PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE

SCHOOL OF ENGINEERING

A COMBINED RRT*-OPTIMAL CONTROL APPROACH FOR KINODYNAMIC MOTION PLANNING FOR MOBILE ROBOTS

NICOLÁS AXEL BUSCH HOPFENBLATT

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Advisor:
MIGUEL TORRES TORRITI

Santiago de Chile, March 2016

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

A COMBINED RRT*-OPTIMAL CONTROL APPROACH FOR KINODYNAMIC MOTION PLANNING FOR MOBILE ROBOTS

NICOLÁS AXEL BUSCH HOPFENBLATT

Members of the Committee:
MIGUEL TORRES TORRITI
JORGE BAIER ARANDA
FERNANDO AUAT CHEEIN
GLORIA ARANCIBIA HERNÁNDEZ

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Santiago de Chile, March 2016

No one wants to learn by mistakes, but we cannot learn enough from successes to go beyond the state of the art ... Such is the nature not only of science and engineering, but of all human endeavors.

HENRY PETROSKI

ACKNOWLEDGEMENTS

This work was supported by the National Commission for Science and Technology Research of Chile (CONICYT) under grants Fondecyt 1110343, Fondecyt 1140575, Fondequip 120141, Basal Project FB0008, and CONICYT-PCHA/MagisterNacional/2014-22141669.

Special thanks to my advisor Miguel Torres, for all the time, support and guidance without which this work could not have been possible.

I would like to express my gratitude to the members of the review committee for their valuable remarks.

Thanks to my friends in the Robotics and Automaton Laboratory (RAL) for providing a friendly environment, and specially to Jorge Reyes for the help on the experimental validation of my work.

Finally, but most importantly, to my family, for their unconditional love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

An important challenge in underground mining is that of motion planning for excavators and hauling trucks because of the tight spaces available to maneuver and fuel consumption. During the last decades several path planning strategies for mobile robots have been developed. However, most of the existing methods only find a path that satisfies certain optimality criteria, but do not consider the feasibility of the path accordingly with the robot's motion dynamics. Thus in this work two motion planning strategies are proposed that combine the RRT* path planning algorithm with the piecewise solution of an optimal control problem over a finite number subintervals of a simpler polygonal trajectory derived from the path found by RRT*. The proposed approaches take into account the motion model of the robot, hence yield kinodynamically feasible trajectories. One strategy solves a local two-point boundary value optimal control problem at the corners of the polygonal path, while the second solves the two-point boundary value optimal control problem taking consecutive points of the polygonal trajectory. Geometrically, the first approach connects straight lines with curves at the turning points of the path, and the second approach yields smooth curves. In both cases the trajectories can be tracked by the motion controller with negligible errors compared to the trajectories that follow the RRT* path. However, an important difference between the proposed methods lies in the fact that the second one is able to yield trajectories that can be followed with an almost constant longitudinal velocity and gradual changes in heading turning rates. The proposed approaches were evaluated in simulations and tested in field experiments using a robotic excavator implemented with a Caterpillar CAT 262C2 compact loader. The results show that proposed approaches are computationally feasible on current standard computers and reduce the combined tracking error and controller effort criteria by $30-70\,\%$, depending on the complexity of the path.

Keywords: motion planning, path planning, kinodynamic model, RRT*, two-point BVP, path tracking controller.

RESUMEN

Uno de los desafíos más importantes en la minería subterránea es la planificación de trayectorias para excavadoras y camiones, debido principalmente al escaso espacio disponible para maniobrar y al consumo de combustible. Múltiples métodos para planificación de trayectorias para robots móviles han sido propuestos en las últimas décadas. Gran parte de estos métodos calculan trayectorias que satisfacen un criterio de optimización, pero no consideran la factibilidad de la trayectoria en términos de la dinámica de movimiento del robot. En este trabajo se presentan dos métodos de planificación de trayectorias que combinan el algoritmo de planificación RRT* con la solución de un problema de control óptimo sobre un número finito de subintervalos de una trayectoria poligonal, la cual se deriva del camino encontrado con RRT*. Los algoritmos propuestos consideran el modelo de movimiento del robot en la etapa de planificación, por lo que las trayectorias generadas son dinámicamente compatibles con el robot. La primera estrategia resuelve un problema de control óptimo en la cercanía de los vértices de la trayectoria poligonal, mientras que el segundo método resuelve el problema de control óptimo conectando directamente los vértices. Geométricamente, el primer enfoque conecta líneas con curvas en los vértices, mientras que el segundo genera curvas suaves. En ambos casos las trayectorias pueden ser seguidas con un controlador de movimiento con errores insignificantes en comparación con las trayectorias generadas con RRT*. No obstante, una diferencia importante entre los métodos propuestos es que el segundo método es capaz de generar trayectorias que pueden ser seguidas con una velocidad longitudinal constante y cambios graduales en la velocidad angular. Los algoritmos propuestos se evaluaron en simulaciones y experimentos usando un cargador frontal Caterpillar CAT 262C. Los resultados muestran una disminución del error de seguimiento y el esfuerzo del controlador de un $30-70\,\%$, dependiendo del escenario.

Palabras Claves: planificación de trayectorias, modelo dinámico, RRT*, BVP, seguidor de trayectorias.

1. INTRODUCTION

1.1. Motivation

The use of autonomous industrial mobile robots in outdoor environments has increased greatly in the last decade, mainly motivated by their effectiveness, commercial viability, and above all, safety (Seward, Pace, & Agate, 2007). Different industries benefit from the use of mobile robots, but perhaps one of the most emblematic cases is the mining industry. Mining is characterized for its hazardous environments, in which mobile robots not only provide an economic advantage, but most importantly, guarantee high safety standards. Currently there exist mines that use driverless trucks, and it is foreseeable that many more will be developed in the next years, especially for underground mining as minerals on the surface are being exhausted and deeper ores cannot be economically extracted by open pit mining.

One of the fundamental challenges an autonomous mobile robot must face is the motion planning problem, i.e. how to find the best trajectory through the workspace while avoiding obstacles and reducing the driving time, distance traveled, or fuel used. Path planning is not a novel concept in robotics, but it is to be noted that most of the existing path planners do not consider the motion constraints imposed by the robot's particular kinematics and dynamics. On the other hand, the few existing approaches that consider the kinodynamic motion constraints do not validate their results with field experiments involving actual industrial machinery like in this work.

1.2. Problem Definition

Following the notation convention used by (Karaman & Frazzoli, 2010b; LaValle, 2006), let $X \subset \mathbb{R}^n$ and $U \subset \mathbb{R}^m$ be the set of *state space* and *controls*, respectively. The system is described by the following equation that determines its state transition:

$$\dot{x}(t) = f(x(t), u(t)), \qquad x(0) = x_{\text{init}} \tag{1.1}$$

where $x(t) \in X$, $u(t) \in U$, for all t, $x_{\text{init}} \in X$, and f is a continuously differentiable function of its variables. Additionally let $\mathcal{X}$ denote the set of all essentially bounded functions $x : [0, T] \to X$ as *state space trajectories*, and $\mathcal{U}$ denote the set of all essentially bounded functions $u : [0, T] \to U$ as *control trajectories*, with $T \in \mathbb{R}_{>0}$ the final time. Following this definition it is clear that a state trajectory $x \in \mathcal{X}$, while a state $x(t) \in X$, for a particular time t. The same logic applies to control trajectories u and controls $u(t)$.

The complete state space X can be divided into two open sub sets, X_{free} and X_{obs}, called the *free space* and the *obstacle region*, respectively, where $X_{\text{free}} = X \setminus X_{\text{obs}}$. Similarly the *goal region* is an open subset of X, with $X_{\text{goal}} \subset X_{\text{free}}$.

Given the state space X, the obstacle region X_{obs}, an initial state $x_{\text{init}} \in X_{\text{free}}$, a goal region $X_{\text{goal}} \subset X_{\text{free}}$, a state transition function f that describes the kinodynamical constraints of the robot, and a cost functional to be minimized $J : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$, the optimal kinodynamic motion planning problem is to find a control trajectory $u \in \mathcal{U}$ with domain $[0, T]$ for some $T \in \mathbb{R}_{>0}$ such that the generated trajectory $x \in \mathcal{X}$,

(i) avoids all the obstacles, i.e. $x(t) \notin X_{\text{obs}} \;\; \forall t \in [0, T]$

(ii) reaches the goal region, i.e. $x(T) \in X_{\text{goal}}$

(iii) satisfies the kinodynamical model of the robot: $\Sigma \overset{def}{=} \{\dot{x}(t) = f(x(t), u(t)) \wedge u \in \mathcal{U}, \;\; \forall t \in [0, T]\}$,

(iv) minimizes the cost functional $J(x, u) = \displaystyle\int_0^T g(x(t), u(t))\, dt$

The motion planning problem is represented in Figure 1.1, which shows the free space X_{free} in white, the obstacle region X_{obs} in gray, and the goal region X_{goal} as a black disc. The robot can execute different feasible trajectories that comply with its kinodynamic model, however only one of the trajectories is optimal and minimizes the cost functional.

It is to be noted that this problem is more complex than the path planning problem, in which only a geometrical path that connects start and end points while avoiding obstacles is sought without consideration of the control commands nor constraints imposed by the kinematics and dynamics of the robot. On the other hand, the motion planning problem involves determining which motions and associated control signals are necessary to generate

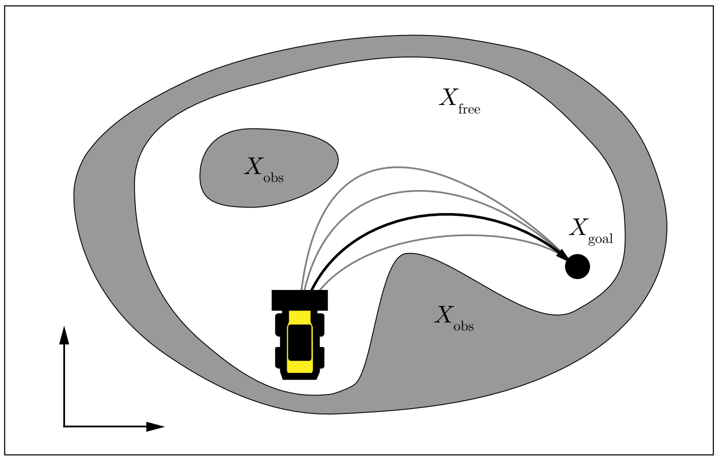


FIGURE 1.1. Motion Planning Problem.

a trajectory that reaches the goal region without colliding into obstacles (LaValle, 2006). If additionally the motion model and constraints given by Equation (1.1) are simultaneously kinematic and dynamic, then the problem is referred to as a kinodynamic motion planning problem (Donald, Xavier, Canny, & Reif, 1993).

1.3. Objectives

The main objective is to design a motion planner capable of generating kinodynamic trajectories that can be followed by a real mobile robot. The motion planner must optimize the trajectory based on a specific objective function, e.g., minimizing time, distance traveled, or fuel used, while avoiding the obstacles of the environment and complying with the kinodynamic constraints that bound the movement of the robot.

1.4. Hypothesis

A motion planner that combine the sampling-based RRT* path planning algorithm with the piecewise solution of an optimal control problem over a finite number subintervals of a simpler polygonal trajectory, derived from the path found by RRT*, can improve the quality of the trajectory, compared to existing approaches. The quality of the planned trajectory is defined as the ability of the robot to properly follow it, which is highly dependent on the kinodynamic characteristics of the robot, and it is here measured as a combination of trajectory tracking error and controller effort.

1.5. Existing Approaches

Although the focus of this thesis is in motion planning, path planning will be discussed first because it is an essential part of many motion planning strategies. Path planning is one of the fundamental problem in robotics (Latombe, 1999). As a result, in the last decades numerous approaches have been proposed. Path planning methods can be divided into two groups: exact methods and sampling-based methods. Exact methods consider mathematical approaches such as Potential Fields (Barraquand, Langlois, & Latombe, 1991; Khatib, 1985), graph traversal and grid pathfinding strategies such as A* (Hart, Nilsson, & Raphael, 1968) and D* (Stentz, 1994), and geometric approaches such as Voronoi Diagrams (Canny, 1985) and Visibility Graphs (Lozano-Pérez & Wesley, 1979). On the other hand, the classic sampling-based planning algorithms include the Probabilistic Roadmap Method (PRM) (Kavraki, Svestka, Latombe, & Overmars, 1996) and Rapidly-exploring Random Trees (RRT) (LaValle & Kuffner, 1999).

