, i.e., RRT_1 and RRT_2 as shown in Figure 5.8. As seen in Figure 5.8, Leapfrog managed to find a solution given both paths for initialization. This indicates that the Leapfrog method is not directly tied to the initial path.

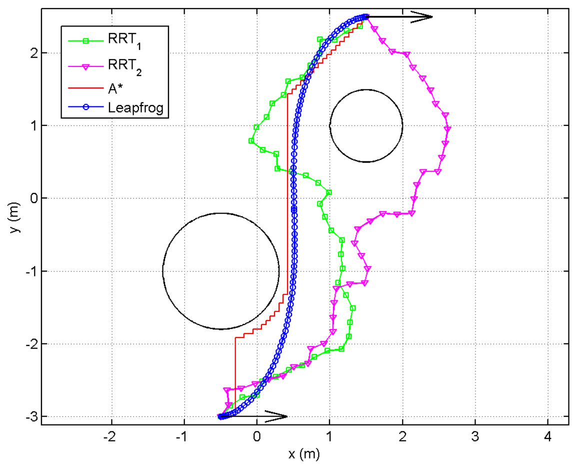
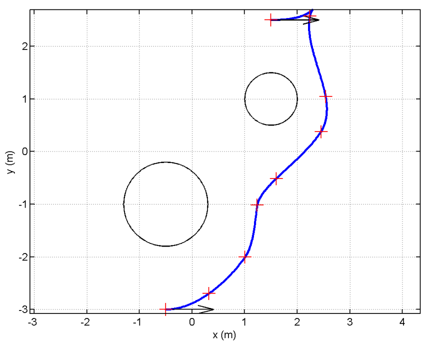


Figure 5.8: Two paths generated by RRT, a path from A* and the path resulting from the Leapfrog method for Case 6.

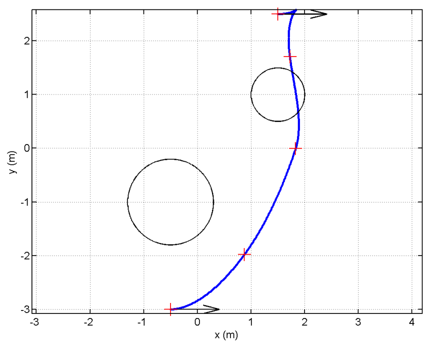
5.5 Discussion

This section discusses the effect that the Gaussian parameters have on generating collision-free paths. Limitations in the Leapfrog numerical implementation for mobile robot planning in an environment with obstacles are also discussed.

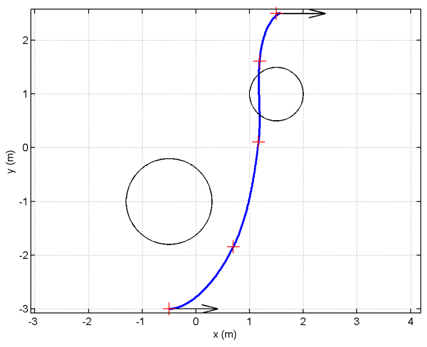
As mentioned before, with Gaussian functions one can model obstacles with approximate details of their size and location. This can be achieved by adjusting parameters such as A_{rep}, C and σ_{rep_i}. Tuning these parameters can also ensure that collision-free paths are obtained, though there is no systematic way to do so [97]. It was observed in simulations with the example of Case 6, initialized using the RRT$_2$ path, that during the Leapfrog iterations the collision avoidance is not entirely successful, as shown in Figure 5.9. Between iterations $k = 7$ and $k = 11$ the path goes over the obstacle but with a reduced cost. The control term in the cost functional can possibly overpower (be higher than) the Gaussian repulsive parameter, thereby allowing a collision even as the path cost reduces.



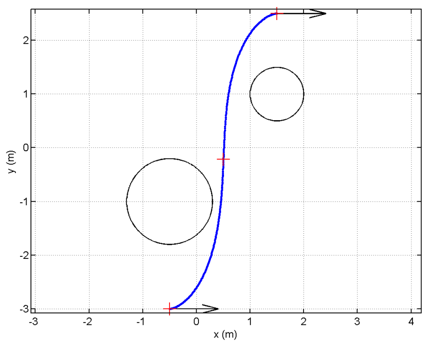
(a) $\mu_z^{(1)}$ with $p = 8$ and cost $= 11.08$.



(b) $\mu_z^{(7)}$ with $p = 4$ and cost $= 7.38$.



(c) $\mu_z^{(11)}$ with $p = 4$ and cost $= 6.30$.

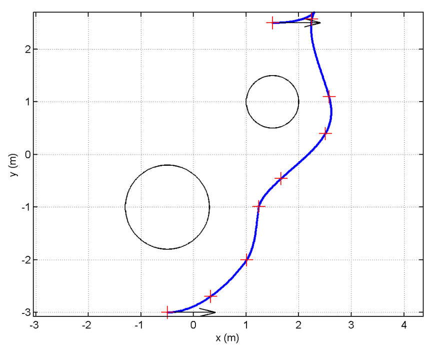


(d) $\mu_z^{(13)}$ with $p = 2$ and cost $= 5.81$.

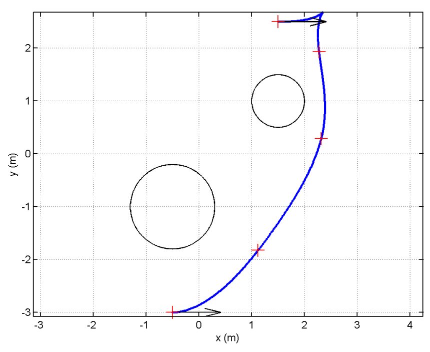
Figure 5.9: Local paths generated by Leapfrog at iterations $k = 1, 7, 11$ and 13 for Case 6 with $A_{rep} = 1$.

For this simulation, the Gaussian parameters were chosen as $A_{rep} = 1$, which means that the cost of crossing the obstacle is relatively low, when compared to the overall path cost. After increasing the value of the obstacle height to be $A_{rep} = 10$, locally feasible paths were achieved in the Leapfrog iterations. Figure 5.10 shows the resulting path generated by the Leapfrog when A_{rep} is increased. Even though the results shown in Figure 5.10 are collision-free, the cost is higher than those shown in Figure 5.9.

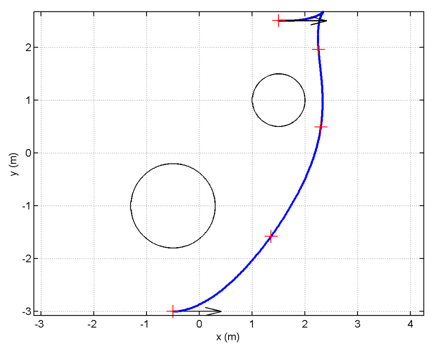
Based on the simulations shown in Figures 5.9 and 5.10, the Leapfrog method can yield collision-free paths provided that the potential field parameters are defined well.



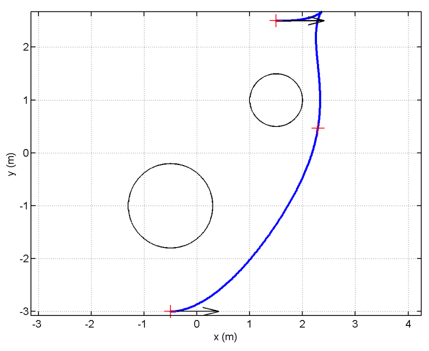
(a) $\mu_z^{(1)}$ with $p = 8$ and cost $= 12.06$.



(b) $\mu_z^{(7)}$ with $p = 4$ and cost $=9.18$.



(c) $\mu_z^{(11)}$ with $p = 4$ and cost $= 9.09$.



(d) $\mu_z^{(13)}$ with $p = 2$ and cost $= 9.09$.

Figure 5.10: Local trajectories generated at Leapfrog iterations $k = 1, 7, 11$ and 13 for Case 6 where $A_{rep} = 10$.

From the Leapfrog method's point of view, it is observed that even though critically feasible paths can be found using the method, there are some shortcomings in the numerical implementations. The first one is related to the number of partition points in the feasible path in the first iteration. The choice of partition points affects the algorithm's convergence time. The fewer the partition points the quicker the rate of convergence, especially in the first iteration. Hence a relatively low number of RRT way-points is used in the simulations.

The second one is related to the affine approximation for initial costate values. Due to the nature of the mobile robot's kinematic model, the affine approximation for costate initialization did not work. Hence in this chapter, a different approach to find initial costates was used.

It is also noted that Leapfrog does not necessarily converge to a unique critical path, as different solutions can be obtained from the same initial feasible path, as shown in Figures 5.9 and 5.10. Therefore, Leapfrog's solution must be bound to the route of the initial feasible path, to ensure that the solutions in the intermediate iterations are collision free.

5.6 Conclusion

In this chapter, path planning for a mobile robot in the presence of obstacles was presented as an optimal control problem, and a TPBVP was derived. The Leapfrog method was implemented to determine paths using the necessary conditions of Pontryagin's Minimum Principle. The proposed algorithm allows the robot to plan a collision-free path through static obstacles by minimizing a cost functional that includes an energy term and the Gaussian potential function.

Simulations were conducted in Section 5.3 given different case studies, and comparisons were made with the BVP4C algorithm showing that similar path costs can be obtained with the Leapfrog approach. In Section 5.4.1 the formation of the Leapfrog initialization using different path planners, e.g., A^* and RRT, was investigated. It was observed in Case 6 that the Leapfrog method did not find a unique path. Therefore, it cannot be concluded that Leapfrog's best solution is not bound to the route of the initial feasible path. The effects of the potential field parameters in terms of guaranteeing collision-free paths were also discussed and demonstrated in the simulations.

From the case studies and numerical results in both Section 5.3 and Section 5.4.1, it was shown that the Leapfrog algorithm performed well in environments with obstacles for the examples presented in this chapter. Also, the path following experiments confirmed that a physical robot is able to follow the Leapfrog paths.

Chapter 6

Leapfrog Paths for a Mobile Manipulator in the Presence of Obstacles

In recent years, research has been conducted on motion planning and obstacle avoidance for wheeled mobile manipulators. Mobile manipulators are robotic systems that consist of an arm manipulator mounted on a mobile robot platform. These systems have been widely used due to their larger workspace and ability to reach targets that are initially outside the manipulator's reach. The motion planning problem for the mobile manipulator involves finding a path for the robot from an initial position to the desired goal. The mobile manipulator has two subsystems connected where the mobile platform is often associated with nonholonomic constraints, and the manipulator may be holonomic. In general, this complicates the motion planning problem, especially for mobile manipulators with many degrees of freedom.

In many applications, the mobile manipulator is required to move the end-effector along a preferred path. A trajectory planning method for a mobile manipulator was proposed in [113] while considering a predefined path for the end-effector. In [114] a planning and control algorithm for a mobile manipulator to follow the desired end-effector and mobile robot platform was presented. The control algorithm was tested on two mobile manipulator systems, where a planar arm was mounted on a differential drive and a car-like robot. A kinematic and dynamic model for a nonholonomic mobile manipulator with two links was considered in [115]. The motion planning was done through artificial potential fields where the mobile manipulator was commanded to move the end-effector while the mobile base was stationary.

In other work, considerable effort has been directed toward finding optimal paths through optimal control approaches. Due to their highly nonlinear nature, optimal control problems are solved using numerical techniques. The formulation often leads to a TPBVP which can be solved by indirect methods.

For mobile manipulators, this approach has been applied in [110; 113] where minimizing a cost for mobile manipulator motion planning is considered. In [113] sub-optimal trajectories for mobile manipulators are presented where the problem is formulated as an optimal control problem, and an iterative method based on the gradient function is used. An established TPBVP in [110] is solved with the BVP4C function in MATLAB, to find optimal trajectories for wheeled mobile manipulators in cluttered environments. Of particular interest from the work by [110] and [116] is the use of potential fields and formulating the obstacle avoidance problem.

In this chapter, the Leapfrog algorithm is applied to path planning for mobile manipulators and represents the first use of the Leapfrog algorithm for this purpose.

6.1 Optimal Control Formulation for the Mobile Manipulator in the Presence of Obstacles

In this section, the optimal control problem for a mobile manipulator is employed in the presence of obstacles. The performance index considered for this problem includes minimum energy control effort and a Gaussian potential function as in the previous chapter. Using PMP, necessary conditions of optimality are obtained. The resulting TPBVP is solved numerically using the Leapfrog method. The main focus of this section is to demonstrate that the Leapfrog algorithm can produce optimal motion trajectories for the mobile manipulator while avoiding obstacles.

6.1.1 Obstacle Avoidance

In order to guarantee a collision-free path for the mobile manipulator given an initial and final configuration, first, a repulsive potential function based on a Gaussian function as defined in Chapter 5 is used. To further restrict the mobile manipulator platform from colliding with the obstacles, inequality constraints adopted from [116] and [117] are added to the obstacle avoidance parameters. The obstacle avoidance problem is formulated in such a way that the mobile platform, the links and joints of the mobile manipulator (see Figure 6.1) avoid collision with the obstacles.