Potential Fields (Khatib, 1985) represent the world as a field of forces, the goal state generates attractive forces while obstacles generate repulsive forces, hence the robot moves as a result of these forces. Although this approach is simple and elegant, the robot may get trapped when the forces cancel each other out. Grid-based algorithms decompose the workspace into small cells, where each cell has a traversing cost associated. A* (Hart et al., 1968) uses an heuristic approach to find the minimum length collision-free path in a connectivity graph, whereas D* (Stentz, 1994) is tailored to dynamic graphs. Graph search methods are efficient for low dimensional scenarios, since the grid representation is not appropriate for large environments. The Voronoi Diagram (Canny, 1985) is formed by lines that maximize the clearance to obstacles. The path found is equidistant to the nearest obstacles, allowing the robot to follow a safe path through the obstacle free space. Another geometric approach is the Visibility Graph (Lozano-Pérez & Wesley, 1979), which attempts to capture the connectivity of the configuration space by joining the vertices of the obstacles with collision free lines. Once the roadmap is built, the shortest path is calculated using simple graph search techniques. The complexity grows with the number of vertices, hence this approach is best suited for environments with a low number of obstacles (Kunchev, Jain, Ivancevic, & Finn, 2006).

Sampling based path planning methods attempt to capture the connectivity of the configuration space by sampling it (Elbanhawi & Simic, 2014). These approaches provide fast solutions for difficult problems, e.g., high dimensional or with kinodynamic constraints. However the solutions found are generally suboptimal, and if they offer some degree of optimality, then a very large or infinite amount of time is needed to reach the optimal solution. Two of the most well known sampling-based algorithms are Probabilistic Roadmap Method (PRM) (Kavraki et al., 1996) and Rapidly-exploring Random Trees (RRT) (LaValle & Kuffner, 1999). PRM first samples the configuration space to generate a map of connections between valid states. Then the start and end points are connected to the respective closest available states in the map. The path from the start to the end that has been created is then followed through the map. The same map can be used to solve different instances of the problem in the same environment, and for this reason it is referred as a multi-query planner. RRT also samples the configuration space, but instead of generating a map, it grows a search tree starting from the initial state that densely covers the configuration space. As opposed to PRM, RRT is a single-query planner, i.e. it will find a path from an initial state to a final state faster than PRM, but the RRT tree is not as adaptable to other problems in the same environment as the PRM map is (Elbanhawi & Simic, 2014).

The main drawback of RRT and PRM is the quality of the generated solution. For RRT, once the goal region is reached additional samples will not improve the quality of the path, and in fact, it has been shown that the RRT algorithm converges to a non-optimal solution with probability one (Karaman & Frazzoli, 2010a). The work of Karaman and Frazzoli (2011) introduced a new family of planners which guaranteed asymptotic optimality, i.e. the probability of converging to the optimal solution approaches almost surely to one as the number of iterations goes to infinity. RRT* and PRM* are the asymptotically optimal counterparts of RRT and PRM, respectively. One of the main drawbacks of RRT* is its slow rate of convergence to the optimal path. This rate is characterized by being fast at the beginning, but the improvement made on each iteration becomes smaller as the number of iterations grows, needing an infinite amount of time to converge to an optimal solution.

Post processing techniques allow a sampling-based algorithms to converge to a smooth path in a finite amount of time, however the optimality of the solution cannot be guaranteed. Path smoothing consists on using a curve to interpolate or fit a set of waypoints. Different methods have been applied to path smoothing, such as cubic polynomials (Thrun et al., 2006), Bezier curves (Jolly, Sreerama Kumar, & Vijayakumar, 2009; Yang & Sukkarieh, 2010) and B-splines (Kanayama & Hartman, 1997; Maekawa, Noda, Tamura, Ozaki, & Machida, 2010; Shan, Dai, Song, & Sun, 2009; Yang et al., 2014). Smoothed paths comply with simple kinodynamic models, as the generated paths have continuous first and second order derivatives. This approach, however, does not find the optimal path between way-points.

In practice it is necessary that the planner takes into account the kinematic and dynamic constraints of the robot, but most of the basic path planners do not consider them, which may result in trajectories that cannot be properly followed by the robot (De Oliveira Vaz, Inoue, & Grassi, 2010). Besides, as stated in (LaValle, 2006), the only known methods for exact planning under differential constraints in the presence of obstacles are for the double integrator system in $\mathbb{R}$ and $\mathbb{R}^2$. For this reason, the majority of solution techniques are sampling-based. However, embedding a kinodynamic model in the RRT* algorithm is not trivial, because numerous two-point Boundary Value Problems (TPBVP) must be solved for every iteration. Therefore, most cases found in the literature use simple kinodynamic models, or no models at all. Simple models have an exact solution to the TPBVP, e.g., the Dubins' Vehicle (Dubins, 1957), and thus the amount of time required to solve the planning problem is manageable. For more complex kinodynamic models there is no exact solution, so numerical approaches must be used to solve the TPBVP.

In the last decade there have been important advances in numerical optimal control and optimization to solve TPBVP, which has open the possibility of embedding numerical solvers into asymptotically optimal motion planning algorithms (Xie, van den Berg, Patil, & Abbeel, 2015). The main drawback of this approach is the time needed to solve a large number of TPBVP. For every iteration the computation time is small, but the total computation time becomes unmanageable as the number of iterations grows. Other approaches, such as Bath Informed Trees (BIT*) (Gammell, Srinivasa, & Barfoot, 2014a), Fast Marching Trees (FMT*) (Janson, Schmerling, Clark, & Pavone, 2015), Kinodynamic RRT* with linear differential constraints (Webb & van den Berg, 2012), Stable Sparse RRT* (Li, Littlefield, & Bekris, 2014), and Sucessive Approximation RRT* (SA-RRT*) (Ha, Lee, & Choi, 2013) have tried to address the problem of using any kinodynamic model with a RRT*-based approach, with different degrees of success. These algorithms usually use variations of the original RRT*.

Additionally, most of the published motion planning algorithms are only tested in simulated environments and not experimentally validated. Some exceptions are found in (Ardiyanto & Miura, 2011; De Oliveira Vaz et al., 2010; Gammell et al., 2014a; Stenning, McManus, & Barfoot, 2013), which conduct experimental validation with small to medium size robots. Although this provides an experimental verification of the theoretical concepts, it is desirable from the application perspective to validate the planning algorithms with full size industrial machinery, as in (Karaman, Walter, Perez, Frazzoli, & Teller, 2011).

1.6. Contributions

The contributions of this work can be summarized as:

(i) Two new algorithms, LC-RRT* and C-RRT*, that combine in a novel way the best characteristics of sampling-based methods, i.e. the fast rate of convergence to a quasi-optimal solution, with the benefits of a TPBVP, i.e. the compatibility between the trajectory and the kinodyamics of the robot.

(ii) The experimental validation of the proposed algorithms with a full-size industrial machinery, hence demonstrating the applicability of the algorithms with real robots.

(iii) A new communication protocol, the Robot Communication Protocol (RCP). RCP is a fast, simple, and minimal binary protocol for communicating between computers and robots.

1.7. Thesis Outline

This thesis is organized as follows. Chapter 2 lays the ground in terms of notation, problem formulation, and basic notions necessary for understanding the proposed algorithms. In Chapter 3 the new proposed algorithms are presented. Simulations are provided in Chapter 4, where the new algorithms are compared against traditional approaches in different scenarios, then advantages/disadvantages are discussed and analyzed. Details of implementation and field-testing are provided in Chapter 5. Chapter 6 presents the results obtained from experimental validation with a large-size industrial machinery. Finally, Chapter 7 contains the lessons learned and conclusions of this work.

2. PRELIMINARY NOTIONS

The proposed method is based on the well known RRT algorithm and its globally asymptotically optimal variant, the RRT* algorithm. Since these algorithms are an essential part of this work, this section covers a basic introduction to the sampling-based RRT and RRT* planning algorithms.

Sampling-based planning algorithms have been widely used in the last years to solve difficult problems, with certain geometric o kinodynamic constraints, that would be practically impossible to solve using other techniques. These algorithms sample the configuration space (C-space or $\mathcal{C}$) to capture its connectivity, in order to find a path that can safely cross it.

The notion of *completeness* for sampling-based planning algorithms is essential and is important to introduce. A deterministic planning algorithm is considered *complete* if in a finite amount of time it is capable of returning a solution. If no solution exists it must report this situation. Unfortunately this notion of completeness cannot be achieved by sampling-based planning algorithms, and a weaker notion of completeness must be introduced. A sampling-based algorithm will be called *probabilistically complete* if the probability of finding a solution converges to one, as the number of sampling points increases to infinity. If there is no solution the algorithm may run forever (LaValle, 2006).

Similarly, the notion of optimality does not exist for sampling-based methods. Instead the notion of *asymptotic optimality* must be introduced, i.e. the probability of converging to the optimal solution approaches almost surely to one as the number of iterations goes to infinity (Karaman & Frazzoli, 2011).

2.1. Rapidly-exploring Random Tree (RRT)

Rapidly-Exploring Random Trees, or RRT, is a sampling-based path planning algorithm, and it was first proposed by LaValle and Kuffner (1999). This algorithm was developed to cover the need for a path planner able to consider complicated kinodynamical constraints and generate feasible trajectories. The basic idea is to incrementally construct a search tree that densely covers the configuration space. With uniform sampling, the probability of expanding towards a given region is proportional to the size of its Voronoi Region, which assures the tree will grow towards large unsearched areas. The algorithm is also capable of considering the kinodynamic constraints of the robot, while maintaining a low computational overhead. These characteristics have made of RRT one the most used sampling-based planning algorithms in the last decade (Karaman & Frazzoli, 2013).

The total path x is subdivided in states q_i. These states correspond to the path evaluated at discrete time steps T_i, such that $q_i = x(T_i)$. Each of these states will be called a vertex, or node, of the tree. The algorithm builds a tree $G = (V, E)$ and stores every vertex in V and every edge between vertices in E.

As seen in Algorithm 1, the procedure starts by initializing the tree G with values for V and E. In every iteration it selects a random point with `Sample`, which returns independent and identically distributed samples from X_{free}. To achieve a faster rate of convergence the algorithm is biased toward the destination point. This is done by making the `Sample` procedure return the final state with probability p, and other random state with probability $(1 - p)$.

```
Algorithm 1: RRT
    input: $q_{\text{init}}, N$
 1  $V \leftarrow \{q_{\text{init}}\}; E \leftarrow \emptyset; i \leftarrow 0;$
 2  while $i \leq N$ do
 3      $q_{\text{rand}} \leftarrow$ Sample($i$); $i \leftarrow i + 1;$
 4      $q_{\text{nearest}} \leftarrow$ Nearest($V, E, q_{\text{rand}}$);
 5      $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow$ Steer($q_{\text{nearest}}, q_{\text{rand}}$);
 6      $q_{\text{new}} \leftarrow x_{\text{new}}(T_{\text{new}});$
 7      if ObstacleFree($x_{\text{new}}$) then
 8          $V \leftarrow V \cup \{q_{\text{new}}\};$
 9          $E \leftarrow E \cup \{(q_{\text{nearest}}, q_{\text{new}})\};$
10  return $G = (V, E)$
```

The `Nearest` method finds the nearest neighbor of a given state. Let the state be q_{rand} and the nearest neighbor be q_{nearest}. In this case the `Nearest`(V, E, q_{rand}) method solves the problem $\arg\min_{q_{\text{nearest}}\in V} \mathtt{d}(q_{\text{nearest}}, q_{\text{rand}})$, where $\mathtt{d}(q_1, q_2)$ returns the euclidian distance between the two points.

The next step is to go from the nearest vertex, q_{nearest}, to the given state q_{rand}. This is done by the $\mathtt{Steer}(q_{\text{nearest}}, q_{\text{rand}})$ method. It calculates the trajectory x_{new}, the control signals u_{new} and the time T_{new}, such that $\arg\min_{u_{\text{new}}\in U} \mathtt{d}(x_{\text{new}}(T_{\text{new}}), q_{\text{rand}})$, while complying with the kinodynamic constraints of the robot. This means $x_{\text{new}}(T_{\text{new}})$ is the closest state to q_{rand} that can be reached from q_{nearest} and satisfies the kinodynamic constraints.

The kinodynamic model used for the `Steer` method is usually represented by a state transition equation of the form $\dot{x}(t) = f(x(t), u(t))$, equivalent to equation (1.1). Solving this problem is equivalent to solve a two-point Boundary Value Problem, which is not trivial, as it will be explained in Section 2.5. Therefore, many of the RRT kinodynamic implementations use a shooting method with random control inputs instead of finding the control input, u_{new}, that minimizes the distance between q_{new} and q_{rand}.