This is achieved by using the repulsive potential field based on a Gaussian function given by

$$F_{rep_i}(\mathbf{q}) = A_{rep} \exp\left\{ -\frac{1}{2}\left(\frac{(\rho^i_{j,k})^2}{\sigma^2_{rep_i}}\right)^C \right\}, \tag{6.1.1}$$

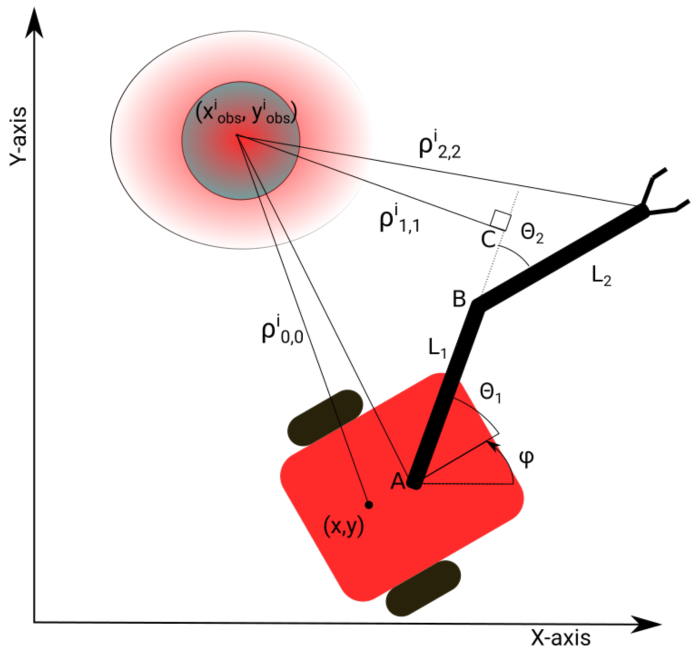


Figure 6.1: Two-link wheeled mobile manipulator.

where A_{rep} is a positive constant, ρ^i is the Euclidean distance between the robot and the i-th obstacle, C determines the height of the obstacle and σ_{rep} is the size of the obstacle. The obstacle avoidance is also applied with the inequality constraints from [116] which further restrict the mobile manipulator from colliding with obstacles.

Let the distance between the mobile manipulator components and the i-th obstacle centered at (x^i_{obs}, y^i_{obs}) be denoted by $\rho^i_{j,k}$. Taking j and k equal to zero, the distance between the center of the mobile platform (x_r, y_r) and the center of the i-th obstacle is given by

$$\rho^i_{0,0} = \sqrt{(x_r - x^i_{obs})^2 + (y_r - y^i_{obs})^2}. \tag{6.1.2}$$

Hence the constraint condition imposed on the mobile base yields

$$\Phi^i_{1,1} = \begin{cases} \sum_{i=1}^{n} A_{rep} \exp\left[-\frac{1}{2}\left(\frac{(\rho^i_{0,0})^2}{\sigma^2_{rep}}\right)^C\right] & \text{if } \rho^i_{0,0} < r^i_{obs} + r_m, \\ 0 & \text{if } \rho^i_{0,0} \geq r^i_{obs} + r_m, \end{cases} \tag{6.1.3}$$

where n represents the number of obstacles, and r_m and r_{obs} represent the radii of the mobile base and the i-the obstacle, respectively.

Assuming that the links of the manipulator are line segments, the distance of the arm manipulator components from the i-th obstacle is denoted as $\rho^i_{j,k}$ where $j = 1$ is the distance between the links ($k = 1$ for first and $k = 2$ for the second link) and the obstacle, and $j = 2$ is the distance from the joints of the first and second link.

Taking $j = 1$, the perpendicular distance between the links and the i-th obstacle is calculated as in [111] as

$$\rho^i_{1,1} = \frac{\left|\det \begin{bmatrix} L_1 \cos(\varphi + \theta_1) & x_r - x^i_{obs} \\ L_1 \sin(\varphi + \theta_1) & y_r - y^i_{obs} \end{bmatrix}\right|}{L_1},$$

$$\rho^i_{1,2} = \frac{\left|\det \begin{bmatrix} L_2 \cos(\varphi + \theta_1 + \theta_2) & x_r + L_1 \cos(\varphi + \theta_1) - x^i_{obs} \\ L_2 \sin(\varphi + \theta_1 + \theta_2) & y_r + L_1 \cos(\varphi + \theta_1) - y^i_{obs} \end{bmatrix}\right|}{L_2}.$$

The distance between the joints of the links and the i-th obstacle, i.e, $j = 2$, is given by

$$\rho^i_{2,1} = \sqrt{(x_r + L_1 \cos(\varphi + \theta_1) - x^i_{obs})^2 + (y_r + L_1 \sin(\varphi + \theta_1) - y^i_{obs})^2},$$

and

$$\rho^i_{2,2} = \sqrt{(x_e - x^i_{obs})^2 + (y_e - y^i_{obs})^2},$$

where x_e and y_e are the end effector coordinates given by

$$\begin{aligned} x_e &= x_r + L_1 \cos(\varphi + \theta_1) + L_2 \cos(\varphi + \theta_1 + \theta_2), \\ y_e &= y_r + L_1 \sin(\varphi + \theta_1) + L_2 \sin(\varphi + \theta_1 + \theta_2). \end{aligned}$$

Then the constraint condition imposed on the manipulator components is expressed as:

$$\Phi^i_{2,k} = \begin{cases} \displaystyle\sum_{i=1}^{n} A_{rep} \exp\left[-\frac{1}{2}\left(\frac{(\rho^i_{1,k})^2}{\sigma^2_{rep_i}}\right)^C\right] & \text{if } 0 \leq \gamma \leq 1 \\ & \text{and } \rho^i_{1,k} < r^i_{obs}, \\ \displaystyle\sum_{i=1}^{n} A_{rep} \exp\left[-\frac{1}{2}\left(\frac{(\rho^i_{2,k})^2}{\sigma^2_{rep_i}}\right)^C\right] & \text{if } \gamma > 1 \text{ and } \rho^i_{2,k} < r^i_{obs}, \\ 0 & \text{otherwise,} \end{cases} \tag{6.1.4}$$

where γ is the ratio of the length from the joint to the point where the perpendicular line from the obstacles intersect the link $\overline{AC}$ in Figure 6.1 over the length on the link $\overline{AB}$.

The obstacle avoidance derivations can then be added to the cost functional so that there is collision avoidance between the mobile platform, links, and joints with the obstacles.

6.1.2 Optimal Trajectory Formulation

The mobile manipulator problem is simplified by assuming a planar manipulator with two links mounted on a wheeled mobile platform. The joint angles for the planar manipulator are θ_1 and θ_2, where the lengths of the links are represented by L_1 and L_2 for the first and second link, respectively. The generalized coordinates of the mobile manipulator are expressed as $\mathbf{q} = [x\ y\ \varphi\ \theta_1\ \theta_2]^T$.

To provide an optimal solution to the wheeled mobile manipulator, an optimal control problem is defined. The aim is to find a control $\mathbf{u}$ which determines the wheeled mobile manipulator's optimal trajectory given the initial $\mathbf{q}_0$ and final $\mathbf{q}_f$ states by minimizing the cost function

$$\min_{\mathbf{u}\in\mathcal{U}} \int_{t_0}^{t_f} \frac{1}{2}\left(\mathbf{u}^T R\mathbf{u} + \Phi_{1,1}^i + \Phi_{2,k}^i\right) dt, \tag{6.1.5}$$

which penalizes the energy and Gaussian repulsive field. Equation (6.1.5) is subjected to the mobile manipulator kinematic system in Eq. (1.2.10) and the following boundary conditions are to be satisfied:

$$\mathbf{q}(t_0) = \mathbf{q}_0,\ \mathbf{q}(t_f) = \mathbf{q}_f,$$

where t_0, t_f, $\mathbf{q}_0$, and $\mathbf{q}_f$ represent the initial and final times (fixed), and the initial and final states (fixed), respectively.

Using the necessary condition for optimality based on PMP and adjoining the constraints in Eq. (1.2.10) to the cost functional in Eq. (6.1.5) with Lagrange multiplier $\boldsymbol{\lambda}$, the Hamiltonian function is given by

$$\begin{aligned} H &= \frac{1}{2}\left(u_1^2 R_{11} + u_2^2 R_{22} + u_3^2 R_{33} + u_4^2 R_{44} + \Phi_{1,1}^i + \Phi_{2,k}^i\right) \\ &+ \lambda_1(u_1 \cos\varphi - u_2\, d \sin\varphi) + \lambda_2(u_1 \sin\varphi - u_2\, d\cos\varphi) \\ &+ \lambda_3 u_2 + \lambda_4 u_3 + \lambda_5 u_4. \end{aligned} \tag{6.1.6}$$

Using the control optimality condition $\dfrac{\partial H}{\partial \mathbf{u}} = 0$, the optimal velocities are

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} -(\lambda_1 \cos\varphi + \lambda_2 \sin\varphi) \\ \lambda_1 d \sin\varphi - \lambda_2 d \cos\varphi - \lambda_3 \\ -\lambda_4 \\ -\lambda_5 \end{bmatrix}. \tag{6.1.7}$$

The state equations can be calculated from

$$\dot{\mathbf{q}} = \frac{\partial H}{\partial \boldsymbol{\lambda}}, \tag{6.1.8}$$

and costate equations from

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \mathbf{q}}. \tag{6.1.9}$$

Substituting the computed control velocities in Eq. (6.1.7) into the state and costate equations, results in a set of ordinary differential equations where the functions $\mathbf{q}(t)$ and $\boldsymbol{\lambda}(t)$ must satisfy the boundary conditions:

$$\begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} H_{\boldsymbol{\lambda}} \\ -H_q \end{bmatrix}, \qquad (6.1.10)$$

$$\text{with} \qquad \mathbf{q}(t_0) = \mathbf{q}_0, \; \mathbf{q}(t_f) = \mathbf{q}_f.$$

To solve the resulting TPBVP for the mobile manipulator, numerical techniques must be employed. When solving numerically, the goal is to iteratively solve the robot differential equations so that they conform to the set of boundary conditions. For the purpose of this chapter, the Leapfrog method and BVP4C are used and compared.

6.1.3 Numerical Results

In this section numerical solutions to the TPBVP derived from the previous section are obtained. To show the efficacy of the Leapfrog method, simulations for different scenarios for the optimal control and motion planning of the mobile manipulator in the presence of obstacles are presented. For comparison, numerical solutions generated by BVP4C are considered.

To initialize BVP4C, an initial mesh and a guess of the solution at the mesh points are required. In the simulations done here, `yinit` which is part of a structure that contains the initial guesses for the solution in BVP4C, is chosen as a default setting for initial guesses for the costates. Once the initial mesh and guesses are selected, the TPBVP with boundary conditions is solved numerically.

For the Leapfrog method implementation, first a feasible path defined as $\mu_z^{(0)}$ needs to be generated between the robot's initial and final configurations. For the simulations, the initial feasible path is generated using the RRT algorithm. Similarly to the previous chapter, the way-points from the RRT path are reduced through B-spline interpolation to $p = 16$ partition points which forms the initial feasible path.

In the simulation, the mobile manipulator is expected to plan a feasible path from a given starting configuration $\mathbf{q}_0$ to the desired goal configuration $\mathbf{q}_f$ while avoiding obstacles. The parameter values for the obstacle avoidance function are as follows: the size of each obstacle is $\sigma_{rep_i} = r^i_{obs}$, the steepness of each obstacle is $C = 1$ and the height of the repellent is $A_{rep} = 1$. For the mobile manipulator, the offset distance between the center of the mobile base wheels and the first joint is $d = 0$, and the length of the links is $L_1 = L_2 = 0.3$m.

In the figures below, the blue rectangles represent the mobile base platform, the magenta lines represent the end-effector (EE), and the red and green lines represent the first and second links, respectively.

Case 1: Suppose that the mobile manipulator moves from an initial state $\mathbf{q}_0 = [-0.5\ 0.5\ 0\ 0\ -\frac{\pi}{2}]$ to final state $\mathbf{q}_f = [2\ -0.2\ -\frac{\pi}{8}\ -\frac{\pi}{4}\ -\frac{\pi}{2}]$ at total time $t_f = 2s$. Also, an obstacle is located at a center point with coordinates $x^1_{obs} = 0.8, y^1_{obs} = 0.4$ and has radius $r^1_{obs} = 0.20$. For the simple scenario shown in Figure 6.2 and Figure 6.3, the paths generated by Leapfrog and BVP4C are given.

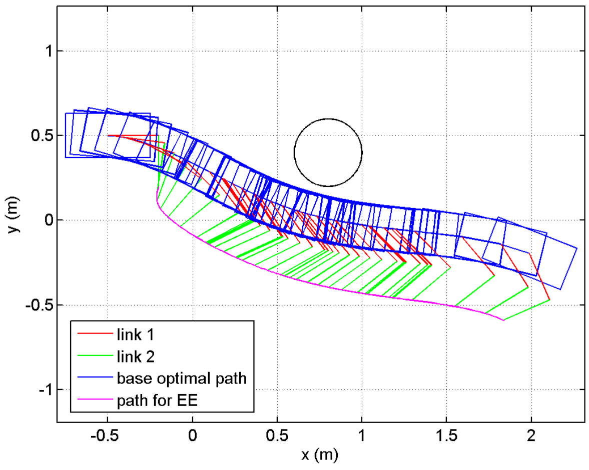


Figure 6.2: Path generated by the Leapfrog method for Case 1.