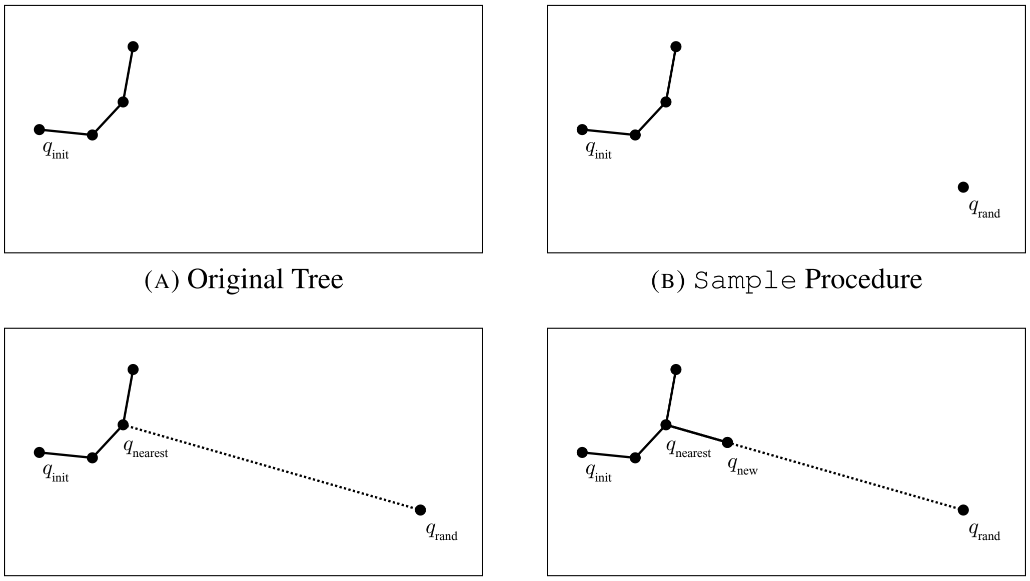
The `ObstacleFree` method is used to check if the trajectory x_{new} avoids all the obstacles, i.e. $x_{\text{new}}(t) \in X_{\text{free}}$ for all $t \in [0, T_{\text{new}}]$. If x_{new} is collision free, then the node q_{new} is added to V and the edge between q_{nearest} and q_{new} is added to E. The main procedure (Lines 2-9) is repeated for a pre-specified number of iterations.

The steps for `Sample`, `Nearest`, and `Steer` can be seen in Figure 2.1. For the sake of simplicity, a holonomic robot model has been used for the `Steer` procedure, although the method can use other kinodynamic models as well.

The main drawback of the RRT algorithm is the quality of the generated solution. Once the goal region is reached, additional samples will not improve the quality of the path, and in fact, it has been showed that the RRT algorithm converges to a non-optimal solution with probability one (Karaman & Frazzoli, 2010a).

2.2. Asymptotically Optimal RRT (RRT*)

This section describes the RRT* path planning algorithm. RRT* is an improved version of RRT, and it was introduced in (Karaman & Frazzoli, 2011). The main advantage



(A) Original Tree (B) `Sample` Procedure

(C) `Nearest` Procedure (D) `Steer` Procedure

FIGURE 2.1. RRT Algorithm.

of RRT* over RRT is its globally asymptotically optimal quality, i.e. the probability of converging to the optimal path approaches almost surely to one as the number of iterations goes to infinity (Karaman & Frazzoli, 2010a). RRT* is capable of finding asymptotically optimal solutions by combining the basic ideas of RRT, such as using a random tree to explore rapidly the state space, with an optimal tree rewiring procedure.

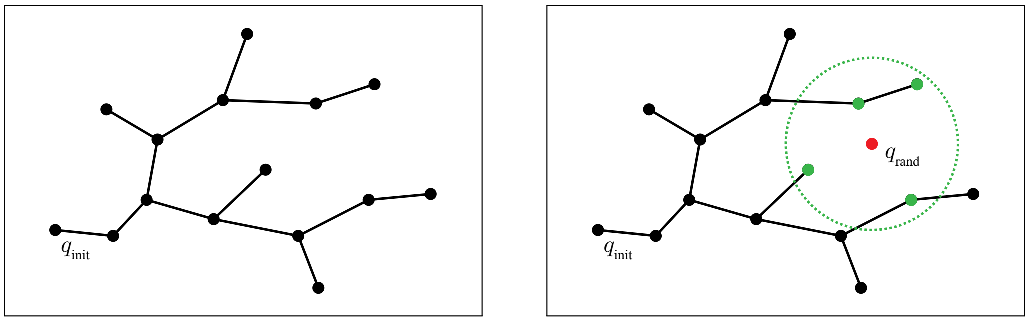
The basic structure is the same as in RRT, and the methods `Sample`, `Nearest`, `Steer`, and `ObstacleFree` are identical to the ones explained in Section 2.1. In order to achieve the global asymptotic optimality of RRT*, the `Steer` method must return the optimal sub-trajectory between the two given points. The additional methods needed for RRT* are explained next, and the complete procedure is summarized in Algorithm 2.

The `NearVertices`(V, E, q_{new}) method (Line 10) finds all the vertices in V that are in the neighborhood of q_{new}, the size of the neighborhood is parametrized by n. It returns Q_{nearby}, which corresponds to any $q_{\text{near}} \in V$ that is inside a ball of radius l_n centered at q_{new}. The value of l_n is chosen in such a way that the volume of the ball is $\gamma \log(n)/n$, where γ is a constant.

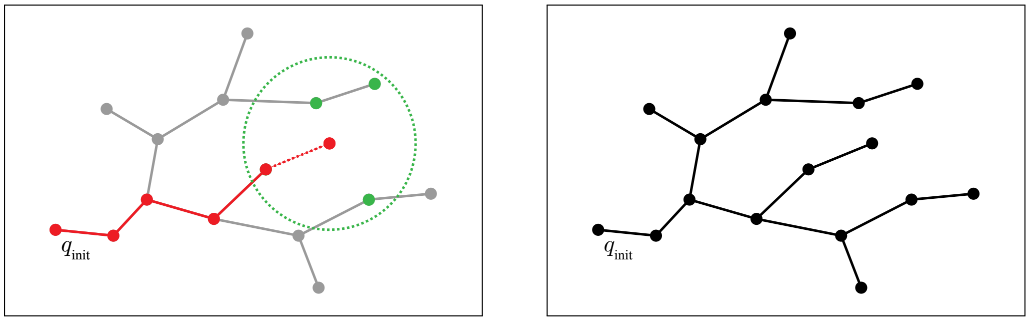
Algorithm 2: RRT*

```
input: $q_{\text{init}}, N$
1  $V \leftarrow \{q_{\text{init}}\}; E \leftarrow \emptyset; i \leftarrow 0;$
2  while $i \leq N$ do
3      $q_{\text{rand}} \leftarrow \texttt{Sample}(i); i \leftarrow i + 1;$
4      $q_{\text{nearest}} \leftarrow \texttt{Nearest}(V, E, q_{\text{rand}});$
5      $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(q_{\text{nearest}}, q_{\text{rand}});$
6      $q_{\text{new}} \leftarrow x_{\text{new}}(T_{\text{new}});$
7      if $\texttt{ObstacleFree}(x_{\text{new}})$ then
8          $V \leftarrow V \cup \{q_{\text{new}}\};$
9          $q_{\text{min}} \leftarrow q_{\text{nearest}}; c_{\text{min}} \leftarrow \texttt{Cost}(q_{\text{new}});$
10         $Q_{\text{nearby}} \leftarrow \texttt{NearVertices}(V, E, q_{\text{new}});$
11         foreach $q_{\text{near}} \in Q_{\text{nearby}}$ do
12             $(x_{\text{near}}, u_{\text{near}}, T_{\text{near}}) \leftarrow \texttt{Steer}(q_{\text{near}}, q_{\text{new}});$
13             if $\texttt{ObstacleFree}(x_{\text{near}})$ and $x_{\text{near}}(T_{\text{near}}) = q_{\text{new}}$ then
14                 if $\texttt{Cost}(q_{\text{near}}) + J(x_{\text{near}}) < c_{\text{min}}$ then
15                     $c_{\text{min}} \leftarrow \texttt{Cost}(q_{\text{near}}) + J(x_{\text{near}});$
16                     $q_{\text{min}} \leftarrow q_{\text{near}};$
17         $E \leftarrow E \cup \{(q_{\text{min}}, q_{\text{new}})\};$
18         foreach $q_{\text{near}} \in Q_{\text{nearby}} \setminus \{q_{\text{min}}\}$ do
19             $(x_{\text{near}}, u_{\text{near}}, T_{\text{near}}) \leftarrow \texttt{Steer}(q_{\text{new}}, q_{\text{near}});$
20             if $\texttt{ObstacleFree}(x_{\text{near}})$ and $x_{\text{near}}(T_{\text{near}}) = q_{\text{near}}$ then
21                 if $\texttt{Cost}(q_{\text{near}}) > \texttt{Cost}(q_{\text{new}}) + J(x_{\text{near}})$ then
22                     $q_{\text{parent}} \leftarrow \texttt{Parent}(q_{\text{near}});$
23                     $E \leftarrow E \setminus \{(q_{\text{parent}}, q_{\text{near}})\};$
24                     $E \leftarrow E \cup \{(q_{\text{new}}, q_{\text{near}})\};$
25 return $G = (V, E)$
```

The next step is to choose the parent node of q_{new} (Lines 11-16). RRT does this by selecting the closest node, but this procedure leads towards non-optimal solutions. Instead, RRT* searches in all the nearby vertices in Q_{nearby} for the node with the smallest cumulative cost. For each $q_{\text{near}} \in Q_{\text{nearby}}$ the method steers towards q_{new}. If the resulting trajectory is obstacle free and reaches the state q_{new}, then the total cost from the root vertex to the final state is calculated, i.e. $\texttt{Cost}(q_{\text{near}}) + J(x_{\text{near}})$. $\texttt{Cost}(q)$ is defined as the total cost from the root vertex to the state q and $J(x_i)$ as the cost along the sub-trajectory x_i. If this new cost



(A) Original Tree (B) Nearby nodes



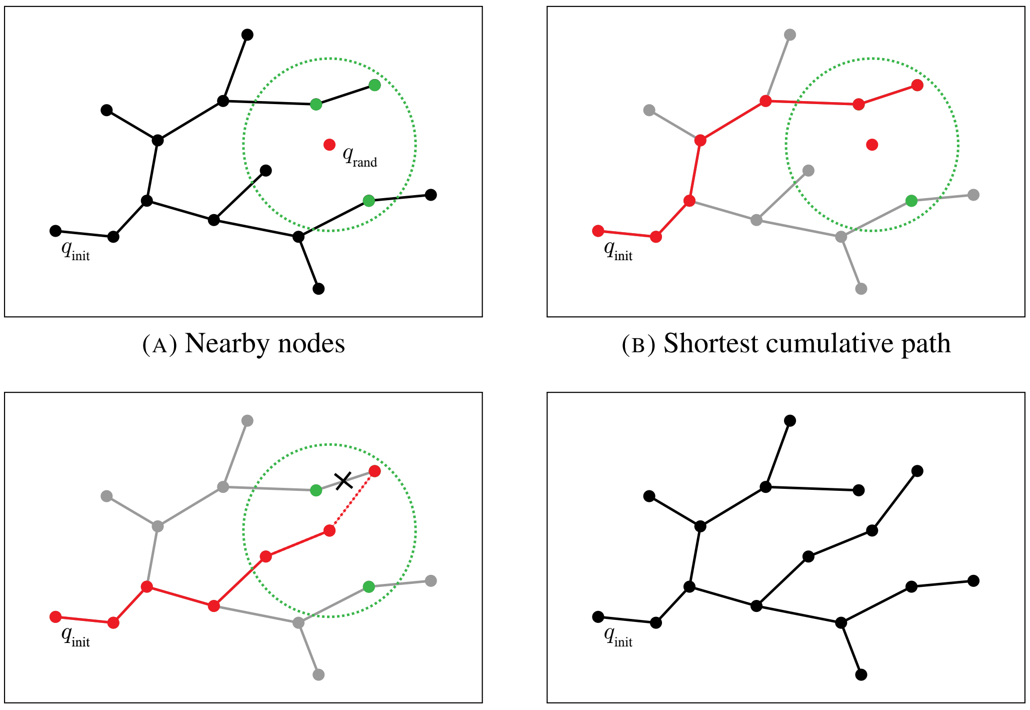
(C) Shortest cumulative path (D) Add node with parent

FIGURE 2.2. RRT* Algorithm - Choose Parent.

is smaller than the previously stored one, then the cost and the vertex are updated. When this procedure is over, the parent of q_{near} is set as q_{min} and the tree is updated. Figure 2.2 depicts the steps needed to choose the parent node of q_{new}.

The final part of the algorithm corresponds to the rewire procedure (Lines 18-24). The procedure optimizes the path by rewiring vertices in the vicinity of the new vertex. This is done as follows: the algorithm steers from q_{new} to each vertex $q_{\text{near}} \in Q_{\text{nearby}} \setminus q_{\text{min}}$. If the trajectory x_{near} is obstacle free and $x_{\text{near}}(T_{\text{near}}) = q_{\text{near}}$, it calculates the total cumulative cost from the root vertex to the near vertex using a new edge, i.e. $\texttt{Cost}(q_{\text{new}}) + J(x_{\text{near}})$. If this cost is smaller than the total cumulative cost using the old edges, i.e. $\texttt{Cost}(q_{\text{near}})$, then the parent of q_{near} is replaced by q_{new}. All the steps of the rewire procedure are presented in Figure 2.3.

The RRT* algorithm is globally asymptotically optimal as proven in (Karaman & Frazzoli, 2010a), as long as the local solution found by the `Steer` procedure is optimal. An



(A) Nearby nodes (B) Shortest cumulative path

(C) New shortest cumulative path (D) Rewire parent node

FIGURE 2.3. RRT* Algorithm - Rewire.

optimal solution only will be found in an infinity time, though the algorithm will be able to find a near-optimal trajectories in reasonable times. Because of its global optimality, the RRT* algorithm is very useful for practical applications. One of the main advantages of this method over other path planner is the ability to generate a path that is kinodynamically compatible with the robot, i.e. the robot will be able to properly follow the trajectory. If a kinodynamic model is used, then RRT* can be used as a motion planner instead of a path planner, which means that the algorithm will not only return the path $x \in \mathcal{X}$, but also the control signals $u \in \mathcal{U}$ for the trajectory.

2.3. Line RRT*

Line RRT*, or L-RRT*, optimizes the path found by RRT* based on the well known *triangle inequality*, which basically states that a one-line path is a shorter than a two-line path.

Given a quasi-optimal trajectory x generated with the RRT* algorithm and the set of nodes $q_i \in V$, such that $q_i = x(T_i)$, the line optimization algorithm directly connects nodes visible to each other. As seen in Algorithm 3, the method starts at the final node, q_{final}, and iteratively moves towards the root of the tree, q_{init}, trying to make collision free connections with successive parents of each node (Lines 5-7). If the process fails, the last successful connection is established and the process starts again from the last successfully connected node (Lines 8-11). This procedure is repeated until the root node is reached, and then the optimized path L is returned.

Algorithm 3: Line Optimization

```
   input: $G = (V, E)$
1  $L \leftarrow \{q_{\text{final}}\}$;
2  $q \leftarrow q_{\text{final}}$;
3  $q_{\text{parent}} \leftarrow$ Parent($q$);
4  while $q \neq q_{\text{init}}$ do
5      if ObstacleFree($q_{\text{parent}}, q$) then
6          $q_{\text{last}} \leftarrow q_{\text{parent}}$;
7          $q_{\text{parent}} \leftarrow$ Parent($q_{\text{last}}$);
8      else
9          $L \leftarrow L \cup \{q_{\text{last}}\}$;
10         $q \leftarrow q_{\text{last}}$;
11         $q_{\text{parent}} \leftarrow$ Parent($q_{\text{last}}$);
12 return $L$
```

Figure 2.4 depicts a comparison between the trajectories generated with RRT* and L-RRT*. The Line Optimization procedure reduces the total length of the path by reducing the number of edges with a low computational overhead. All new edges between vertices are collision free, and the new path shares the principal properties of RRT*, i.e. probabilistic completeness and asymptotic optimality.

As can be seen, L-RRT* effectively reduces the cost of the path found by RRT* with a low additional computational overhead. The main drawback is that the path is not compatible with the kinodynamic constraints, at least for most non-holonomical kinodynamic models.

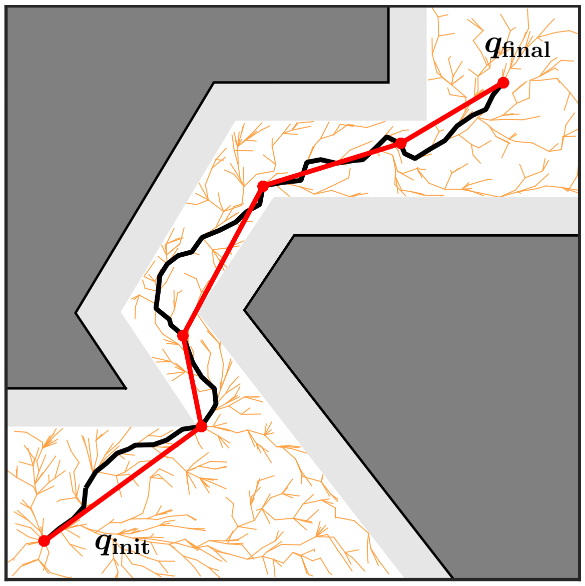


FIGURE 2.4. L-RRT* algorithm example.

L-RRT* is based on the work presented in (Islam, Nasir, Malik, Ayaz, & Hasan, 2012), which introduced Smart-RRT*. Smart-RRT* is an extension of RRT*, and it accelerates the convergence rate by two means: path optimization and intelligent sampling. First it reduces the path to a minimum number of states, based on the same principle used in L-RRT*, and then uses these states as biases for further sampling. As stated in (Gammell, Srinivasa, & Barfoot, 2014b), the biased sampling procedure violates the RRT* assumption of uniform sampling, thus Smart-RRT* loses the asymptotic optimality property of the original RRT* algorithm. For this reason, line RRT*, or L-RRT*, is introduced here as a simplified version of Smart-RRT* that only uses the path optimization procedure, hence preserving the probabilistic convergence properties of RRT*.

2.4. Kinematic Model

The kinodynamic model is considered as part of the motion planning problem in order to produce trajectories that satisfy the kinodynamic constraints. For simplicity and to reduce the computational burden, the model employed here considers the kinematic equations of a popular differential-drive robot given by Σ,

$$\Sigma: \quad \begin{bmatrix} \mathrm{x}_{k+1} \\ \mathrm{y}_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} \mathrm{x}_{k} \\ \mathrm{y}_{k} \\ \theta_{k} \end{bmatrix} + \begin{bmatrix} \lambda_{\mathrm{v}} \mathrm{v}_k \cos(\theta_k)\Delta_t \\ \lambda_{\mathrm{v}} \mathrm{v}_k \sin(\theta_k)\Delta_t \\ \lambda_w w_k \Delta_t \end{bmatrix} \tag{2.1}$$

where $[\mathrm{x}_k \ \mathrm{y}_k \ \theta_k]$ is the state of the mobile robot at time instant k consisting of the global position and orientation, v_k and w_k are the longitudinal and angular velocities, λ_{v} and λ_w are longitudinal and angular slippage factors, and Δ_t is the sampling time. For the kinematic model of the differential drive robot, the inputs are the left- and right-wheel velocities, which are related to the longitudinal and rotational velocity by $\mathrm{v}_k = \frac{\mathrm{v}_{\phi_r} + \mathrm{v}_{\phi_l}}{2}$ and $\omega_k = \frac{\mathrm{v}_{\phi_r} - \mathrm{v}_{\phi_l}}{W}$, where W is the width of the robot. This model is also a good approximation to kinematic model of skid-steer mobile bases. If the control inputs are directly the longitudinal and angular velocities v_k and w_k, then the equations correspond to those of the kinematic model for a unicycle.

2.5. Two-point Boundary Value Problem

Generally speaking, a boundary value problem (BVP) is a differential equation with a set of additional constraints, called the boundary conditions (LaValle, 2006). In the field of motion planning a TPBVP solves the optimal trajectory through a state space that connects a given initial state with a given final state while satisfying differential constraints. A motion planner also has to avoid the obstacles of the environment, but unfortunately TPBVPs are not designed to handle obstacles, and thus it is necessary to combine the BVP solver with a motion planning algorithm that avoids the obstacles of the environment.

Given the recent advances in numerical optimal control and nonlinear optimization, it is feasible to solve the TPBVPs in a systematic way for arbitrary kinodynamic systems, and integrate these methods with existing sampling-based optimal planning methods (Xie et al., 2015).

The optimal control problem can be stated as,

$$\min_{u,x,T} \quad J(x,u) \tag{2.2}$$

$$\dot{x}(t) = f(x(t), u(t)) \quad \forall t \in [0, T] \tag{2.3}$$

$$x(0) = x_{\text{init}} \tag{2.4}$$

$$x(T) = x_{\text{final}} \tag{2.5}$$

$$u(t) \in U \quad \forall t \in [0, T] \tag{2.6}$$

$$T > 0 \tag{2.7}$$

where the state transition function given by $f : X \times U \to X$ describes the kinodynamical constraints of the robot, $x_{\text{init}} \in X_{\text{free}}$ and $x_{\text{final}} \in X_{\text{free}}$ are the initial and final state, respectively, $T \in \mathbb{R}_{>0}$ is the final time, and $J : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the cost functional to minimize. The final time T is not known in advance so it is introduced as another variable in the system. In this stage no obstacles are considered, since TPBVPs are not well suited to handle obstacle regions, thus it is fundamental to check if the solution found is collision-free.

3. PROPOSED MOTION PLANNING STRATEGY

Asymptotically optimal sampling-based methods, such as RRT*, need an infinite amount of time to converge to an optimal solution. Therefore, the trajectories generated in finite time will never be smooth enough for a path tracking controller to follow without generating an oscillatory behavior in the control signals, as will be shown in Chapter 4. On the other hand, the simplified path obtained with the L-RRT* method is not kinodynamically compatible with most real robots because of the polygonal trajectory, that produces tracking errors that the path tracking controller cannot handle especially at tight polygonal turns.

These facts motivate the development of two algorithms that are built upon RRT* and L-RRT*, but include the solution of an optimal control two-point boundary value problem at specific turning points in the trajectory, thus reducing the tracking error and controller effort. The approach yields a trajectory that is kinodynamically compatible with the motion model, and thus its realization by the real robot is feasible.

3.1. Line-Corner RRT*

There exist kinodynamic models where the optimal trajectory is the combination of straight lines and curves. For example, the optimal trajectory between two states for the Dubins' Vehicle is the combination of straight lines and turns with constant curvature, as shown in (Dubins, 1957). For some types of models the optimal solution for sections of the trajectory are straight lines, e.g., a car-like robot going in straight line, an airplane flying, a submarine, but as opposed to the Dubins' Vehicle the turn section may have a non-constant curvature. In these scenarios the problem consists of identifying which sections of the optimal trajectory are straight lines and which are turns.

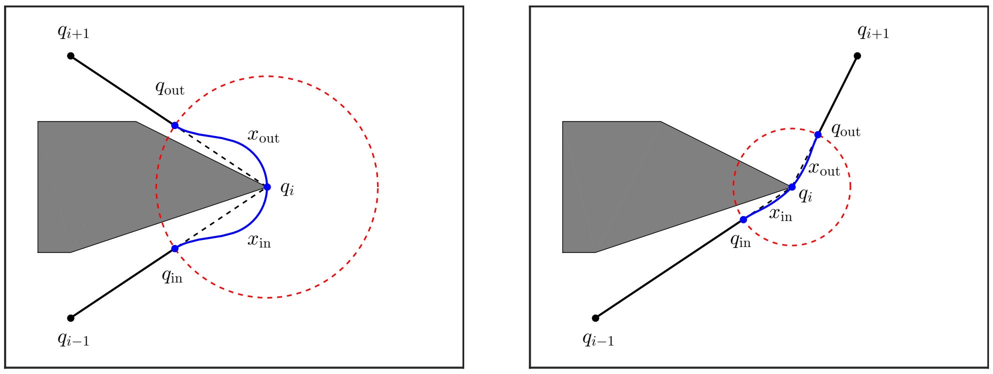
Line-Corner RRT*, or LC-RRT*, combines lines with locally optimal curves to create kinodynamically compatible trajectories as shown in Figure 3.1. The line segments $\overline{q_{i-1}q_i}$ and $\overline{q_iq_{i+1}}$ are based on the path connecting q_{i-1}, q_i and q_{i+1} obtained with L-RRT*, while the curves are generated with a TPBVP solver to guarantee the trajectory complies with the robot's motion model. Transition points q_{in} and q_{out} between lines and curves will depend on the tightness of the corner. The tighter the corner, the more space it will be needed to generate a smooth curve. This means that as the robot approaches the corner, it will at some point q_{in} deviate from the line to make a smooth turn. Figure 3.1 illustrates a tight and an open turn, and the corresponding circular zones (indicated by dotted lines around a corner q_i) in which the turning maneuver is executed. The intersection of the circle centered at a corner point q_i and the line segments $\overline{q_{i-1}q_i}$ and $\overline{q_iq_{i+1}}$ yields points q_{in} and q_{out} that are the boundary values for the TPBVP.

The radius r of the circular zone around a corner depends on the internal angle of the corner α, and is defined in terms of the logistic funciton:

$$r : \alpha \in [0, \pi] \to (r_{\text{max}} - r_{\text{min}}) \left(\frac{1}{1 + e^{k(\alpha - \alpha_0)}} \right) + r_{\text{min}},$$

where r_{min} and r_{max} are the minimum and maximum radius of the zone around the corner in which the turning maneuver is implemented, k determines the steepness of the curve and α_0 the angle of the midpoint value at which $r = (r_{\text{max}} + r_{\text{min}})/2$. The values of r_{min} and r_{max} are chosen according to the motions that can be performed by the robot. For a car-like robot, the value of r_{max} has to be at least the car's turning radius because this is the distance that would be required to perform a U-turn around the corner in an extreme maneuver for which $\alpha \approx 0$. On the other hand, as the internal angle α approaches π, the maneuver approaches the rectilinear motion, thus r_{min} could be zero. As shown in Figure 3.1, corners with small internal angles require a larger zone to complete the turning maneuver than corners with large internal angles.