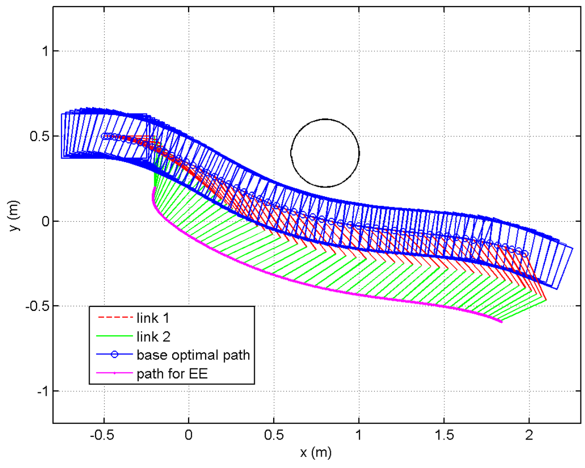


Figure 6.3: Path obtained from BVP4C for Case 1.

As it can be seen from Figures 6.2 and 6.3, the motion planning for the mobile manipulator has led to collision-free trajectories in the presence of one obstacle.

Case 2(a): For this case two scenarios are considered. Firstly, it is required that the mobile manipulator moves from the initial state at $\mathbf{q}_0 =$

$[-0.5\ -3\ 0\ \frac{\pi}{2}\ 0]$ to the final state at $\mathbf{q}_f = [1.5\ 2.5\ 0\ -\frac{\pi}{2}\ 0]$ at total time $t_f = 1.9s$. The obstacles are located at center points $x^1_{obs} = 1.5, y^1_{obs} = 1$ and $x^2_{obs} = -0.5, y^2_{obs} = 0.8$, respectively. The radii of the obstacles are $r^1_{obs} = 0.5$ and $r^2_{obs} = 0.5$. The paths of the robot are shown in Figure 6.4 and Figure 6.5, and it can be seen that the generated paths satisfy the initial and final configurations of the scenario.

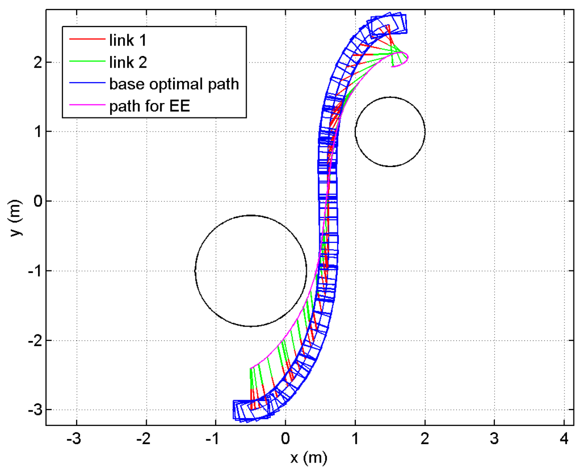


Figure 6.4: Path generated by the Leapfrog method for Case 2(a).

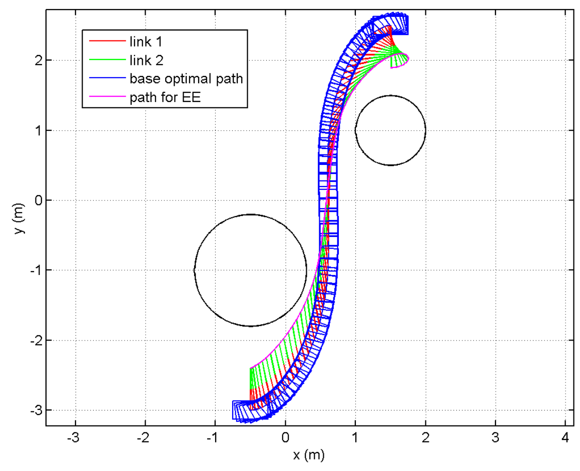


Figure 6.5: Path obtained from BVP4C for Case 2(a).

In this simulation, it is shown in Figures 6.4 and 6.5 that the mobile manipulator managed to move among the obstacles properly and the proposed methods generated collision-free paths.

Case 2(b): Secondly, the mobile manipulator is required to be moving from the initial state at $\mathbf{q}_0 = [-0.5\ -3\ 0\ -\frac{\pi}{2}\ -\frac{\pi}{2}]$ to the final state at $\mathbf{q}_f = [1.5\ 2.5\ 0\ 0\ -\frac{\pi}{2}]$ at total time $t_f = 1.9s$. Figure 6.6 and Figure 6.7 show the paths generated by the Leapfrog method and BVP4C where the configurations of the joint angles produce collision-free trajectories.

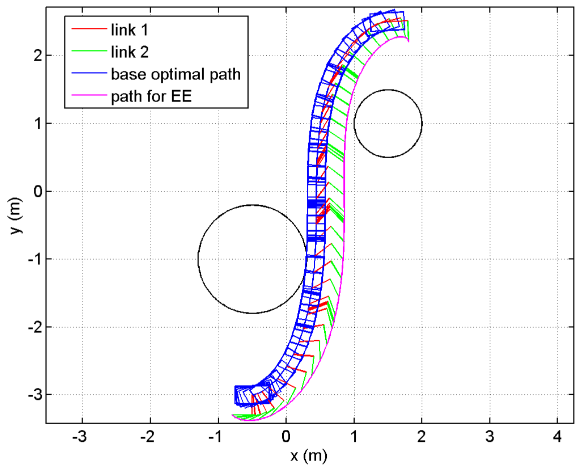


Figure 6.6: Path generated by the Leapfrog method for Case 2(b).

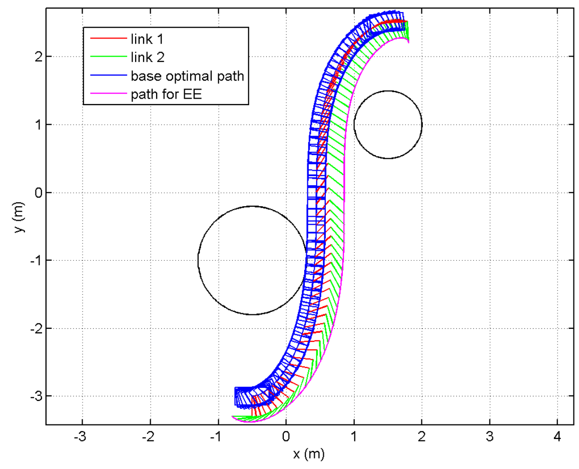


Figure 6.7: Path obtained from BVP4C for Case 2(b).

Case 3(a): Consider three obstacles located at center points $x^1_{obs} = 1, y^1_{obs} = 2$, $x^2_{obs} = 2.5, y^2_{obs} = 1$ and $x^3_{obs} = 2.5, y^3_{obs} = 2.6$ with radius $r^i_{obs} = 0.3$ for $i = 1, \ldots, 3$. The first simulation in Figure 6.8 and Figure 6.9 shows the mobile manipulator moving from an initial state $\mathbf{q}_0 = [0\ 0\ \frac{\pi}{4}\ 0\ \frac{\pi}{2}]$ to the desired goal state at $\mathbf{q}_f = [4\ 4\ \frac{\pi}{4}\ 0\ \frac{\pi}{2}]$ with total time $t_f = 1.9s$ while avoiding obstacles.

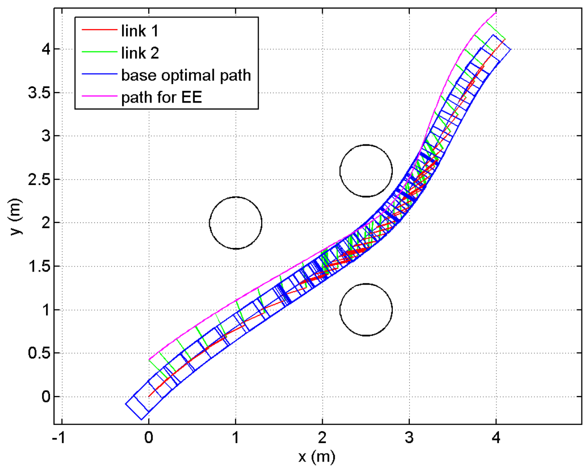


Figure 6.8: Path generated by the Leapfrog method for Case 3(a).

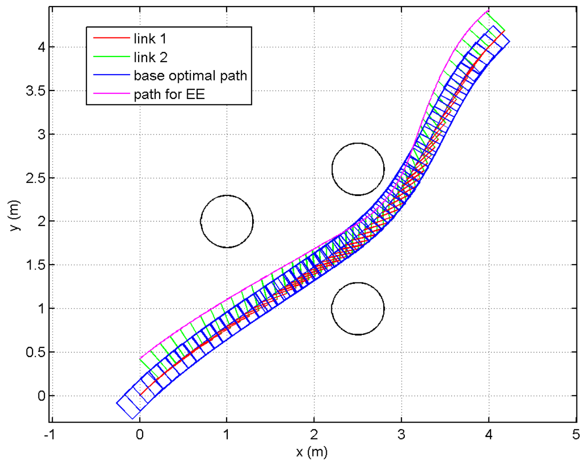


Figure 6.9: Path obtained from BVP4C for Case 3(a).

As it is seen in Figures 6.8 and 6.9, collision-free trajectories were achieved because the joints and the links of the manipulator managed to move away from the obstacles. Paths were obtained with both Leapfrog and BVP4C.

Case 3(b): The second simulation in this case is considered where the mobile manipulator is required to move from an initial state $\mathbf{q}_0 = [0\ 0\ \frac{\pi}{4}\ 0\ 0]$ to the desired goal state at $\mathbf{q}_f = [4\ 4\ \frac{\pi}{4}\ 0\ 0]$ with total time $t_f = 1.9s$ in the presence of obstacles. Figures 6.10 and 6.11 show the trajectories of the mobile platform and the end-effector obtained from the Leapfrog method and BVP4C.

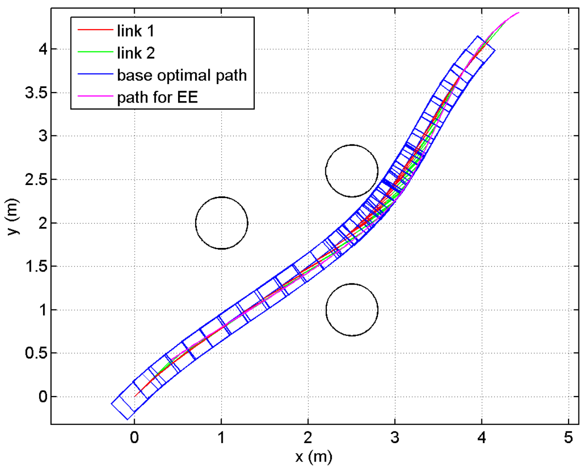


Figure 6.10: Path generated by the Leapfrog method for Case 3(b).

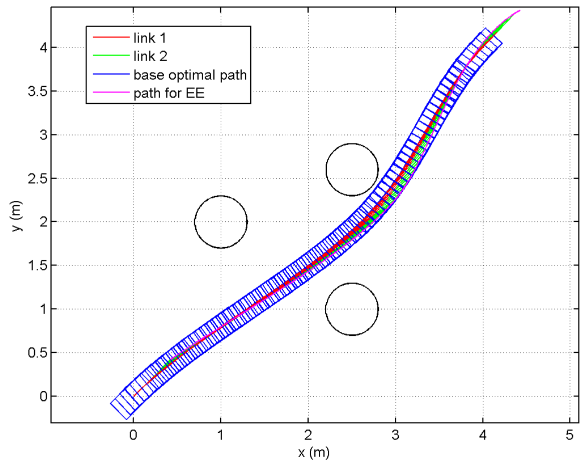


Figure 6.11: Path obtained from BVP4C for Case 3(b).

Case 4: In the final scenario, suppose that the mobile manipulator is required to move from an initial state $\mathbf{q}_0 = [-0.2 \; 0.5 \; -\frac{\pi}{2} \; -2\frac{\pi}{9} \; 5\frac{\pi}{9}]$ to the desired goal state at $\mathbf{q}_f = [1.2 \; 2 \; 2\frac{\pi}{3} \; 2\frac{\pi}{9} \; -13\frac{\pi}{15}]$ with overall time $t_f = 1.2s$ in the presence of obstacles. Figures 6.12 and 6.13 show the trajectories of the mobile platform and the end-effector obtained from the Leapfrog method and BVP4C.

In this simulation, the mobile manipulator was required to start facing in a different direction as the final state which will require a reversal motion in order

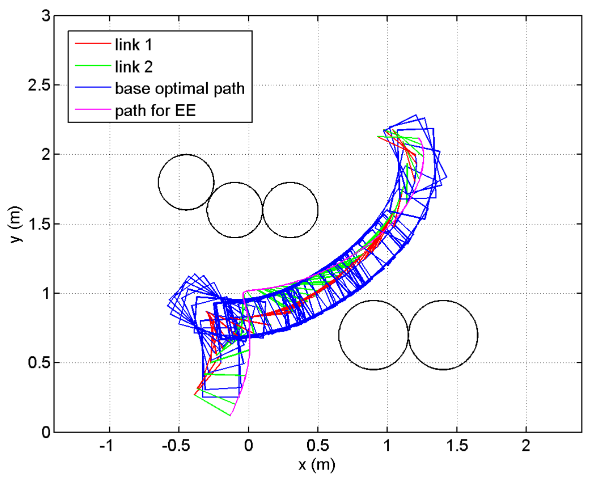


Figure 6.12: Path generated by the Leapfrog method for Case 4.

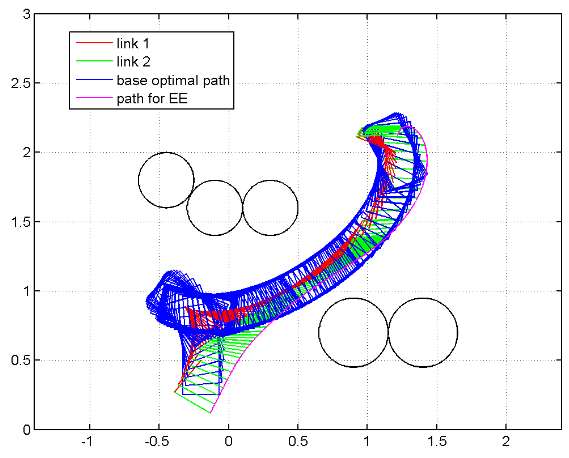


Figure 6.13: Path obtained from BVP4C for Case 4.

to get to the final state. As it can be seen in Figure 6.12, the Leapfrog method generated a different path for the end-effector as compared to the BVP4C path in Figure 6.13. The BVP4C solver tried unsuccessfully to achieve convergence for this case. It gave a warning that it needs more mesh points to get the solution to a specified accuracy. Hence the motion of the joint configurations generated by the Leapfrog method is different from the BVP4C solution.

In order to evaluate the performance for the Leapfrog and BVP4C numerical solutions, the path cost is calculated for each case presented above. This is done by taking the forward and angular velocities together with the obstacle avoidance term for each time point and used in the calculation of the integrand. The resulting discrete-time function is then integrated numerically using `trapz()` in MATLAB and the results are shown in Table 6.1.

Table 6.1: Path cost values.

Case	Leapfrog	BVP4C
1	2.46	2.46
2(a)	16.78	16.79
2(b)	15.59	15.59
3(a)	9.01	9.01
3(b)	8.55	8.55
4	11.52	18.56

As it can be seen in Table 6.1, the results obtained show that the Leapfrog method is capable of producing feasible and paths with similar path costs to BVP4C.

6.2 Conclusion

In this chapter, the optimal control problem for the motion planning for a mobile manipulator was presented in an environment with obstacles. First, an overview of related work on motion planning for mobile manipulator was given. Then an optimal motion control of the platform's kinematic system under the necessary conditions of optimality was derived and the Leapfrog method was applied to find paths for the derived system. A Gaussian potential function was applied in the cost function as an obstacle avoidance parameter. For evaluation, the paths generated by the Leapfrog method were compared to those generated by BVP4C. Simulations showed that the Leapfrog method is capable of finding critical, feasible paths in the presence of obstacles.

Chapter 7

Optimal Kinodynamic Motion Planning

Over the past decades, motion planning has gained immense popularity and importance in robotics research. Early methods solved the motion planning problem by finding feasible and optimal paths from a given initial state to a final state. In these methods, the robot kinematic or dynamic constraints were neglected which often caused the generated path to be non-smooth and not necessarily executable by the robot. To tackle this problem, Donald [14] proposed the idea of kinodynamic planning, which is achieved by incorporating the velocities and dynamic constraints during planning.

Kinodynamic motion planning is the problem of finding the optimal robot path under kinematic or dynamic constraints from a starting configuration to a goal configuration, and the path has to be collision-free. This approach increases the complexity of the motion planning problem with the introduction of high-dimensional spaces and, most importantly, constraints imposed by the system dynamics. The class of kinodynamic planning problems is at least PSPACE-hard [77] with exact, time-optimal trajectory planning in state space being NP-hard [14].

Given its importance, however, kinodynamic planning has attracted a lot of attention in the robotics community, particularly in sampling-based methods [19; 86]. Since the RRT* method [10], there has been considerable interest in solving optimal paths in the context of kinodynamic planning. With kinodynamic RRT*, the main focus is on developing an efficient steering function which involves solving a TPBVP. Solving a TPBVP for nonlinear systems remains challenging and is typically done using numerical techniques [27; 40; 66].

Some effort has been made towards developing effective steering functions for different types of dynamical systems. Some authors extended RRT* to systems where linear and quadratic functions represent dynamics and cost, by defining a suitable steering function based on the LQR principle [16; 38; 39]. Other work avoids solving a TPBVP and uses methods which find approximate paths between two states. For example, in [66], the shooting method is used to find an approximate path between two states in the tree. Despite these challenges, kinodynamic-RRT* has shown progress in successfully finding optimal paths for mobile robots [26].

Similar to RRT*, the optimal control approach is a powerful tool in finding optimal paths for mobile robots in cluttered environments. Through the work discussed in Chapter 2 on kinodynamic-RRT*, numerical methods arising from optimal control are used to find local solutions (without consideration of obstacles) for motion planning problems. With a numerical method, such as Leapfrog, a sequence of feasible local solutions that converges to a critical, and possibly globally optimal, solution is obtained. It was demonstrated in the previous chapters that the Leapfrog method can produce critical paths given two-wheeled mobile robot kinematics, which is a desirable feature for motion planning. Of particular interest in this chapter, is finding optimal trajectories for the two-wheeled mobile robot model described in Chapter 5 using the RRT* algorithm. The solutions obtained will then be compared to the Leapfrog solutions.

In this chapter, the optimal motion planning for the two-wheeled mobile robot described in Chapter 5 is considered. Given the robot kinematics model, the initial state, the goal state, and the obstacles, a kinodynamic-RRT* algorithm is tasked to find a collision-free path for the robot from the initial state to the goal state with minimum cost, given available information, and without colliding with the obstacles. The same strategy applies to the Leapfrog method. The methods are evaluated in a set of examples with a variety of obstacles. Performance comparison for optimality criteria such as path cost, runtime and a total number of tree nodes is done and compared to solutions obtained by the Leapfrog method.

7.1 Kinodynamic-RRT* Algorithm

This section gives a brief description of the kinodynamic-RRT* algorithm with reference to the RRT* algorithm described in Chapter 2, where the abstract form of the algorithm is given in Algorithm 5. The notation used in this section is adapted from the RRT* algorithm in [66].

The kinodynamic-RRT* algorithm solves an optimal motion planning problem by growing a tree $\mathcal{T} = (V, E)$, with vertex set V of states connected by edges E of the feasible path segments, to find a path that connects exactly to the goal state with minimum cost, using the basic algorithmic procedures described below.

`Sample`: The `Sample` function generates a random state $z_{rand} \in \mathcal{C}_{free}$ from the obstacle-free region of the robot configuration space.

Algorithm 5 Kinodynamic-RRT*

```
1: $\mathcal{V} \leftarrow \{z_{init}\}$ ; $E \leftarrow \emptyset$
2: $\mathcal{T} = (V, E)$;
3: while $i < N$ do
4:     $z_{rand} \leftarrow$ Sample(i);
5:     $z_{nearest} \leftarrow$ Nearest($\mathcal{T}, z_{rand}$);
6:     $z_{new} \leftarrow$ Steer($z_{nearest}, z_{rand}$);
7:     if CollisionFree($z_{new}$) then
8:         $Z_{nearby} \leftarrow$ Near($\mathcal{T}, z_{new}, |V|$);
9:         $V \leftarrow V \cup \{z_{new}\}$;
10:        $z_{min} \leftarrow$ Parent($Z_{nearby}, z_{nearest}, z_{new}$);
11:        $E \leftarrow E \cup \{(z_{min}, z_{new})\}$;
12:        $\mathcal{T} \leftarrow$ Rewire($Z_{nearby}, z_{min}, z_{new}$);
13:    end if
14: end while
15: return $\mathcal{T}$
```

`Nearest`: Given a graph $\mathcal{T} = (V, E)$ and a state z_{rand}, the function `Nearest`$(\mathcal{T}, z_{rand})$ returns a nearest state $z_1 \in V$ from which z_{rand} can be reached with the lowest cost.

`Steer`: For any given $z_{nearest}, z_{rand} \in \mathcal{C}_{free}$, `Steer`$(z_{nearest}, z_{rand})$ returns an optimal path from $z_{nearest}$ to z_{rand}, when such a path exists.

`CollisionFree`: The `CollisionFree`(z_{new}) function checks whether the trajectory lies in an obstacle-free region of the configuration space, considering the robot size (radius around the robot). It then returns true if the path lies in $\mathcal{C}_{free}$, and false otherwise.

`Near`: Given a set V of vertices in the tree and a state z_{rand}, the function `Near`$(\mathcal{T}, z_{rand}, |V|)$ computes the set of states in V that are inside the ball centered at z_{rand} and with radius r given in Eq. (2.1.1).

`Parent`: The function `Parent`$(Z_{nearby}, z_{nearest}, z_{new})$ chooses the best parent (z_{min}) from the set of neighbours, returned by the `Near` function of a new node z_{new} where the chosen parent has the lowest cost to reach z_{new}.

`Rewire`: The function `Rewire`$(Z_{nearby}, z_{min}, z_{new})$ checks if the cost to the nodes in Z_{nearby} is less through z_{new} as compared to their older costs, and if so, its parent is changed to z_{new}.

7.1.1 Kinodynamic-RRT* Implementation

This section describes the implementation of the kinodynamic-RRT* with its extension to the two-wheeled mobile robot. The differences with the kinodynamic-RRT* implemented in this chapter are given, i.e., the sampling, steering, and the near neighbour functions.

For the algorithm implementation, the sampling strategy `Sample` is done uniformly in 3-dimensional spaces of states $q = [x,\, y,\, \varphi]^T$, where the position (x, y) is sampled from a free space (in meters), and the orientation is within a range of $\varphi \in [0, 2\pi]$ rad. Also, the sampling procedure is biased to the goal which ensures that the path connects exactly to the desired goal state. Goal biasing [118] is done by attempting to connect a newly added state at the end of each new segment to the goal, given it is within a predefined vicinity. The goal biasing is recommended to be less than 10% to maintain the randomness of the search.

In searching for the near neighbours and nearby vertices within a ball radius, a Euclidean distance is considered in the `Near` and `Nearest` functions since the orientation of the robot is ignored. However, in other functions of the algorithms, the full configuration, $\mathbf{q}$, is used.

Computing the `Steer` function with the robot differential constraints can be a computational challenge since it results in solving a TPBVP. In this chapter, the shooting method [119] is used to solve the resulting TPBVP. The shooting method, however, has convergence difficulties which may lead to an infeasible path when finding a solution between two states. For the implementation of the shooting method, if the maximum number of iterations is reached without convergence, the connection between the two configurations is regarded as infeasible, and a new random node is selected.

In order to evaluate a given path between two states, the following cost functional is minimized:

$$J = \int_0^T [1 + \frac{1}{2}\mathbf{u}^T R \mathbf{u} + \sum_{i=1}^{n} F_{rep_i}(\mathbf{q})]dt. \tag{7.1.1}$$

Equation (7.1.1) represents the trade-off between the arrival time and the expanded control effort and is commonly used for kinodynamic-RRT* planning [16; 27]. In addition to the cost functional, the obstacle avoidance is represented by a Gaussian potential function as in Chapter 5.

As mentioned in the literature, kinodynamic-RRT* typically solves the TPBVP without consideration of the obstacles. However, using the cost functional that includes obstacles will allow a fair comparison of the paths obtained from both RRT* and the Leapfrog method.

7.1.2 Optimal Control Formulation

This section presents the formulation of the optimal control-based solution for a two-wheeled mobile robot as in Chapter 5. Considering the two-wheeled mobile robot kinematic model (1.2.1) and the cost functional (7.1.1), the aim is to find control input $\mathbf{u}$, and a corresponding state $\mathbf{q}$ which minimizes the cost function and satisfies the boundary conditions, $\mathbf{q}(t_0) = \mathbf{q}_0$ and $\mathbf{q}(t_f) = \mathbf{q}_f$. Following the typical procedure to generate an optimal path, the Hamiltonian function is considered as:

$$\begin{aligned} H \;=\; & \left[1 + \frac{1}{2}\left(v^T R_{11} v + w^T R_{22} w + \sum_{i=1}^{n} F_{rep_i}(\mathbf{q})\right)\right] \\ & + \lambda_1 v \cos\varphi + \lambda_2 v \sin\varphi + \lambda_3\, \omega. \end{aligned} \tag{7.1.2}$$

Using the necessary conditions for optimal control, the optimal trajectory planning for the mobile robot can be achieved by solving the following TPBVP:

$$\begin{gathered} \begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} H_{\boldsymbol{\lambda}} \\ -H_q \end{bmatrix}, \\ \mathbf{q}(t_0) = \mathbf{q}_0, \;\; \mathbf{q}(t_f) = \mathbf{q}_f. \end{gathered} \tag{7.1.3}$$

For RRT*, the solution of the TPBVP with the boundary conditions in (7.1.3) is the optimal path connecting two states in the tree.

7.2 Numerical Results

This section presents a performance comparison between the Leapfrog and RRT* algorithms, where the best path from a starting state to the desired goal state is computed. The numerical simulations approximate the solution of the derived TPBVP, similarly to Chapter 5, for the two-wheeled mobile robot. The difference in this chapter is the cost functional which penalizes the duration of the path and energy control inputs and also includes the Gaussian potential function for obstacle avoidance. In addition, it is noted that RRT* and Leapfrog have different implementation and execution procedures. However, the difference does not affect the final solution. This chapter attempts to compare these methods using measures such as path cost and runtime.