LC-RRT* is presented in Algorithm 4. The algorithm iteratively connects each pair of nodes with a combination of lines and curves. The inputs are the path L, found with L-RRT*, and the kinodynamic model Σ. The path L is a set of consecutive states q_i that go from q_{init} to q_{final}, where each state $q_i \in X$. The position $(\mathrm{x}_i, \mathrm{y}_i)$ associated to each state q_i is readily obtained from L-RRT*, but the orientation θ_i must be calculated as the mean of the orientations of the segments $\overline{q_{i-1}q_i}$ and $\overline{q_iq_{i+1}}$, where the orientation of each segment



(A) Tight turn (B) Open turn

FIGURE 3.1. Corner examples for LC-RRT*. The algorithm combines lines with locally optimal curves. Boundary conditions for the optimal control problem depend on the tightness of the corner.

```
Algorithm 4: LC-RRT*
   input: $L, \Sigma$
1  $x^* \leftarrow \emptyset; u^* \leftarrow \emptyset; T^* \leftarrow 0$;
2  foreach $q_i \in L \setminus \{q_{\text{init}}, q_{\text{final}}\}$ do
3      $(\theta_i, \alpha_i) \leftarrow$ Angle$(q_{i-1}, q_i, q_{i+1})$;
4      Update $\theta_i$ in $q_i$;
5      $r_i \leftarrow$ Radius$(\alpha_i)$;
6      $(q_{\text{in}}, q_{\text{out}}) \leftarrow$ Intersect$(q_{i-1}, q_i, q_{i+1}, r_i)$;
7      $(x_{\text{in}}, u_{\text{in}}, T_{\text{in}}) \leftarrow$ BVPSolver$(q_{\text{in}}, q_i, \Sigma)$;
8      $(x_{\text{out}}, u_{\text{out}}, T_{\text{out}}) \leftarrow$ BVPSolver$(q_i, q_{\text{out}}, \Sigma)$;
9      if ObstacleFree$(x_{\text{in}})$ and ObstacleFree$(x_{\text{out}})$ then
10         $x^* \leftarrow \{x^*, x_{\text{in}}, x_{\text{out}}\}$;
11         $u^* \leftarrow \{u^*, u_{\text{in}}, u_{\text{out}}\}$;
12         $T^* \leftarrow T^* + T_{\text{in}} + T_{\text{out}}$;
13 return $(x^*, u^*, T^*)$
```

is defined as the angle between the segment and the horizontal axis (Line 3). The new orientation θ_i is assigned to the state q_i, which is then used as a boundary condition for the TPBVP solver (Line 4). Next, the function `Radius`(q_{i-1}, q_i, q_{i+1}) calculates the radius r_i in terms of the angle α_i between the segments $\overline{q_{i-1}q_i}$ and $\overline{q_i q_{i+1}}$ (Line 5). The circle with center at q_i and radius r_i is intersected with the segments $\overline{q_{i-1}q_i}$ and $\overline{q_i q_{i+1}}$ to find the states q_{in} and q_{out} (Line 6).

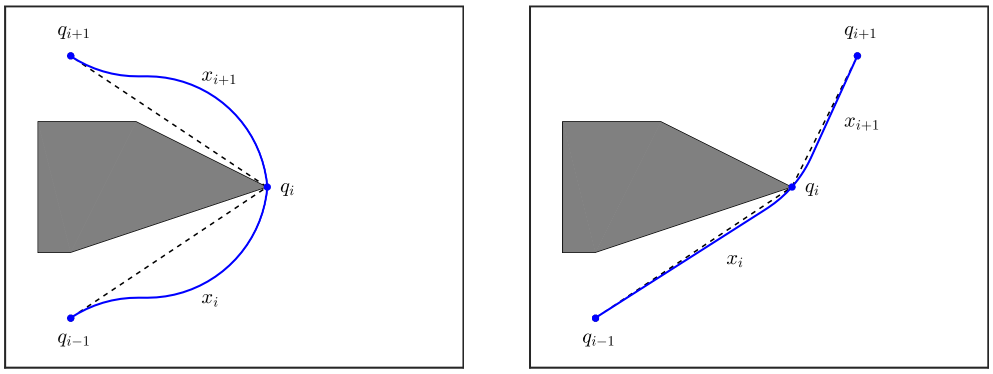
The first part of the turning maneuver is a trajectory x_{in} generated with `BVPSolver`, using q_{in} as the initial condition and q_i as the final condition (Line 7). The output is a locally optimal trajectory $x_{\text{in}} : [0, T_{\text{in}}] \to X$ compatible with the kinodynamic model Σ of the robot, with boundary conditions $x_{\text{in}}(0) = q_{\text{in}}$ and $x_{\text{in}}(T_{\text{in}}) = q_i$, the control trajectories $u_{\text{in}} : [0, T_{\text{in}}] \to U$, and the final time T_{in}. The `BVPSolver` procedure is repeated using q_i as the initial condition and q_{out} as the final condition, to generate the second part of the turning maneuver, yielding the trajectory $x_{\text{out}} : [0, T_{\text{out}}] \to X$, $u_{\text{out}} : [0, T_{\text{out}}] \to U$ and T_{out}. The vertex q_i is used as a mid-point to ensure the trajectories x_{in} and x_{out} found as the solution of the TPBVPs do not collide with obstacles. Hence the turning problem is separated into two consecutive TPBVPs, where the final point of the first TPBVP, $x_{\text{in}}(T_{\text{in}}) = q_i$, is the initial point of the second TPBVP, $x_{\text{out}}(0) = q_i$.

If both trajectories x_{in} and x_{out} are obstacle free, i.e. $x_{\text{in}}(t) \in X_{\text{free}}\ \forall t \in [0, T_{\text{in}}]$ and $x_{\text{out}}(t) \in X_{\text{free}}\ \forall t \in [0, T_{\text{out}}]$ (Line 9), then x_{in} and x_{out} are concatenated into the global trajectory $x^* \in \mathcal{X}$, and u_{in} and u_{out} are concatenated into $u^* \in \mathcal{U}$ (Lines 10-11). The output of LC-RRT* is a global trajectory $x^* : [0, T^*] \to X$, with $x^*(0) = q_{\text{init}}$ and $x^*(T^*) = q_{\text{final}}$, the control trajectories $u^* : [0, T^*] \to U$, and the total time T^*. In Algorithm 4, the trajectories along the line segments $\overline{q_{i-1}q_{\text{in}}}$ and $\overline{q_{\text{out}}q_{i+1}}$ before passing q_{in} and after leaving q_{out}, respectively, are executed at constant velocity and have been omitted for the sake of simplicity.

It is to be noted that even if LC-RRT* method does not generate a globally optimal trajectory, its suboptimality is bounded because L-RRT* retains the asymptotic optimality of RRT* and the number of intermediate states q_i obtained from L-RRT* is finite.

3.2. Corner RRT*

Instead of connecting consecutive nodes q_{i-1}, q_i and q_{i+1} of the line path L with lines and corners as the previous LC-RRT* approach does, it is possible to directly connect nodes q_i by curves calculated with a TPBVP solver in two steps using first boundary conditions q_{i-1} and q_i, and then q_i, q_{i+1} as shown in Figure 3.2. This new method is called Corner



(A) Tight turn (B) Open turn

FIGURE 3.2. Corner examples for C-RRT*. Waypoints are connected with locally optimal curves generated with a TPBVP solver.

RRT*, or C-RRT*, and it is based on the same kinodynamic principle used in LC-RRT*, where the kinodynamic model of the robot is embedded in the motion planning stage. However, C-RRT* does not require the definition of a turning maneuver zone nor the entry and exit points as LC-RRT* (compare Figure 3.1 and Figure 3.2).

The C-RRT* approach is summarized in Algorithm 5. C-RRT* is initialized with the path L found with L-RRT* and iteratively connects successive nodes with locally optimal kinodynamically compatible curves. As for LC-RRT*, the orientation θ_i of every node q_i must be calculated. The function $\texttt{Angle}(q_{i-1}, q_i, q_{i+1})$ calculates the mean of the orientations of the segments $\overline{q_{i-1}q_i}$ and $\overline{q_iq_{i+1}}$ (Line 3). The orientations, together with the positions, define states $q_i = [\mathrm{x}_i\ \mathrm{y}_i\ \theta_i]$, used as boundary conditions for the TPBVP.

For each node q_i in the path L the algorithm solves a TPBVP between the node q_i and the previous node q_{i-1} (Line 6). The output of BVPSolver is a trajectory $x_i : [0, T_i] \to X$ compatible with Σ, with $x_i(0) = q_{i-1}$ and $x_i(T_i) = q_i$, the control signals $u_i : [0, T_i] \to U$, and the time T_i that takes the robot to follow the trajectory. If the trajectory is obstacle free, i.e. $x_i(t) \in X_{\text{free}}\ \forall t \in [0, T_i]$ (Line 7), then x_i is concatenated into the global trajectory $x^* \in \mathcal{X}$ and u_i is concatenated into $u^* \in \mathcal{U}$ (Lines 8-9). Figure 3.2 shows the trajectories generated by C-RRT* for tight and open turns. In both cases C-RRT* yields a smooth path that deviates from the path L generated by L-RRT*.

Algorithm 5: C-RRT*

```
   input: $L, \Sigma$
1  $x^* \leftarrow \emptyset; u^* \leftarrow \emptyset; T^* \leftarrow 0$;
2  foreach $q_i \in L \setminus \{q_{\text{init}}, q_{\text{final}}\}$ do
3      $\theta_i \leftarrow$ Angle$(q_{i-1}, q_i, q_{i+1})$;
4      Update $\theta_i$ in $q_i$;
5  foreach $q_i \in L \setminus \{q_{\text{init}}\}$ do
6      $(x_i, u_i, T_i) \leftarrow$ BVPSolver$(q_{i-1}, q_i, \Sigma)$;
7      if ObstacleFree$(x_i)$ then
8          $x^* \leftarrow \{x^*, x_i\}$;
9          $u^* \leftarrow \{u^*, u_i\}$;
10         $T^* \leftarrow T^* + T_i$;
11 return $(x^*, u^*, T^*)$
```

Like LC-RRT*, the C-RRT* method does not yield a globally optimal trajectory, but its suboptimality is bounded because it inherits the asymptotic optimality of RRT* through L-RRT* and the number of intermediate states q_i in the path L is finite. Furthermore, the trajectory is built from the piecewise solution of an optimal control problem.

4. SIMULATIONS

4.1. Path Tracking Controller

In order to evaluate the performance of the different motion planning algorithms a path tracking controller based on the methodology presented in (Cheein & Scaglia, 2014) was implemented. The work of Cheein and Scaglia proposes a path tracking controller for bicycle platforms, though it can be easily adjusted to work with other platforms, such as a simple unicycle or a differential drive robot. Furthermore, it was shown to yield lower trajectory tracking errors than other path tracking controllers proposed in the recent literature. Using the kinematic model of a unicycle described in Section 2.4, Equation (2.1), the following control law was obtained,

$$\begin{cases} \mathrm{v}_k &= \dfrac{1}{\lambda_\mathrm{v}\Delta_t}\left[\Delta_\mathrm{x}\cos(\theta_{ez,k}) + \Delta_\mathrm{y}\sin(\theta_{ez,k})\right] \\ \omega_k &= \dfrac{\theta_{ez,k+1} - k_\theta\left(\theta_{ez,k} - \theta_k\right) - \theta_k}{\lambda_\omega \Delta_t} \end{cases} \quad \text{with } k_\theta < 1 \tag{4.1}$$

where $\Delta_\mathrm{x} = \mathrm{x}_{\mathrm{ref},k+1} - k_\mathrm{x}(\mathrm{x}_{\mathrm{ref},k} - \mathrm{x}_k) - \mathrm{x}_k$, $\Delta_\mathrm{y} = \mathrm{y}_{\mathrm{ref},k+1} - k_\mathrm{y}(\mathrm{y}_{\mathrm{ref},k} - \mathrm{y}_k) - \mathrm{x}_k$, and θ_{ez} is the value of θ that satisfies Equation (4.2).

$$\frac{\sin(\theta_k)}{\cos(\theta_k)} = \frac{\Delta_\mathrm{y}}{\Delta_\mathrm{x}} \tag{4.2}$$

The values of k_x, k_y and k_θ define the behavior of the controller. To find adequate values for the control constants, a given cost function must be optimized.

The performance of the trajectory tracking controller is measured as a combination of the *total positional error cost*, defined as the summation $C_{\mathrm{x,y}} = \sum_k C_{\mathrm{x,y};k}$ of instantaneous positional errors $C_{\mathrm{x,y};k} = (\mathrm{x}_{\mathrm{ref},k} - \mathrm{x}_k)^2 + (\mathrm{y}_{\mathrm{ref},k} - \mathrm{y}_k)^2$, and the *total effort of the control law*, defined as the summation $C_{\mathrm{v},\omega} = \sum_k C_{\mathrm{v},\omega;k}$ of instantaneous controller efforts $C_{\mathrm{v},\omega} = \mathrm{v}_k^2 + \omega_k^2$, where v_k and ω_k are control inputs for the translational and angular velocity of the robot. Therefore, the controller performance is calculated as:

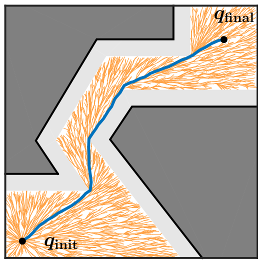
$$C = \sum_k (\mathrm{x}_{\mathrm{ref},k} - \mathrm{x}_k)^2 + (\mathrm{y}_{\mathrm{ref},k} - \mathrm{y}_k)^2 + \sum_k \mathrm{v}_k^2 + \omega_k^2 \tag{4.3}$$

A Monte Carlo experiment was performed to find the values of k_x, k_y and k_θ that minimize the cost function given by Equation (4.3). To ensure zero error convergence of the path tracker the values of the constants must be bounded, i.e. $0 \leq k_\mathrm{x}, k_\mathrm{y}, k_\theta \leq 1$. The values found with a 5000 iterations Monte Carlo experiment were $k_\mathrm{x} = 0.98$, $k_\mathrm{y} = 0.98$, and $k_\theta = 0.92$. For more details about the implementation of the controller the reader is referred to (Cheein & Scaglia, 2014).