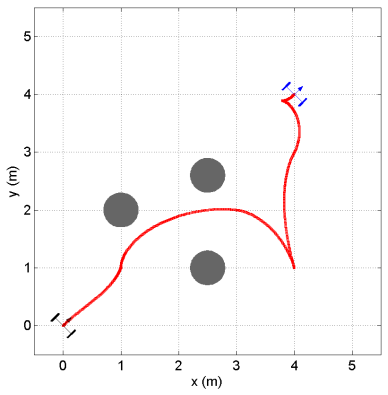
To evaluate the RRT* algorithm and the Leapfrog method for the two-wheeled mobile robot model, a set of examples in different environments is considered. The simulations are carried out in MATLAB on an Intel Core i7 computer with 8Gb RAM.

7.2.1 Kinodynamic-RRT* Simulations

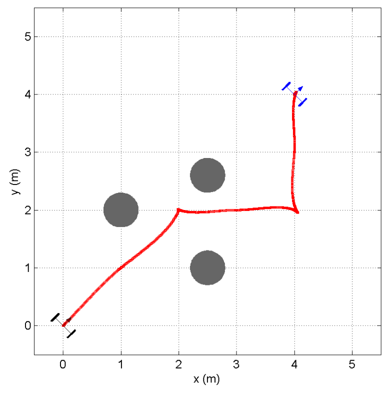
The RRT* simulations are performed using the same random seed, represented as `rng` in MATLAB. Fixing `rng` ensures that the algorithm generates the same sequence of random numbers every time it runs. For each example, the cost and computational time of a path generated within a certain number of nodes are recorded.

In the plots related to the simulations discussed in this section, the grey regions represent obstacles, and white regions are the free spaces that the robot can travel on. The starting and goal poses of the robot are represented in black and blue, respectively. The red curve represents the best path found for a given numbers of nodes.

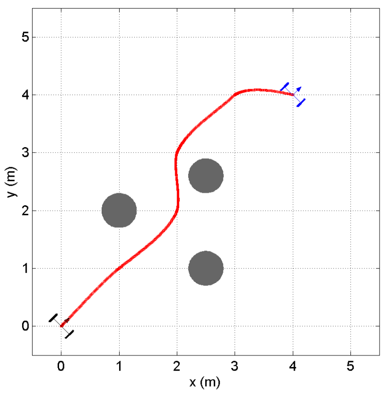
Case 1: In this simulation, the robot is required to find a path from initial state $[0, 0, \frac{\pi}{4}]$ to final state $[4, 4, \frac{\pi}{4}]$ with position limits $x \in [0, 5]$ and $y \in [0, 5]$. Figure 7.1 shows the paths generated by the RRT* amongst three obstacles.



(a) 26 nodes



(b) 200 nodes



(c) 500 nodes

Figure 7.1: Paths generated by kinodynamic-RRT* at different numbers of nodes, where Figure 7.1(a) shows the first path obtained by the algorithm.

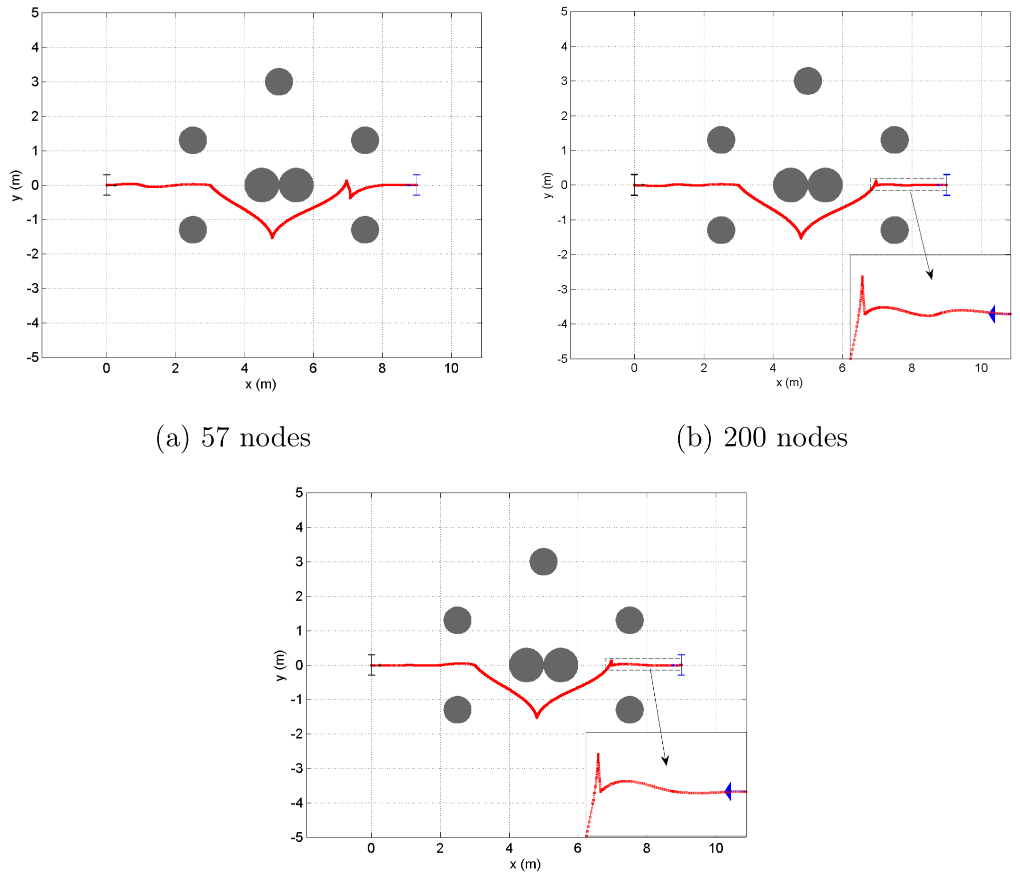
The simulations show the path progression of the algorithm from the first path reached with 26 nodes. It can be observed that the solution improved as the

Table 7.1: Properties of the solution generated by the kinodynamic-RRT* algorithm for Case 1.

No. of nodes	Path cost	CPU time (s)
26	33.60	385.31
200	24.42	5410.31
500	23.66	17519.63

number of nodes increased. Table 7.1 shows the evaluations for the simulation in Case 1 after it reaches a given maximum number of nodes.

Case 2: In this simulation, the mobile robot is expected to find a path from initial state $[0, 0, 0]$ to final state $[9, 0, \pi]$ while avoiding seven obstacles. Figure 7.2 shows collision-free paths generated by the kinodynamic-RRT* algorithm between obstacles.



(a) 57 nodes

(b) 200 nodes

(c) 500 nodes

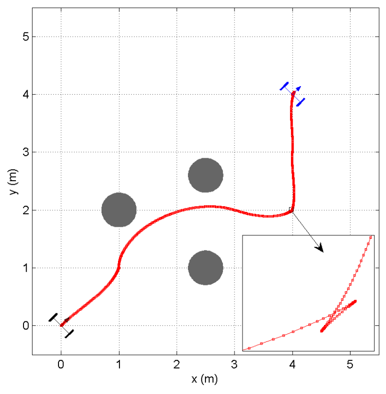
Figure 7.2: Paths resulting from kinodynamic-RRT* at different numbers of nodes, with the initial path obtained by the algorithm shown in Figure 7.2(a).

Table 7.2: Properties of the solution generated by the kinodynamic-RRT* algorithm for Case 2.

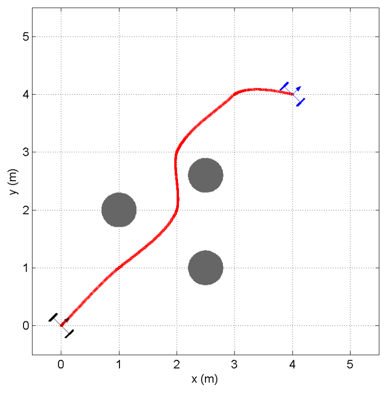
No. of nodes	Path cost	CPU time (s)
57	88.68	2494.09
200	84.58	10086.07
500	80.26	49066.53

The overall results show that the RRT* algorithm with the mobile robot system can produce improvement on the path given more iterations, even though this may increase the computational time. During the simulations, it was noted that the RRT* algorithm returns almost the same path after 500 nodes. Hence for the set of examples considered here the maximum number of nodes used is 500. However, in a case where the robot needs smooth paths due its differential limitations on sharp turns, a larger number of nodes and increased rewiring radius can be considered.

The effect of the increased radius in `Rewire` (Eq. (2.1.1)), on the path quality and computational time of the algorithm is demonstrated in Figure 7.3. For this example (Case 1), it shows that the algorithm produced a smoother (less jagged) path when $\Upsilon = 6$ with a cost of 23.66 as compared to when $\Upsilon = 4$ with a cost of 26.15, using the same number of nodes.



(a) $\Upsilon = 4$ and CPU time $= 2974.35s$



(b) $\Upsilon = 6$ and CPU time $= 20300.27s$.

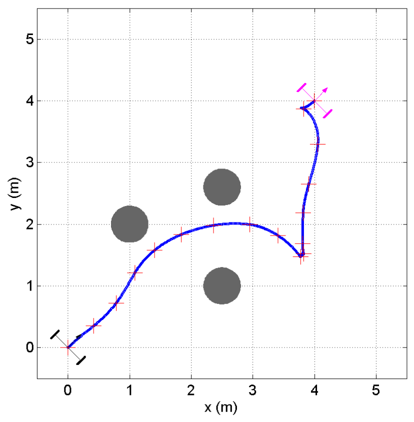
Figure 7.3: The best path generated by kinodynamic-RRT* at 500 nodes with different values of Υ.

These simulations show that using a larger value of Υ can have an efficiency trade-off. Even though it improved path cost, it also slowed down the convergence rate (measured CPU time) of the algorithm.

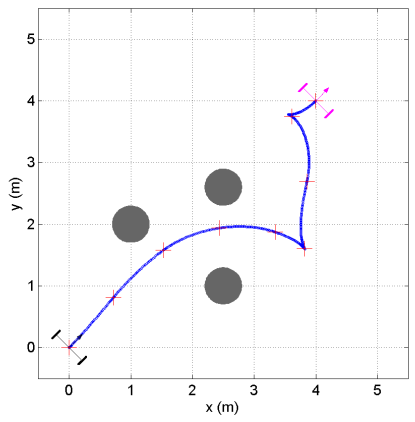
7.2.2 Leapfrog Simulations

To use the Leapfrog method, an initial feasible path is necessary and helps to provide the initial conditions for the costate variables. Hence in this section, the Leapfrog method makes use of the first path generated by the kinodynamic-RRT* as an initial feasible path. It should be noted that the initial path created by the kinodynamic-RRT* is only used for initial partition points, and Leapfrog is not used to improve kinodynamic-RRT*. Also, the time for Leapfrog execution includes the time required to generate the first kinodynamic-RRT* path, since it is used as the starting point for Leapfrog. It should be noted that the Leapfrog initial path can be computed using another algorithm, as shown in Chapter 5. The way-points which form the Leapfrog initial feasible paths are reduced using B-spline interpolation to $p = 16$, as in the previous chapter.

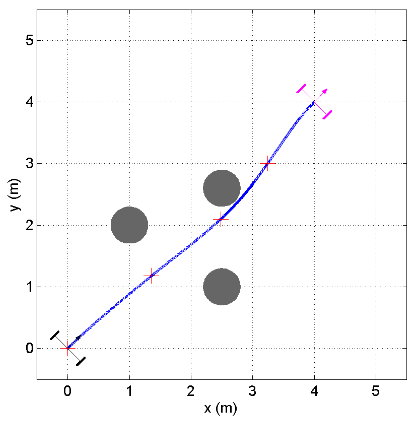
The Leapfrog simulations are generated by solving a TPBVP as in the previous section, together with the cost functional (7.1.1). The state trajectories (blue) obtained by the Leapfrog method for Case 1 and Case 2 are shown in Figure 7.4 and Figure 7.5. To further illustrate the effectiveness of the Leapfrog algorithm in each simulation, the Leapfrog iterations are shown. The simulations also display both the initial partition and the path μ_z^1 obtained as a result of the first Leapfrog iteration, where the red crosses represent partition points. The running time and the costs along each iteration of the Leapfrog algorithm are also reported.



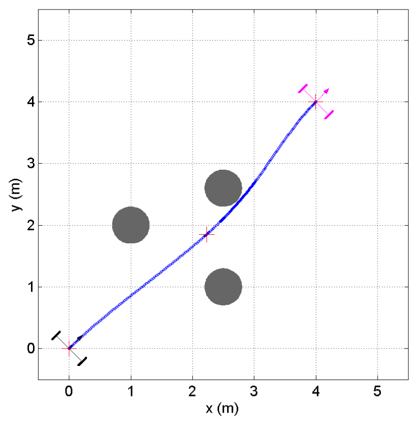
(a) $\mu_z^{(1)}$ with $p = 16$, and cost $= 28.61$.



(b) $\mu_z^{(5)}$ with $p = 8$, and cost $= 24.77$.



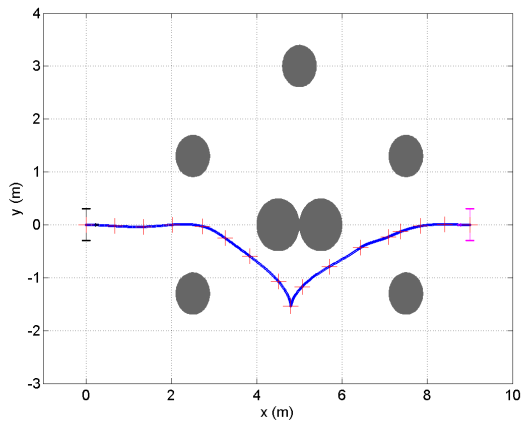
(c) $\mu_z^{(9)}$ with $p = 4$, and cost $=11.80$.



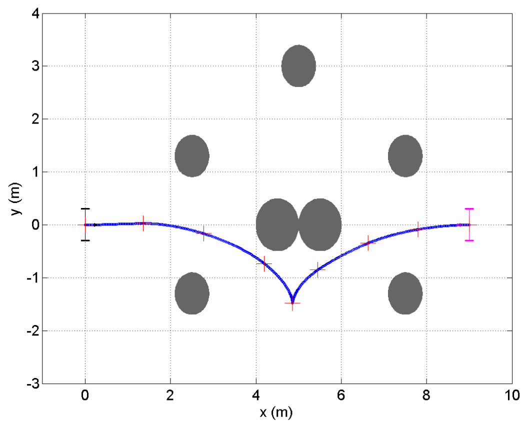
(d) $\mu_z^{(14)}$ with $p = 2$, and cost $= 11.65$.

Figure 7.4: Locally optimal paths $\mu_z^{(k)}$ for $k = 1, 5, 9$ and 14 during Leapfrog iterations for Case 1.

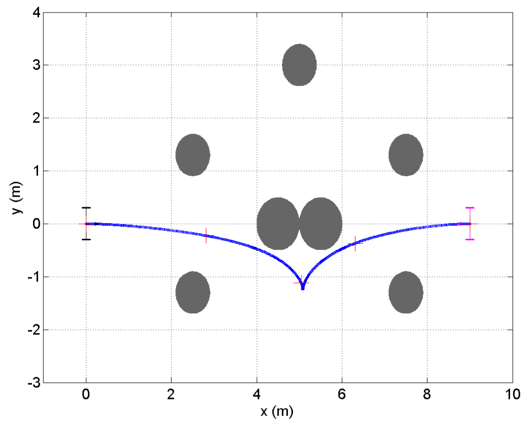
It was observed during the Leapfrog execution that at iteration $k = 9$, just after reducing the number of partition points to $p = 4$, the Leapfrog method approaches a minimum solution. The paths in Figure 7.5(c) and Figure 7.5(d) are similar but the cost is different by a small margin.



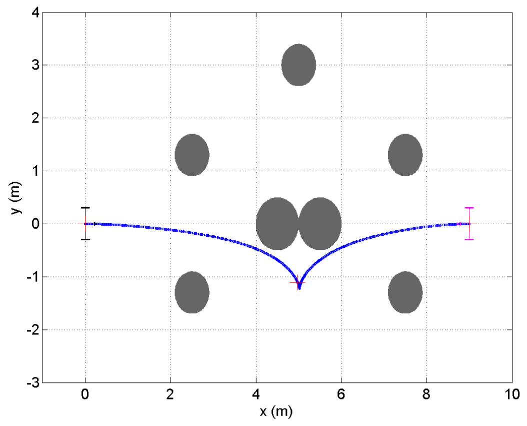
(a) $\mu_z^{(1)}$ with $p = 16$, and cost $= 29.54$.



(b) $\mu_z^{(5)}$ with $p = 8$, and cost $= 26.81$.



(c) $\mu_z^{(9)}$ with $p = 4$, and cost $=26.40$.



(d) $\mu_z^{(14)}$ with $p = 2$, and cost $= 26.40$.

Figure 7.5: Locally optimal paths $\mu_z^{(k)}$ for $k = 1, 5, 9$ and 14 during Leapfrog iterations for Case 2.

The execution time to produce the trajectories for the Leapfrog method is reported in Table 7.3 together with the path cost.

Table 7.3: Path cost and CPU time for the Leapfrog simulation results.

Case	Cost	CPU time(s)
1	11.65	732.800
2	26.40	3706.583

To conclude the Leapfrog numerical results, the following observations are made. In the last iteration ($k = 14$), after reducing the number of partitions to $p = 2$, the solution clearly converges and so the algorithm can be terminated. In this case, the path shape in the figure indicates that this solution is likely very close to the optimal solution. It can be observed from the simulations above that the cost decreases in each iteration. Also, the cost incurred along the trajectory obtained from Leapfrog is less than the cost along the initial feasible trajectory. This is some validation that the algorithm iterates from the initial feasible trajectory towards an optimal trajectory. Moreover, it is clear that the paths generated on the iterations depicted in the simulations are feasible.

7.3 Conclusion

In this chapter, a brief overview of the procedures used for the kinodynamic-RRT* algorithm under robot differential constraints was presented. Numerical simulations were conducted and showed the effectiveness and efficiency of Leapfrog and kinodynamic-RRT*. The simulation results depicted that both methods converge gradually to optimal paths. It was noted that the Leapfrog method produces less jagged and shorter paths with smaller path cost and lower execution time, as compared to kinodynamic-RRT*.

Further tests could be conducted in the future to highlight the capability of the Leapfrog method in generating optimal and collision-free paths in higher-dimensional environments.

Chapter 8

Conclusion

In this dissertation, an optimal control approach based on the Leapfrog algorithm is proposed for motion planning in mobile robotic systems, and in particular, for a two-wheeled mobile robot and a mobile manipulator. The aim is to find optimal trajectories from a given initial state to a desired goal state in the presence of obstacles. In general, this approach employs Pontryagin's minimum principle to find the state and costate equations which form a two-point boundary value problem (TPBVP) for mobile robot optimal motion control. The Leapfrog method is investigated to find numerical solutions to the resulting TPBVP. The purpose of this chapter is to summarize the contributions presented in this dissertation underlying the problem described above and also to give some perspective for future work.

An overview of some of the well-known motion planning algorithms is given in Chapter 2. These methods include the families of sampling-based, graph search and potential fields methods. The chapter also gives a short introduction on the sampling-based approach, RRT*, that addresses mobile kinematic or dynamic constraints (kinodynamic constraints, in short). Even though these methods are widely used to plan sub-optimal and optimal trajectories, they have limitations. Alternative to the traditional planning techniques is the use of optimal control to find optimal paths for mobile robot motion planning. Chapter 3 describes the mathematical models to formulate the optimal control problem and discusses numerical methods including the Leapfrog method.

Chapter 4 proposes Leapfrog, a method for solving nonlinear optimal control problems, to solve the motion planning problem for a two-wheeled mobile robot. Numerical results are presented and show that the Leapfrog method is capable of finding optimal trajectories for the two-wheeled mobile robot kinematic model. Also, the Leapfrog method manages to find paths where the BVP4C method, used for comparison, failed. Path following experiments undertaken on a Pioneer 3-DX mobile robot also show that the trajectories generated by Leapfrog are drivable without any need for post-processing.

Chapter 5 deals with determining optimal paths for the two-wheeled mobile robot in an environment with obstacles. Obstacle avoidance is represented by potential fields and added to the cost functional. This approach allows the robot to plan an optimal collision-free path through static obstacles which are represented by Gaussian functions. This chapter also focuses on further investigating parts of the Leapfrog method, such as the initial feasible path, the initial guess for costate variables and the feasibility of the trajectory generated by Leapfrog.

In the first part of Chapter 5, optimal paths in the presence of obstacles are produced using the Leapfrog method. Due to the nature of the kinematic model for the robot, the affine approximation embedded in Leapfrog does not always converge to a solution in the sub-problem. A gradient-based approach is used to obtain the initial condition for the costates.

The second part of Chapter 5 investigates the methods for obtaining the initial feasible path for the Leapfrog initial partition. It is noted that using the path generated by different path planners as an initial partition does not affect the solution obtained by the Leapfrog method. It is also emphasized that the Leapfrog method is not used to improve these paths but only uses them for initialization. This is illustrated through simulations.

Lastly, the possibility that the Leapfrog method may not always generate collision-free paths is investigated. It is shown through simulations (Section 5.5) that the local solutions obtained by Leapfrog in the subdivisions can collide with obstacles for some environments. It is noted that the paths can collide with obstacles when the Gaussian parameters are not properly selected for each example. Selecting the Gaussian parameters must be done prior to executing Leapfrog.

The work presented in Chapters 4 and 5 involves the kinematic model of a two-wheeled mobile robot, and the Leapfrog method gives promising results. This caused interest to extend the work and implement the proposed method for a robot platform with a higher-dimensional planning requirement (5 DoF), that is, the mobile manipulator. This is achieved in Chapter 6 where the optimal motion planning problem for the mobile manipulator is considered in the presence of obstacles. To guarantee collision-free paths, additional constraints in the obstacle avoidance formulation are considered. Through numerical simulations, it is demonstrated that the Leapfrog method generates collision-free trajectories on each iteration of its computation for the mobile manipulator. As with the previous chapters, Leapfrog solutions are compared to those generated by BVP4C. Though BVP4C has difficulties with the initial guess for costate values for some cases, the solver converges towards optimal collision-free paths with similar cost and in less execution time compared to Leapfrog. For some cases, however, it is observed that the Leapfrog method generated lower cost paths as compared to BVP4C paths.

In Chapter 7, a general framework for the sampling-based method, RRT*, concerning kinodynamic planning is given. The basic concepts and procedures structuring the algorithm are explained. Numerical simulations for the two-wheeled mobile robot using kinodynamic-RRT* are conducted and compared to the Leapfrog method. For a fair comparison between the two methods, the TPBVP considered in this chapter includes obstacle avoidance, as in the previous chapters for both methods. Also, the execution of the Leapfrog method's initial path is included in the algorithm's computational time. It is shown through the simulations that, even though RRT* can find asymptotically optimal paths in high-dimensional spaces, the Leapfrog method produces better paths in terms of cost and within a shorter time, in the environments tested.

8.1 Discussion

This section focuses on some of the issues that were observed when finding optimal trajectories for mobile robots using the Leapfrog method. Simulation and experimental results show that the Leapfrog method is capable of finding kinematically feasible paths in motion planning for wheeled mobile robots.

The following pointers provide practical considerations.

(i) **Feasible initial trajectory**

Choosing a feasible path for obstacle-free situations like those in Chapter 4 is relatively simple, and a straight line between the initial and final states can be used for this purpose. For the planning problem with obstacles such as in Chapter 5, the feasible initial paths can be generated using different path planners, for example A* or RRT. The challenge is that every time the RRT algorithm is executed, it produces a different path or no solution if the goal is not reached. As such, not all paths generated by RRT can be used or tested with Leapfrog. It is shown in Chapter 5 that the Leapfrog method does not necessarily depend on the method used to generate the initial path.

With Leapfrog, the number of partition points on the first iteration depends on the initial feasible path constructed and the system model being solved. The challenge is that paths generated by RRT can have many way-points (points that form a trajectory). Choosing the number of partition points has a trade-off between using more partitions points so that the sub-problem solutions are simple, and convergence time. A small number of points is better for convergence time, but it is important that there are enough points so that the sub-problems are easily solved, otherwise convergence will be slow. This motivated the use of a B-spline approach to interpolate RRT partition points where a relatively small number of partition points (e.g., $p = 8$) are chosen for the initial feasible paths in Chapters 4 and 5. In Chapters 6 and 7 the number of partition points on the first iteration is chosen as $p = 16$ in order for the method to converge to a better path.

(ii) **Partition points reduction**

Analyzing when to reduce the partition points required for the algorithm to converge to the optimal trajectory, is a challenge. With the Leapfrog method, the number of partition points needs to eventually be reduced to $p = 2$ as the solution is approached. If the partition points are reduced appropriately, the convergence of Leapfrog towards a critical trajectory is guaranteed [4]. However, reducing the number of partition points too early or late in the process might lead to algorithm failure or produce sub-optimal paths. Currently, trial and error is used to reduce the partition points. One way of overcoming this is to monitor the change in the path cost after each iteration is achieved and reduce p if the difference is less than a threshold. This may add to the computational time of the method's execution as the number of iterations might increase.

(iii) **Initial guess**

Solving a TPBVP requires boundary conditions on the states and costates. The boundary conditions for the states are known whereas there is no prior information regarding the costates. With the Leapfrog method, the initial conditions of the costates are not required to be estimated. As part of the algorithm, an affine approximation helps in choosing good initial guesses for the costates. In the case of the examples of the wheeled mobile robot motion planning considered for this dissertation, however, the affine approximation does not give a suitable costate estimate. A gradient-based approach is implemented to obtain the initial condition for the costates.

(iv) **Feasible paths along each Leapfrog iteration**

Solving the motion planning for wheeled mobile robots entails generating feasible paths between initial and final configurations. This is one of the benefits one gets from the Leapfrog method. In Chapter 4 it is shown that during the Leapfrog iterations the trajectories produced by the method are feasible in each iteration. Should the Leapfrog method fail, or if there is a need to halt execution before it reaches the final iteration for some reason, the intermediate paths generated are still feasible, though possibly sub-optimal. However, this might not always be true for cases where obstacles are considered. As shown in Chapter 5, for some scenarios the local solutions are not collision-free. It is also shown that by increasing the Gaussian parameter A_{rep} the algorithm will move away from obstacles as the cost of going onto the obstacle will be high.