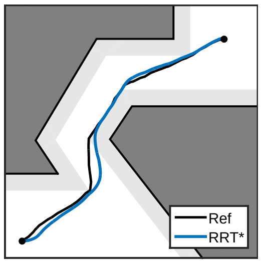
4.2. Simulation Results

Simulations were carried out to show the advantages and drawbacks of the proposed algorithms, and evaluate their ability to plan kinodynamically compliant trajectories while achieving global quasi-optimality, i.e. the trajectories are suboptimal by a bounded factor. Four different scenarios were considered. The first two shown in Figure 4.1 and Figure 4.2 correspond to tunnels with open and sharp turns. These were created to test the algorithms in situations similar or more complex that encountered in underground mining tunnels. In both cases, there is only one possible route to the goal, so these scenarios are intended to generate situations in which maneuvering and generating kinodynamically feasible paths is the main challenge. The third scenario has two possible paths to the goal, while the fourth scenario has more than three possible paths to the goal. The third and fourth scenarios, shown in Figure 4.3 and Figure 4.4, represent other situations encountered in industrial environments and were conceived to test the capacity of the approaches to find a globally optimal solution. The fourth scenario was purposefully made similar to a loop-closing test to evaluate possible cumulative path tracking errors.

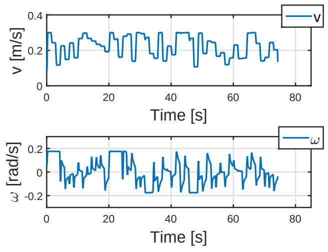
The obstacles in the different scenarios are constructed with polygons for simplicity and to keep a low computational overhead. A considerable amount of computation time



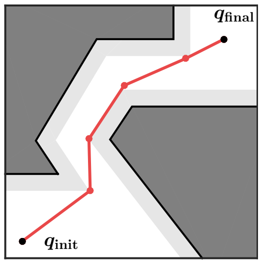
(A) RRT* Path



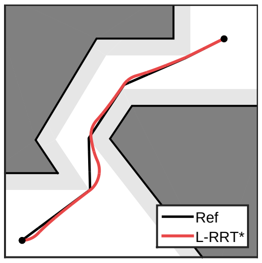
(B) RRT* Tracked Path



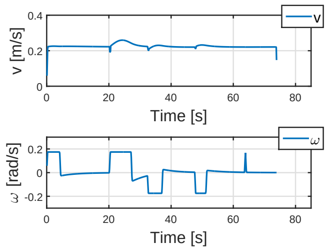
(C) RRT* Controls



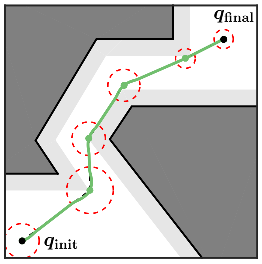
(D) L-RRT* Path



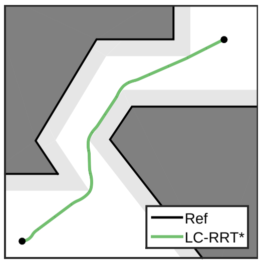
(E) L-RRT* Tracked Path



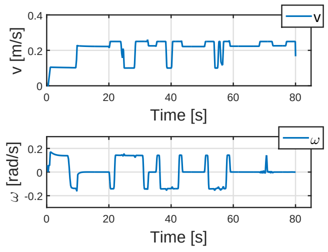
(F) L-RRT* Controls



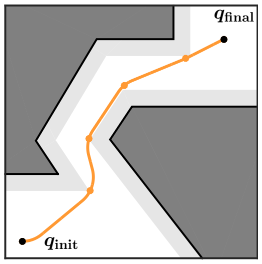
(G) LC-RRT* Path



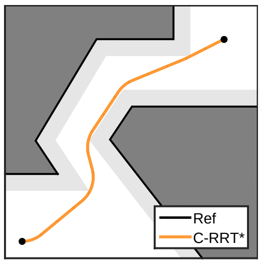
(H) LC-RRT* Tracked Path



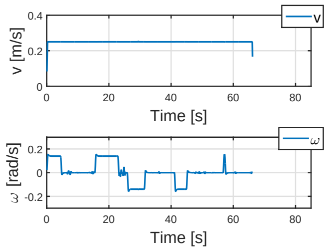
(I) LC-RRT* Controls



(J) C-RRT* Path

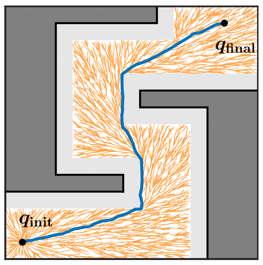


(K) C-RRT* Tracked Path

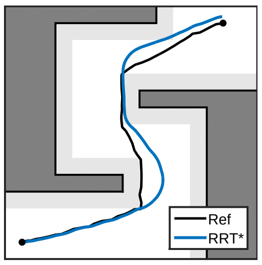


(L) C-RRT* Controls

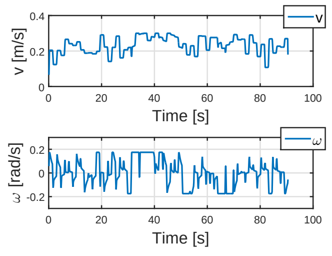
FIGURE 4.1. Simulations for RRT*, L-RRT*, LC-RRT*, and C-RRT* in scenario 1. Subfigures (A), (D), (G) and (J) show the trajectories generated with each algorithm. Subfigures (B), (E), (H) and (K) show the reference trajectory and the followed trajectory. Subfigures (C), (F), (I) and (L) show the control signals for every case.



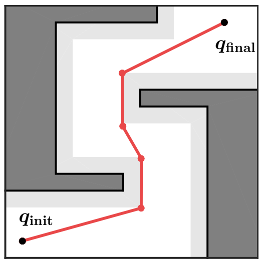
(A) RRT* Path



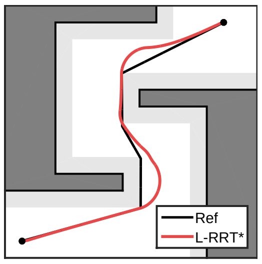
(B) RRT* Tracked Path



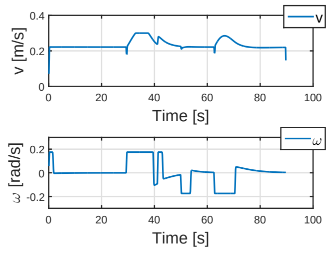
(C) RRT* Controls



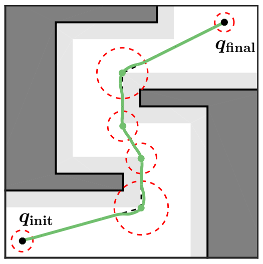
(D) L-RRT* Path



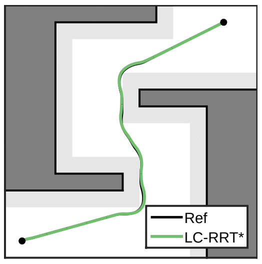
(E) L-RRT* Tracked Path



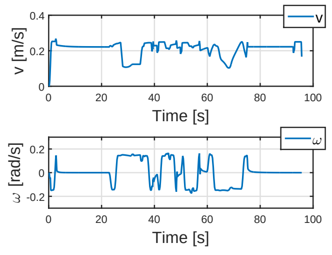
(F) L-RRT* Controls



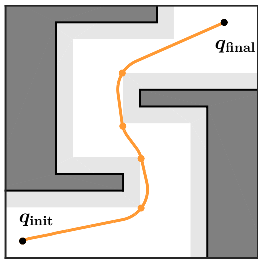
(G) LC-RRT* Path



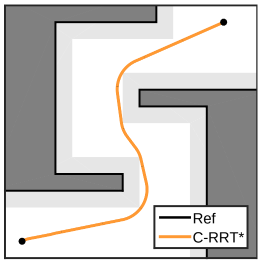
(H) LC-RRT* Tracked Path



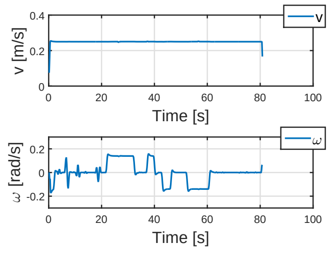
(I) LC-RRT* Controls



(J) C-RRT* Path

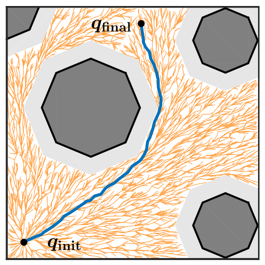


(K) C-RRT* Tracked Path

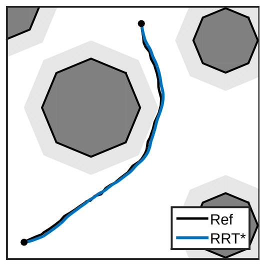


(L) C-RRT* Controls

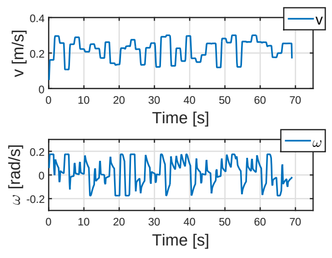
FIGURE 4.2. Simulations for RRT*, L-RRT*, LC-RRT*, and C-RRT* in scenario 2. Subfigures (A), (D), (G) and (J) show the trajectories generated with each algorithm. Subfigures (B), (E), (H) and (K) show the reference trajectory and the followed trajectory. Subfigures (C), (F), (I) and (L) show the control signals for every case.



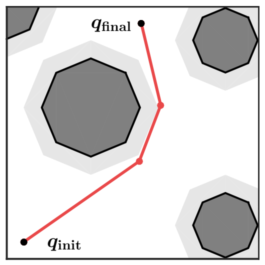
(A) RRT* Path



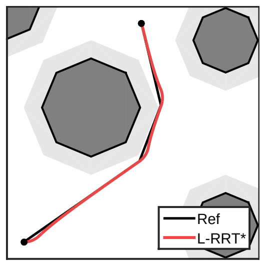
(B) RRT* Tracked Path



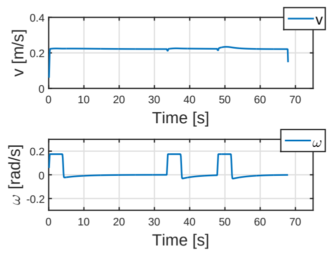
(C) RRT* Controls



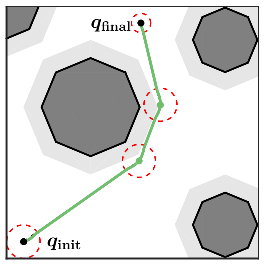
(D) L-RRT* Path



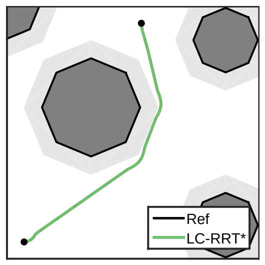
(E) L-RRT* Tracked Path



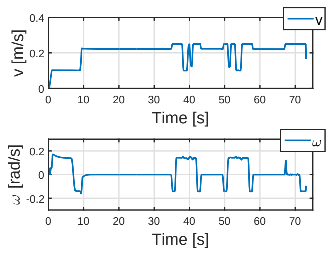
(F) L-RRT* Controls



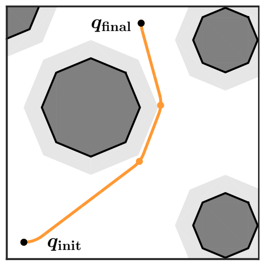
(G) LC-RRT* Path



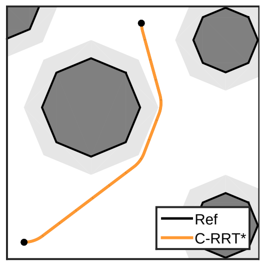
(H) LC-RRT* Tracked Path



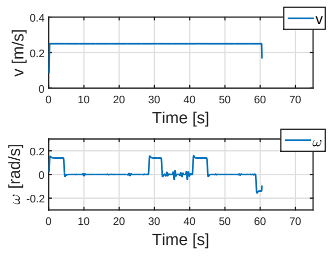
(I) LC-RRT* Controls



(J) C-RRT* Path

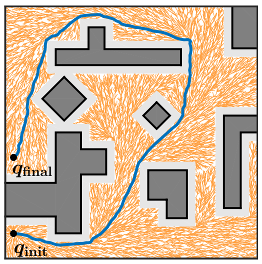


(K) C-RRT* Tracked Path

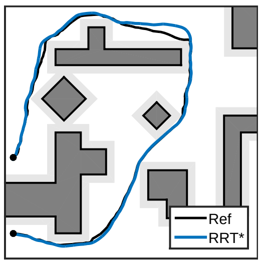


(L) C-RRT* Controls

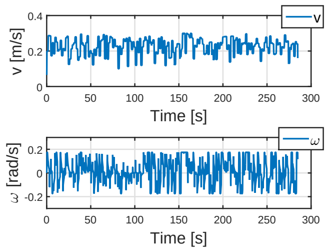
FIGURE 4.3. Simulations for RRT*, L-RRT*, LC-RRT*, and C-RRT* in scenario 3. Subfigures (A), (D), (G) and (J) show the trajectories generated with each algorithm. Subfigures (B), (E), (H) and (K) show the reference trajectory and the followed trajectory. Subfigures (C), (F), (I) and (L) show the control signals for every case.



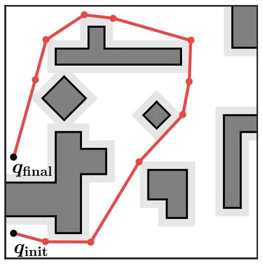
(A) RRT* Path



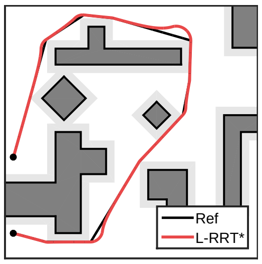
(B) RRT* Tracked Path



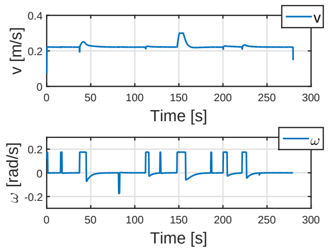
(C) RRT* Controls



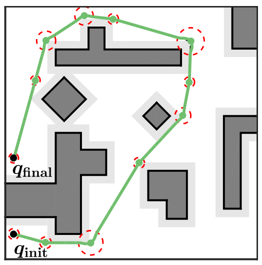
(D) L-RRT* Path



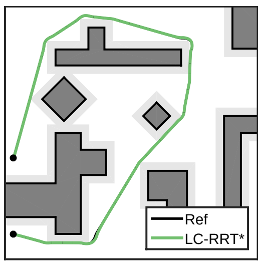
(E) L-RRT* Tracked Path



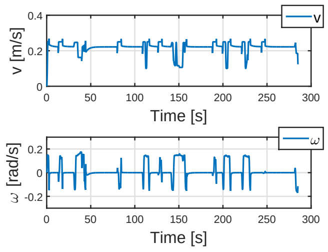
(F) L-RRT* Controls



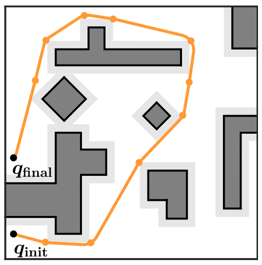
(G) LC-RRT* Path



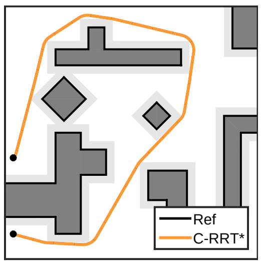
(H) LC-RRT* Tracked Path



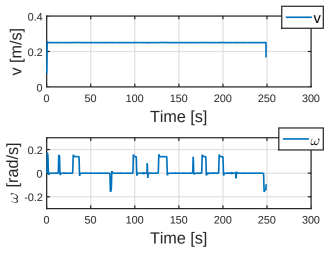
(I) LC-RRT* Controls



(J) C-RRT* Path



(K) C-RRT* Tracked Path



(L) C-RRT* Controls

FIGURE 4.4. Simulations for RRT*, L-RRT*, LC-RRT*, and C-RRT* in scenario 4. Subfigures (A), (D), (G) and (J) show the trajectories generated with each algorithm. Subfigures (B), (E), (H) and (K) show the reference trajectory and the followed trajectory. Subfigures (C), (F), (I) and (L) show the control signals for every case.

corresponds to obstacle collisions checking, thus an efficient obstacle representation is crucial. As with most path planning algorithms, the size of the robot is not considered, which can cause collisions since the optimal paths will tangentially approach the obstacles. In order to avoid these type of collisions the obstacles are dilated by half the width of the robot plus a security factor. The motion model considered in the simulations is a simplified version of the kinodynamic model of the CAT 262 skid-steer loader developed in (Aguilera, Torres-Torriti, & Auat, 2014, 2015). Here the arm of the excavator is assumed to be fixed, thus its dynamics is not taken into account. On the other hand, only planar motion of the mobile platform on a 2D flat surface is considered without the wheel-ground interaction dynamics nor variations in terrain slope. More details are available in Section 2.4.

The simulation results presented in Figures 4.1–4.4 show the trajectories generated by motion planning algorithms in subfigures (A), (D), (G), (J) for RRT*, L-RRT*, LC-RRT* and C-RRT*, respectively. The corresponding trajectories resulting from the application of the path tracking controller are shown in subfigures (B), (E), (H), (K) for each of the four scenarios. The third column in each figure shows the longitudinal and angular velocity command signals computed by path tracking controller with the reference trajectory generated by each algorithm in subfigures (C), (F), (I), (L). In all scenarios, the subfigure (A) shows the trajectory planned with RRT* and the tree spanning from the start node after a maximum of 5000 iterations or less if the improvement between iterations is below the stopping threshold. Additionally, the total instantaneous cost for each scenario, corresponding to the combined tracking error and controller performance, are shown in Figures 4.5–4.8, respectively.

The results for the first scenario is presented in Figure 4.1 show that since no kinodynamic model is considered by RRT*, the controller is unable to make the robot correctly follow the path, which has a large tracking error (see the comparison between the planned and executed trajectory in Figure 4.1B). Furthermore, due to its random nature, RRT* only converges to a smooth path in an infinite number of iterations, thus the resulting control signals are more oscillatory, as shown in Figure 4.1C. L-RRT* improves the quality of the RRT* path by connecting nodes with collision free lines, as depicted in Figure 4.1D.

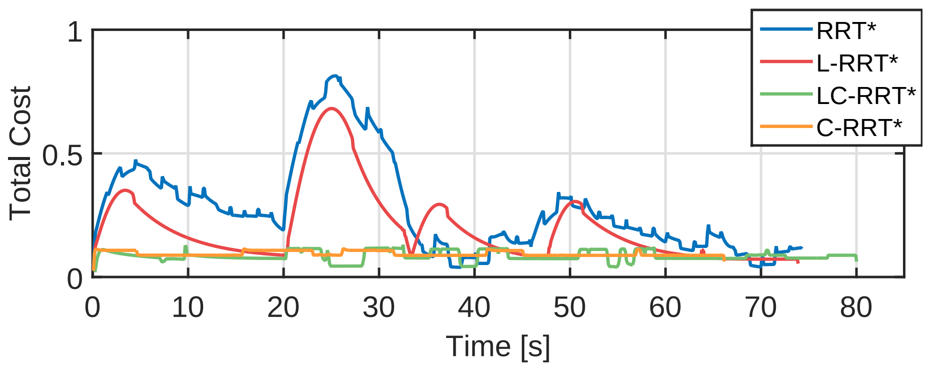


FIGURE 4.5. Instantaneous cost $C_{\mathrm{x,y};k}$ for scenario 1.

This solution eliminates the oscillatory behavior of RRT* and smooths the control signals, as seen in Figure 4.1F. Nevertheless, the incompatibility between the trajectory and the kinodynamic model generates tracking errors in the followed trajectory, clearly visible in Figure 4.1E. LC-RRT* further improves the quality of L-RRT* by replacing the corners with kinodynamically compatible turns, as depicted in Figure 4.1G. As a result the tracking error in the corners is clearly reduced, as seen in Figure 4.1H. Figure 4.1I shows the control signals for the LC-RRT* case. The combination of lines and curves generates a forced behavior on the controller in the turning sections, causing a decrease in the longitudinal velocity v and a small oscillation in the angular velocity ω. Finally, it is possible to see in Figure 4.1K that the trajectory produced by C-RRT* is tracked without any appreciable errors. In contrast to LC-RRT*, C-RRT* control signals are smoother and show no oscillation in longitudinal velocity, as can be seen from the comparison of Figure 4.1I and Figure 4.1L. A comparison of the instantaneous cost $C_{\mathrm{x,y};k}$ between the algorithms can be seen in Figure 4.5. The effects of the tracking errors in RRT* and L-RRT* are clearly visible, generating a high cost. On the other hand LC-RRT* and C-RRT* maintain a low cost during the trajectory.

Comparing the second scenario in Figure 4.2 and that of the first scenario in Figure 4.1, it is possible to observe that the tracking errors are larger, particularly for RRT* in Figure 4.2B. Once again, RRT* without any path smoothing strategy nor any kinodynamic model produces an oscillatory trajectory and controls as seen in Figure 4.2C. In comparison

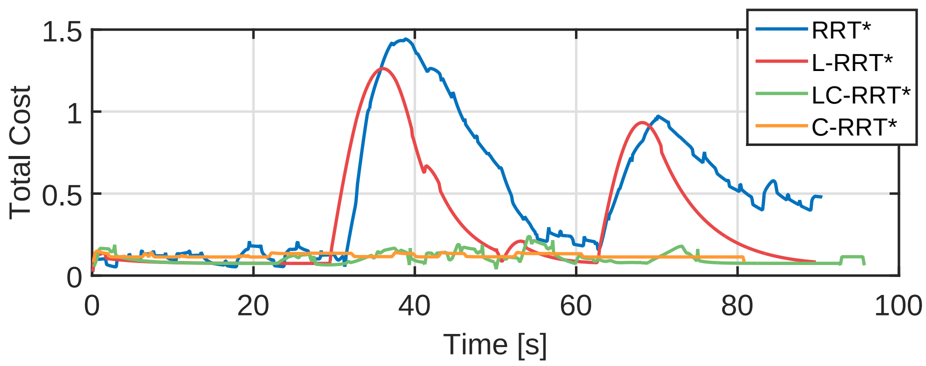


FIGURE 4.6. Instantaneous cost $C_{\mathrm{x,y};k}$ for scenario 2.

to RRT*, major improvements can be obtained with L-RRT*, Figure 4.2D, regarding the oscillations of the control signals; see Figure 4.2F. Nevertheless, a considerable tracking error is still present, as shown in Figure 4.2E. Figure 4.2G presents the trajectory generated with LC-RRT*. As for the first scenario, in this case LC-RRT* also reduces the tracking error considerably; see Figure 4.2H. Figure 4.2K shows that once again C-RRT* produces no visible tracking errors. It is possible to see in Figure 4.2L that the longitudinal velocity signal is also constant and only changes in the angular velocity control occur for every turning maneuver along the path. Once again, the costs of LC-RRT* and C-RRT* are significantly lower than the ones of RRT* and L-RRT*, as seen in Figure 4.6.

For the third scenario presented in Figure 4.3, RRT* shows an oscillatory behavior and a significant tracking error like in the previous cases; see Figure 4.3C. The oscillatory behavior is reduced with L-RRT* (see Figure 4.3F), but the tracking error is still present (see Figure 4.3E). LC-RRT* and C-RRT* reduce the tracking error to a negligible amount, as can be seen in Figure 4.3H and Figure 4.3K. The results show that C-RRT* finds a path with very few turning maneuvers and constant velocity, as can be observed in Figure 4.3L. The total cost is very small and practically constant, compared to RRT* and L-RRT* as in Figure 4.7.

Finally, in spite of the obstacles and different possible paths in the fourth scenario, all algorithms converge to the almost optimal solution thanks to the global asymptotic optimality of the underlying RRT* as shown in Figure 4.4. As for the previous cases, the

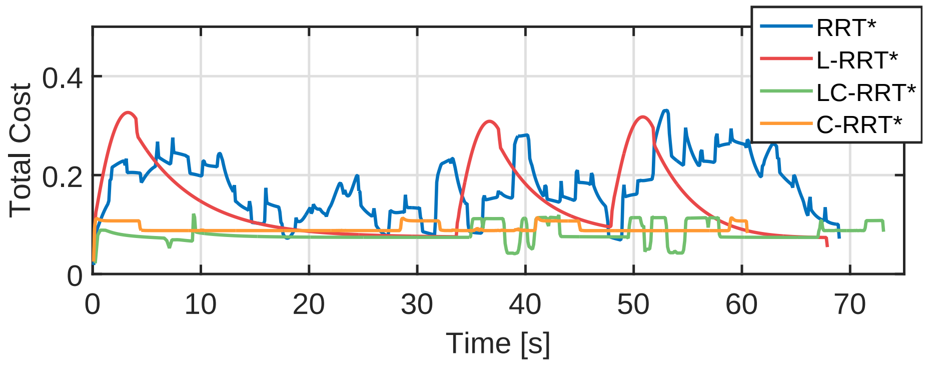


FIGURE 4.7. Instantaneous cost $C_{\mathrm{x},\mathrm{y};k}$ for scenario 3.