(v) **The Leapfrog method compared to other methods**

- **BVP4C**

  For this dissertation, the BVP4C is used as a reference or ground truth to fairly evaluate the Leapfrog method. In Chapter 4 (Case 3) it is noticed that the BVP4C method generates a singular Jacobian error while attempting to solve the TPBVP. This is traced to a problem with the initial guess of the solution. It is noted that an inappropriate estimate can also cause the solving algorithm to fail ultimately, yielding no result, as in this case.

  Given the limited information as to how the states vary over the solution space, the initial estimate provided to the BVP4C solver is arbitrary. The initial guess is used to give the solver a starting point and can affect the resulting solution so much so that a different solution may be found by merely varying the initial guess. If a different initial guess is provided, the solver may successfully find a solution.

  Apart from providing a good guess for the boundary value problem, setting tolerances can also be an issue when using BVP4C. In the experiments of Chapter 6 (Case 4) the solver struggles to find an accurate solution and returns a warning. Even though a solution is found, it has an offset which results in a different solution as compared to the Leapfrog method.

  The numerical solutions presented in this dissertation show that the BVP4C solver can achieve optimal paths in less time than the Leapfrog method. However, the BVP4C solver suffers from the initialization of costates that results in poorer (inaccurate) solutions or no solution, while the Leapfrog method does not have this limitation.

- **Optimal kinodynamic planning**

  Numerical studies are done for the kinodynamic-RRT* method in Chapter 7 and the results are compared to those obtained by the Leapfrog method. From the simulations, it is evident that the kinodynamic-RRT* generates a collision-free path and improves it towards optimality as the number of iterations increases. However, it is observed that the algorithm has a limitation of slow convergence to an optimal solution. As a result, it consumes a lot of memory and time as the tree becomes larger.

  The numerical results also show that the Leapfrog method produces not only lower-cost paths but with better smoothness and in less time compared to kinodynamic-RRT*.

8.2 Concluding Summary

Overall, this dissertation presents an optimal control approach to solve the motion planning problems for mobile robot systems. The key motivation for using an optimal control approach is the focus on producing optimal motion which incorporates differential constraints, smoothness, and obstacle avoidance in a mathematically precise objective. Hence, the Leapfrog method is proposed as a tool to solve the motion planning problem for a two-wheeled mobile robot and mobile manipulator. This led to the key contribution for this dissertation which is the first application of the Leapfrog method to motion planning of wheeled mobile robots.

The benefits of using the Leapfrog method is that it initializes easily using a feasible, sub-optimal path, and can find kinematically feasible, critical, and possibly globally optimal paths. In addition, it converges to critical paths that are accurate, as seen in the comparisons with BVP4C. Moreover, the Leapfrog method finds better solutions that are suitable for typical robot trajectory following controllers in less time as compared with kinodynamic-RRT*.

The Leapfrog method, however, inherits some limitations from the optimal control approach in that globally optimal paths are not always guaranteed, though locally optimal, feasible solutions can be achieved, including in environments cluttered with obstacles.

8.3 Future Work

At the end of this dissertation, several matters remain unsolved, and further developments in motion planning for mobile robots using the Leapfrog method can be made. In this work, the kinematics of the wheeled mobile robots are considered. However, it would be interesting to investigate extending the motion planning approach to include dynamic models. With dynamics, the problem additionally involves inertia, forces, torque and other components of the robot motion. Dynamic robot models are better than the kinematic models in terms of reliability, speed, and accuracy.

The motion planning presented herein develops optimal paths for mobile robots on a plane. Even though this assumption works well for many indoor environments, taking into account a 3D space is needed. This will be an improvement, especially in the mobile manipulator motion planning problem because the workspace will be extended for the robot to make more motions. Also, increasing the number of degrees of freedom (for example, 7 DoF) for the manipulator arm would be interesting to investigate as more tasks can be done.

In motion planning for real-world applications, obstacles are represented in various forms. In this work, obstacles are modelled as circular obstacles because the formulation is simple and requires fewer computations as compared to other obstacle shapes. Including obstacles with arbitrary shape or size would improve the approach and also allow the Leapfrog method to be tested in the same planning scenarios as with other planning methods. Moving to arbitrarily shaped obstacles may require obstacle avoidance methods other than Gaussian potential fields, to be investigated for use with Leapfrog.

List of References

[1] LaValle, S.M.: *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006. ISBN 0521862051.

[2] Choset, H.: Robotic motion planning: Potential functions. Tech. Rep., Robotics Institute, Carnegie Mellon University, 2010.

[3] Subchan, S. and Zbikowski, R.: *Computational Optimal Control: Tools and Practice*. John Wiley & Sons, Ltd, 2009. ISBN 978-0-470-71440-9.

[4] Kaya, C.Y. and Noakes, L.: The leapfrog algorithm for optimal control. *Society for Industrial and Applied Mathematics Journal of Numerical Analysis (SIAM)*, vol. 46, no. 6, pp. 2795–2817, 2008.

[5] Gènèration robots: Pioneer p3-dx mobile robot. 2014.
Available at: `http://www.generationrobots.com`

[6] Paden, B., Cap, M., Yong, S.Z., Yershov, D. and Frazzoli, E.: A survey of motion planning and control techniques for self-driving urban vehicles. In: *IEEE Transactions on Intelligent Vehicles*, vol. 1, pp. 33–55. 2016.

[7] Gasparetto, A., Boscariol, P., Lanzutti, A. and Vidoni, R.: Path planning and trajectory planning algorithms: A general overview. *Motion and Operation Planning of Robotic Systems. Mechanisms and Machine Science*, vol. 29, pp. 3–27, 2015.

[8] Hart, P.E., Nilsson, N.J. and Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, vol. SSC-4, no. 2, pp. 100–107, 1968.

[9] Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, April 1986.

[10] Karaman, S. and Frazzoli, E.: Incremental sampling-based algorithms for optimal motion planning. In: *Proceedings of Robotics: Science and Systems*, vol. abs/1005.0416. 2010.

[11] Pan, J., Zhang, L. and Manocha, D.: Collision-free and curvature-continuous path smoothing in cluttered environments. In: *Robotics: Science and Systems VII*, vol. 7, pp. 233–240. MIT Press Journals, 2012.

[12] Karaman, S. and Frazzoli, E.: Sampling-based algorithms for optimal motion planning. *International Journal Robotics Research*, vol. 30, no. 7, pp. 846–894, June 2011.

[13] Karaman, S. and Frazzoli, E.: Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[14] Donald, B., Xavier, P., Canny, J. and Reif, J.: Kinodynamic motion planning. *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, November 1993. ISSN 0004-5411.

[15] Karaman, S. and Frazzoli, E.: Optimal kinodynamic motion planning using incremental sampling-based methods. In: *Conference on Decision and Control (CDC)*, pp. 7681–7687. IEEE, 2010.

[16] Webb, D.J. and van den Berg, J.: Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints. *CoRR*, vol. abs/1205.5088, 2012.

[17] Karaman, S. and Frazzoli, E.: Sampling-based optimal motion planning for non-holonomic dynamical systems. In: *2013 IEEE International Conference on Robotics and Automation*, pp. 5041–5047. 2013.

[18] Jeon, J.H., Cowlagi, R.V., Peters, S.C., Karaman, S., Frazzoli, E., Tsiotras, P. and Iagnemma, K.: Optimal motion planning with the half-car dynamical model for autonomous high-speed driving. In: *American Control Conference (ACC)*, pp. 188–193. 2013.

[19] LaValle, S.M. and Kuffner, J.J.: Randomized kinodynamic planning. *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[20] Kirk, D.: *Optimal control theory an introduction*. Prentice-Hall, 1998.

[21] Souères, P. and Boissonnat, J.-D.: Optimal trajectories for nonholonomic mobile robots. In: *Laumond J.P. (eds) Robot Motion Planning and Control*, vol. 229, pp. 93–169. Springer-Verlag, 1998.

[22] Zhengxiong, G. and Xinsheng, G.: A particle swarm optimization for the motion planning of wheeled mobile robot. In: *Proceedings of the 8th World Congress on Intelligent Control and Automation*, pp. 2410–2414. 2010.

[23] Korayem, M., Nazemizadeh, M., Binabaji, H. and Azimirad, V.: Optimal motion planning of non-holonomic mobile robots in presence of multiple obstacles. In: *Int. Conf. Emerg. Trends in Robot. Comm. Tech.*, pp. 269–272. 2010.

[24] Kaya, C.Y. and Noakes, J.L.: Geodesics and an optimal control algorithm. *Proceedings of the 36th IEEE Conference on Decision and Control (CDC)*, pp. 4918–4919, 1997.

[25] Korayem, M., Nazemizadeh, M. and Nohooji, H.: Optimal point to point motion planning of nonholonomic mobile robots in the presence of multiple obstacles. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 36, pp. 221–232, 2014.

[26] Park, J.J. and Kuipers, B.: Feedback motion planning via non-holonomic RRT* for mobile robots. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4035–4040. 2015.

[27] Ha, J., Choi, H. and Jeon, J.H.: Iterative methods for efficient sampling-based optimal motion planning of nonlinear systems. *CoRR*, vol. abs/1603.04112, 2016. `1603.04112`.

[28] Murray, R.M., Sastry, S.S. and Zexiang, L.: *A Mathematical Introduction to Robotic Manipulation.* 1st edn. CRC Press, Inc., 1994. ISBN 0849379814.

[29] Siegwart, R., Nourbakhsh, I.R. and Scaramuzza, D.: *Introduction to Autonomous Mobile Robots.* Bradford Company Scituate, 2004.

[30] Latombe, J.-C.: *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[31] Dijkstra, E.W.: A note on two problems in connexion with graphs. *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[32] Russell, S. and Norvig, P.: *Artificial Intelligence: A Modern Approach.* 3rd edn. Prentice Hall Press, 2009.

[33] Kavraki, L., Svestka, P., Latombe, J.-C. and Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In: *IEEE International Conference on Robotics and Automation*, vol. 12, pp. 566–580. 1996.

[34] Karaman, S., Walter, M.R., Perez, A., Frazzoli, E. and Teller, S.: Anytime motion planning using the RRT*. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1478–1483. May 2011.

[35] Lavalle, S.M. and Kuffner Jr., J.J.: Rapidly-exploring random trees: Progress and prospects. In: *Algorithmic and Computational Robotics: New Directions*, pp. 293–308. 2000.

[36] Islam, F., Nasir, J., Malik, U., Ayaz, Y. and Hasan, O.: RRT*-smart: Rapid convergence implementation of RRT* towards optimal solution. *2012 IEEE International Conference on Mechatronics and Automation*, pp. 1651–1656, 2012.

[37] Noreen, I., Khan, A. and Habib, Z.: Optimal path planning using RRT* based approaches : A survey and future directions. *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, pp. 97–107, 2016.

[38] Perez, A., Platt, R., Konidaris, G., Kaelbling, L. and Lozano-Perez, T.: LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2537–2542. May 2012.

[39] Goretkin, G., Perez, A., Platt, R. and Konidaris, G.: Optimal sampling-based planning for linear-quadratic kinodynamic systems. In: *2013 IEEE International Conference on Robotics and Automation*, pp. 2429–2436. May 2013. ISSN 1050-4729.

[40] Ha, J., Lee, J. and Choi, H.: A successive approximation-based approach for optimal kinodynamic motion planning with nonlinear differential constraints. In: *52nd IEEE Conference on Decision and Control*, pp. 3623–3628. Dec 2013. ISSN 0191-2216.

[41] Siciliano, B. and Khatib, O.: *Springer Handbook of Robotics*. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 354023957X.

[42] Lumelsky, V.J. and Stepanov, A.A.: Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, vol. 2, no. 1-4, pp. 403–430, November 1987. ISSN 0178-4617.

[43] Lumelsky, V.J. and Skewis, T.: Incorporating range sensing in the robot navigation function. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 5, pp. 1058–1069, 1990.

[44] Oroko, J. and Nyakoe, G.: Obstacle avoidance and path planning schemes for autonomous navigation of a mobile robot: A review. In: *Proceedings of the 2012 Mechanical Engineering Conference on Sustainable Research and Innovation*, vol. 4, pp. 314–318. 2012.

[45] Laumond, J.-P.: *Robot Motion Planning and Control.* Springer-Verlag New York, Inc., 1998.

[46] Guo, J., Gao, Y. and Cui, G.: Path planning of mobile robot based on improved potential fields. *Information Technology Journal*, vol. 12, no. 11, pp. 2188–2194, 2013.

[47] Rimon, E. and Koditschek, D.: Exact robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, 1992.

[48] Kim, J.-O. and Khosla, P.: Real-time obstacle avoidance using harmonic potential functions. *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 338–349, 1992.

[49] Borenstein, J. and Koren, Y.: Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989. ISSN 0018-9472.

[50] Borenstein, J. and Koren, Y.: The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[51] Biglarbegian, M.: Toward more efficient methods for path planning of mobile robots:simplified non-convex constraints. 2011.

[52] Howard, T., Green, C. and Kelly, A.: Receding horizon model-predictive control for mobile robot navigation of intricate paths. In: *Proceedings of the 7th International Conferences on Field and Service Robotics*, vol. 62, pp. 69–78. Springer, 2010.

[53] Strongin, R.G. and Sergeyev, Y.D.: *Global Optimization with Non-Convex Constraints - Sequential and Parallel Algorithms (Nonconvex Optimization and Its Applications Volume 45) (Nonconvex Optimization and Its Applications).* Springer-Verlag, Berlin, Heidelberg, 2000. ISBN 0792364902.

[54] Korayem, M., Nazemizadeh, M. and Azimirad, V.: Optimal trajectory planning of wheeled mobile manipulators in cluttered environments using potential functions. *Scientia Iranica*, vol. 18, no. 5, pp. 1138–1147, Oct 2011.

[55] Mohamed, A., Ren, J., Sharaf, A.M. and Gindy, M.E.: Optimal path planning for unmanned ground vehicles using potential field method and optimal control method. *International Journal of Vehicle Performance*, vol. 4, no. 1, pp. 1–14, 2018.

[56] Elbanhawi, M. and Simic, M.: Sampling-based robot motion planning: A review. *IEEE Access*, vol. 2, pp. 56–77, 2014.

[57] Dubins, L.E.: On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, vol. 79, pp. 497–516, 1957.

[58] Reeds, J.A. and Shepp, L.A.: Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[59] Sussmann, H. and Tang, G.: Shortest paths for the Reeds-Shepp car: A worked out example of the use of geometric techniques in nonlinear optimal control. Tech. Rep. SYNCON 91-10, Dept. of Mathematics, Rutgers University, Piscataway, NJ, 1991.

[60] Boissonnat, J.-D., Cérézo, A. and Leblond, J.: Shortest paths of bounded curvature in the plane. *Journal of Intelligent and Robotic Systems*, vol. 11, pp. 5–20, 1994.

[61] Betts, J.T.: *Practical methods for optimal control using nonlinear programming*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA, 2001. ISBN 0-89871-488-5.

[62] Rao, A.V.: A survey of numerical methods for optimal control. In: *AAS/AIAA Astrodynamics Specialist Conference*, vol. 135, pp. 497–528. 2009.

[63] Betts, J.T.: Survey of numerical methods for trajectory optimization. *Journal of Guidance Control Dynamics*, vol. 21, pp. 193–207, 1998.

[64] Howard, T.M. and Kelly, A.: Optimal rough terrain trajectory generation for wheeled mobile robots. *I. J. Robotics Res.*, vol. 26, pp. 141–166, 2007.

[65] Ferguson, D., Howard, T. and Likhachev, M.: Motion planning in urban environments: Part i. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1063–1069. 2008.

[66] Jeon, J.H., Karaman, S. and Frazzoli, E.: Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*. In: *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pp. 3276–3282. 2011. ISBN 978-1-61284-800-6.

[67] Keller, H.B.: *Numerical Solution of Two Point Boundary Value Problems*. Society for Industrial and Applied Mathematics (SIAM), 1976.

[68] Vamsikrishna, A., Mahindrakar, A.D. and Tiwari, S.: Numerical and experimental implementation of leapfrog algorithm for optimal control of a mobile robot. In: *2017 Indian Control Conference (ICC)*, pp. 123–128. 2017.

[69] Matebese, B.T., Withey, D.J. and Banda, M.K.: Application of the leapfrog method to robot path planning. In: *IEEE International Conference Information and Automation (ICIA)*, pp. 710–715. 2014.

[70] Matebese, B., Withey, D. and Banda, M.K.: Modified Newton's method in the leapfrog method for mobile robot path planning. In: *Artificial Intelligence and Evolutionary Computations in Engineering Systems*, pp. 71–78. Springer, Singapore, 2018.

[71] Yagnik, D., Ren, J. and Liscano, R.: Motion planning of a multi-link robot with artificial potential field method and modified simulated annealing. In: *MESA Conference on Electrical and Comp. Engineering*, pp. 421–427. 2010.

[72] h. Jeon, J., Karaman, S. and Frazzoli, E.: Optimal sampling-based feedback motion trees among obstacles for controllable linear systems with linear constraints. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4195–4201. May 2015. ISSN 1050-4729.

[73] Kaya, C.Y. and Noakes, J.L.: The leap-frog algorithm and optimal control: Background and demonstration. *Proceedings of International Conference on Optimization and Applications (ICOTA)*, pp. 835 – 842, 1998.

[74] Matebese, B., Withey, D. and Banda, M.K.: Initialization of the leapfrog algorithm for mobile robot path planning. In: *Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech)*, pp. 1–6. 2016.

[75] Matebese, B., Withey, D. and Banda, M.K.: Optimal paths for a mobile manipulator using the leapfrog method. In: *2019 Southern African Universities Power Engineering Conference/Robotics and Mechatronics/Pattern Recognition Association of South Africa (SAUPEC/RobMech/PRASA)*, pp. 42–48. 2019.

[76] Matebese, B., Withey, D. and Banda, M.K.: Path planning with the leapfrog method in the presence of obstacles. In: *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 613–618. 2016.

[77] Reif, J.H.: Complexity of the mover's problem and generalizations. In: *20th Annual Symposium on Foundations of Computer Science (SFCS 1979)*, pp. 421–427. Oct 1979. ISSN 0272-5428.

[78] Cheng, P. and LaValle, S.M.: Resolution completeness for sampling-based motion planning with differential constraints. In: *Submitted to the International Journal of Robotics Research*. 2004.

[79] Lavalle, S.M.: Rapidly-exploring random trees: A new tool for path planning. Tech. Rep., Computer Science Dept., Iowa State University, 1998.

[80] Lindemann, S.R. and LaValle, S.: Current issues in sampling-based motion planning. In: Dario, P. and Chatila, R. (eds.), *Robotics Research. The Eleventh International Symposium*, pp. 36–54. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31508-7.

[81] Shkolnik, A.C. and Tedrake, R.: Path planning in 1000+ dimensions using a task-space voronoi bias. In: *International Conference on Robotics and Automation (ICRA)*, pp. 2061–2067. IEEE, 2009.

[82] Shkolnik, A.C. and Tedrake, R.: Sample-based planning with volumes in configuration space. *CoRR*, vol. abs/1109.3145, 2011.

[83] Ferguson, D. and Stentz, A.T.: Anytime RRTs. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5369–5375. October 2006.

[84] Abbasi-Yadkori, Y., Modayil, J. and Szepesvri, C.: Extending rapidly-exploring random trees for asymptotically optimal anytime motion planning. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 127–132. 2010.

[85] Hsu, D., Kavraki, L.E., Latombe, J.-C., Motwani, R. and Sorkin, S.: On finding narrow passages with probabilistic roadmap planners. In: *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics : The Algorithmic Perspective*, WAFR '98, pp. 141–153. A. K. Peters, Ltd., Natick, MA, USA, 1998. ISBN 1-56881-081-4.

[86] Kindel, R., Hsu, D., Latombe, J. and Rock, S.: Kinodynamic motion planning amidst moving obstacles. In: *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 537 – 543. 2000. ISSN 1050-4729.

[87] Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T.: *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1988. ISBN 0-521-35465-X.

[88] Hansen, E.A. and Zhou, R.: Anytime heuristic search. *Journal of Artificial Intelligence Research (JAIR)*, vol. 28, pp. 267–297, 2007.

[89] Likhachev, M., Gordon, G. and Thrun, S.: ARA*: Anytime A* with provable bounds on sub-optimality. In: *Advances in Neural Information Processing Systems 16*, pp. 767–774. MIT Press, 2004.

[90] Likhachev, M., Ferguson, D., Gordon, G., Stentz, A. and Thrun, S.: Anytime dynamic A*: An anytime, replanning algorithm. In: *Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling*, pp. 262–271. AAAI Press, 2005.

[91] Likhachev, M., Ferguson, D., Gordon, G., Stentz, A. and Thrun, S.: Anytime search in dynamic graphs. *Artificial Intelligence*, , no. 172, pp. 1613–1643, 2008.

[92] Koren, Y. and Borenstein, J.: Potential field methods and their inherent limitations for mobile robot. *Neoplasia*, pp. 1398–1404, 1991.

[93] Ge, S. and Cui, Y.: New potential functions for mobile robot path planning. *IEEE Transactions Robotics and Automation*, vol. 16, no. 5, pp. 615–620, 2000.

[94] Ge, S. and Cui, Y.: Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, vol. 13, no. 3, pp. 207–222, 2002.

[95] Ren, J. and McIsaac, K.: A hybrid-systems approach to potential field navigation for a multi-robot team. *IEEE Transactions on Robotics and Automation*, vol. 3, pp. 3875–3880, 2003.

[96] Jia, Y., Yin, G., Zhao, L., Li, X. and Sun, S.: Path planning for mobile robot using the novel repulsive force algorithm. In: *International Conference on Computer Sciences and Automation Engineering*, pp. 575–578. 2015.

[97] Keij, J.J.A.M., Lambrechts, D.I.P.F. and Witternan, D.I.W.J.: Dynamics and control group ii obstacle avoidance for wheeled mobile robotic systems. 2003.

[98] Bischoff, R., Huggenberger, U. and Prassler, E.: Kuka youbot - a mobile manipulator for research and education. In: *2011 IEEE International Conference on Robotics and Automation*, pp. 1–4. 2011.

[99] Hvilshøj, M., Bøgh, S., Nielsen, O. and Madsen, O.: Autonomous industrial mobile manipulation (AIMM): past, present and future. *Industrial Robot: International Journal of Robotics Research and Application*, vol. 39, no. 2, pp. 120–135, 2012.

[100] Naidu, D.S. and Naidu, S.: *Optimal Control Systems*. CRC Press, Inc., Boca Raton, FL, USA, 2002.

[101] Pontryagin, L.S., Boltianski, V.G., Gamkrelidze, R.V., Mishchenko, E.F. and Brown, D.E.: *The Mathematical Theory of Optimal Processes*. International series of monographs in pure and applied mathematics. Pergamon Press, 1964. A Pergamon Press book.

[102] Lastman, G.J.: A shooting method for solving two-point boundary-value problems arising from non-singular bang-bang optimal control problems. *International Journal of Control*, vol. 27, no. 4, pp. 513–524, 1978.

[103] Morrison, D.D., Riley, J.D. and Zancanaro, J.F.: Multiple shooting method for two-point boundary value problems. *Commun. ACM*, vol. 5, no. 12, pp. 613–614, December 1962. ISSN 0001-0782.

[104] Shampine, L.F., Kierzenka, J.A. and Reichelt, M.W.: Solving boundary value problems for ordinary differential equations in MATLAB with BVP4C. 2000.
Available at: `ftp://ftp.mathworks.com/pub/doc/papers/bvp/`

[105] Arian, A., Mostafa, G. and Mohammad, J.S.: Trajectory optimization of a mobile robot with flexible links using the pontryagin's method. In: *International Journal of Mecahnical Engineering and Robotics Research*, vol. 3, pp. 149–164. 2014.

[106] Nikoobin, A., Vezvari, M. and Ahmadieh, M.: Optimal balancing of planar cable robot in point to point motion using the indirect approach. In: *2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM)*, pp. 499–504. 2015.

[107] Mathworks: MATLAB software description.
Available at: `http://www.mathworks.com`

[108] Kaya, C.Y.: Leapfrog code email sent to B.T Matebese by C.Y Kaya. 2013.

[109] Nazemizadeh, M., Rahimi, H. and Khoiy, K.A.: Trajectory planning of mobile robots using indirect solution of optimal control method in generalized point-to-point task. *Frontier of Mechanical Engineering*, vol. 7, no. 1, pp. 23–28, 2012.

[110] Korayem, M., Rahimi, H. and Nikoobin, A.: Path planning of mobile elastic robotic arms by indirect approach of optimal control. *International Journal of Advanced Robotic Systems*, vol. 8, no. 1, pp. 10–20, 2001.

[111] Heidari, H., Haghpanahi, M. and Korayem, M.H.: Payload maximization for mobile flexible manipulators in an environment with obstacle. *Journal of Theoretical and Applied Mechanics*, vol. 53, no. 4, pp. 911–923, 2017.

[112] Sgorbissa, A.: An integrated approach to path following, obstacle avoidance,and motion planning in 2D and 3D workspaces: a case study with mobile, flying, and underwater robots. Tech. Rep., University of Genova, 2015.

[113] Mohri, A., Furuno, S., Iwamura, M. and Yamamoto, M.: Sub-optimal trajectory planning of mobile manipulator. In: *Proceedings of the IEEE International Conference on Robotics and Automation, (ICRA)*, pp. 1271–1276. 2001.

[114] Papadopoulos, E. and Poulakakis, J.: Planning and model-based control for mobile manipulators. In: *2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 1810–1815. 2000.

[115] Wu, Y. and Hu, Y.: Kinematics dynamics and motion planning of wheeled mobile manipulators. In: *Proceeding of International Conference on CSIMTA'04*, pp. 221–226. 2004.

[116] Mohri, A., Furuno, S. and Yamamoto, M.: Trajectory planning of mobile manipulator with end-effector's specified path. In: *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 4, pp. 2264–2269. 2001.

[117] Mohri, A., Yang, X.D. and Yamamoto, M.: Collision free trajectory planning for manipulator using potential function. In: *Proceedings of the 1995 International Conference on Robotics and Automation, Nagoya, Aichi, Japan, May 21-27, 1995*, pp. 3069–3074. 1995.

[118] Kuffner Jr., J.J. and Lavalle, S.M.: RRT-connect: An efficient approach to single-query path planning. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 995–1001. 2000.

[119] Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P.: *Numerical Recipes in FORTRAN (2nd Ed.): The Art of Scientific Computing.* Cambridge University Press, New York, NY, USA, 1992. ISBN 0-521-43064-X.