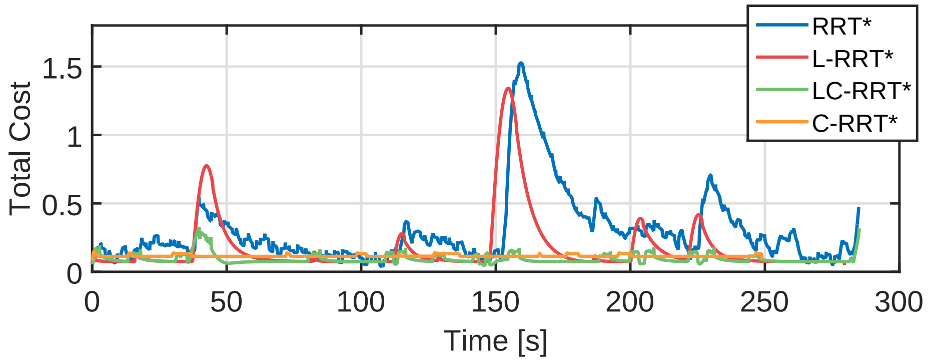


FIGURE 4.8. Instantaneous cost $C_{\mathrm{x},\mathrm{y};k}$ for scenario 4.

executed trajectory for RRT* and L-RRT* has overshoots where tight turning maneuvers must be performed, which leads to significant tracking errors as shown in Figure 4.4B and Figure 4.4E. This problem is corrected by LC-RRT* and C-RRT*, as seen in Figure 4.4H and Figure 4.4K. As with the previous scenarios, the proposed methods effectively reduce the total cost, as depicted in Figure 4.8.

Table 4.1 summarizes trajectory tracking performance results obtained with RRT*, L-RRT*, LC-RRT* and C-RRT* for each scenario. The performance index is calculated as explained in Section 4.1. For every case the total cost for each algorithm is provided together with the relative cost reduction. The cost reduction is calculated relative to the RRT* approach taken as base method. As expected, RRT* has the highest cost in all scenarios because it does not take into account the motion model of the robot and because convergence to a smooth solution requires a infinite number of iterations. L-RRT* shows in average a

TABLE 4.1. Cost comparison for simulation. The total cost is calculated as a combination of the tracking error and the controller effort, $C = C_{\mathrm{x,y}} + C_{\mathrm{v},\omega}$. The cost reduction compared to RRT* is calculated as $\Delta C = \frac{C^{\mathrm{RRT*}} - C^{i}}{C^{\mathrm{RRT*}}}$, $i =$ L-RRT*, LC-RRT*, C-RRT*.

Algorithm	Scenario 1		Scenario 2		Scenario 3		Scenario 4	
	C	ΔC	C	ΔC	C	ΔC	C	ΔC
RRT*	206.0	-	440.7	-	120.9	-	839.2	-
L-RRT*	147.9	$-28.2\,\%$	287.2	$-34.8\,\%$	98.0	$-18.9\,\%$	489.1	$-41.7\,\%$
LC-RRT*	67.9	$-67.0\,\%$	99.7	$-77.4\,\%$	59.1	$-51.1\,\%$	268.6	$-68.0\,\%$
C-RRT*	62.4	$-69.7\,\%$	97.8	$-77.8\,\%$	55.8	$-53.8\,\%$	289.2	$-65.5\,\%$

$30.9\,\%$ cost reduction, compared to RRT*. This improvement in the tracked path can be attributed to the smoothening of the path, which significantly reduces the oscillatory behavior of the path tracking controller. However, the controller still cannot make the robot follow precisely the path at tight corners. The motion planners that consider the kinodynamic constraints, LC-RRT* and C-RRT*, in the planning stage generate trajectories that can be properly followed by the real robot, thus reducing the tracking error. On average a $65.9\,\%$ and $66.7\,\%$ reduction is respectively achieved with LC-RRT* and C-RRT*, compared to RRT*. The differences between planners are highlighted in scenarios with tight corners, such as scenario 2 (Figure 4.2), where the cost reduction of LC-RRT* and C-RRT* is $77.4\,\%$ and $77.8\,\%$, respectively. In a simpler scenario with no tight curves, such as scenario 3 (Figure 4.3), the cost reduction is less significant, $51.1\,\%$ for LC-RRT* and $53.8\,\%$ for C-RRT*, but nevertheless is still noteworthy. It is to be noted that both proposed algorithms LC-RRT* and C-RRT* outperform L-RRT* and RRT* in every scenario by a significant margin. Thus highlighting the importance of considering the motion model in the trajectory planning strategy.

Concerning the optimilaty of each path measured in term of its length, the differences between the approaches are negligible and less than $1\,\%$ on average for the different scenarios because all are based on RRT* algorithm to generate the base path. It must be stressed that the proposed LC-RRT* and C-RRT* approaches refine the RRT* path making it compatible with the motion model and without compromising the path lenght of the base path solved by RRT*. In terms of execution time, the path computed with the C-RRT* approach is executed $8 - 10\,\%$ faster than the path produced by RRT* or L-RRT*. This is explained by the reduction in oscillations and overshoots that is possible to achieve with the C-RRT* approach.

The ability of LC-RRT* and C-RRT* to generate paths that are compatible with the motion model of the robot is not only a desirable feature, but an aspect that can make a big difference between avoiding and running into possible collisions. An example of the importance of the ability to foreseeing unfeasible motions and ensuring that motions can be tracked by the controller is found in the first and second scenarios, where RRT* and L-RRT* generate paths that are not tracked precisely and the trajectories slightly cuts the safe zones, as seen Figure 4.1B, Figure 4.1E and Figure 4.2B. In these scenarios with tight passages, LC-RRT* and C-RRT* do not have problems to generate a feasible and safe trajectory.

5. IMPLEMENTATION AND TESTING METHODOLOGY

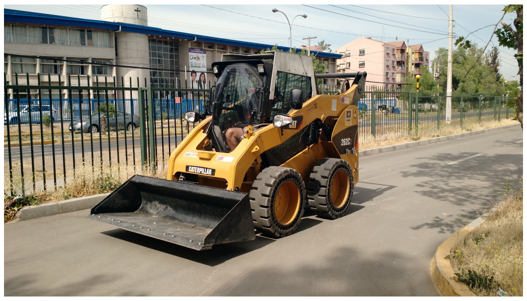
5.1. Experimental setup

Experiments were conducted using a CAT262C skid-steer compact loader, shown in Figure 5.1, in order to validate the proposed motion planning algorithms. A system implementation diagram showing the interactions between the different parts of the system is depicted in Figure 5.2. The GPS system is directly connected to the navigation computer, while the encoders signals are processed by our custom made signal conditioning and communication circuitry, indicated in the figure as MasterBoard R3, which passes the odometry to the navigation computer and sends the control signals to the machine.

The CAT262C loader is not designed for research purposes, and thus it is not easy to modify or manipulate as most research experimental platforms, however it provides a more realistic approach to the industrial world. To manually control the machine an operator handles the joysticks, which through the duty-cycle of a PWM signal adjust the power delivered to the actuators. To conduct the experiments a computer-machine interface was needed. For this purpose an electrical board, MasterBoard R3, was design by the Robotics & Automation Laboratory (RAL) PUC research team to emulate the PWM signals generated by the joysticks, among other tasks.

In order to establish a reliable communication between the computer and the MasterBoard R3 a new communication protocol was designed, the Robot Communication Protocol (RCP). RCP is a fast, simple, and minimal binary protocol for communicating between computers and robots. It can be used to transmit control signals to the robot, and retrieve information from the sensors in a fast and reliable way. The message structure allows variable payload size, making it suitable for a wide range of message types. Includes a Cycle Redundancy Check (CRC) for error detection, thus improving the security standards needed for operating industrial machinery. Details about RCP can be found in Appendix B.

A SwitfNav Piksi real-time kinematic (RTK) GPS was used. RTK systems use two units, base and rover, for achieving centimeter level accurate relative position. In order to



(A) Front view

(B) Back view

FIGURE 5.1. Compact skid-steer loader used for experimental validation.

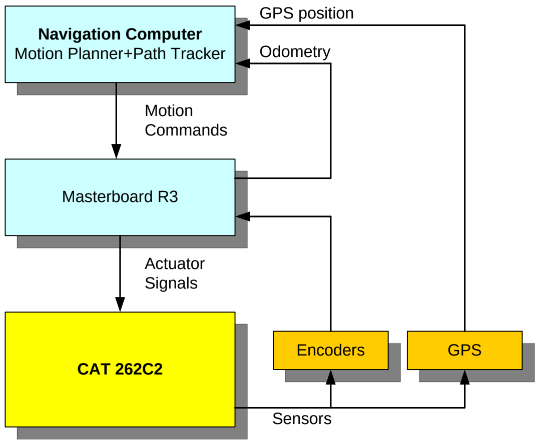


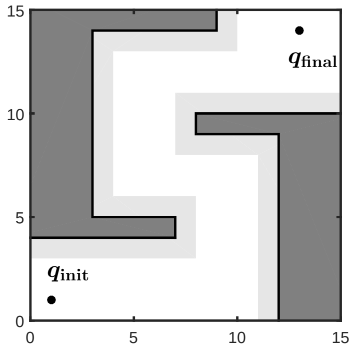
FIGURE 5.2. System architecture diagram that shows the interaction between the main components of the navigation system robotic skid-steer loader.

achieve this level of accuracy the base station sends observation data to the rover, used to correct its position. The information is used for two purposes, first allows to minimize the error induced by the earth's atmosphere as long as the rover is close to the base, and second uses the difference in phase of the two carrier waves to reduce the error to a few centimeters. The internal encoders of the machine were used for position correction when the GPS signal was lost. The internal encoders of the CAT262C are coupled to one wheel of each side, producing 680 pulses per revolution, which is equivalent to 0.5294 degrees per count. Even with the high accuracy of the RTK system data filtering is highly recommended, so an Extended Kalman Filter was implemented to reduce the variability of the GPS measurements. The kinematic model a differential robot, stated in Equation (2.1), was used for the EKF.

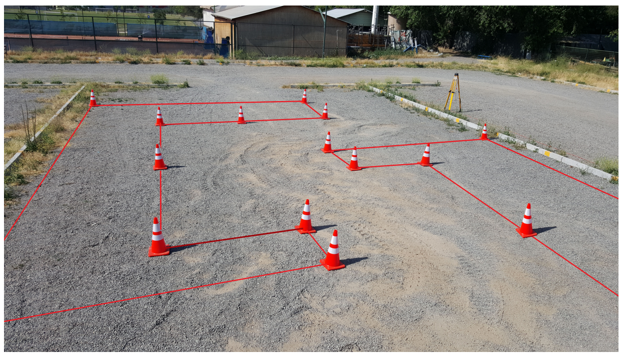
The RCP protocol and low level controllers for the MasterBoard R3 were implemented on the onboard microcontroller, an Atmel SAM3X8E ARM Cortex-M3 CPU running with a $84\,\mathrm{kHz}$ clock and capable of sending encoder measurements and maneuvering commands every $40\,\mathrm{ms}$. The middle level path tracking controller and the EKF were implemented in the navigation computer with an Intel Core i7 $2.6\,\mathrm{GHz}$ processor. The computer was directly connected to the MasterBoard R3, and all the communications between the devices used the RCP protocol. Middle level algorithms for path tracking were implemented in C++, taking advantage of the extensive use of this language in robotics applications and its real time capabilities. High level motion planning algorithms were implemented in MATLAB® on the same navigation computer. To solve the numerous TPBVPs with a low computational overhead a numerical solver is used, the ACADO Toolkit (Houska, Ferreau, & Diehl, 2011). The toolkit is implemented in C++, which makes it suitable for almost real-time implementations. The amount of time needed by the solver to find a solution depends highly on the problem type, the accuracy needed, and the initial solution.

5.2. Testing location

Field tests were performed to evaluate the motion planning algorithms' performance in practice. The tests were conducted on an empty parking lot, with enough space to perform the same trajectories that were used for simulations. Two of the simulated scenarios were used for the real world experiments, scenario 2 and scenario 3, shown in Figure 5.3 and Figure 5.4, respectively. Traffic cones were used to mark the vertices of the obstacles, which were positioned with the RTK GPS system to match the scenario used in simulations.

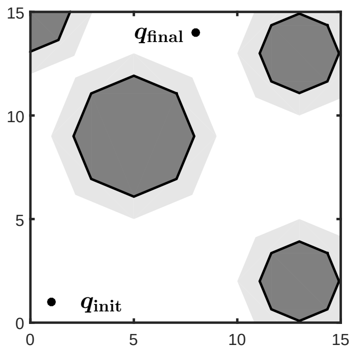


(A) Map

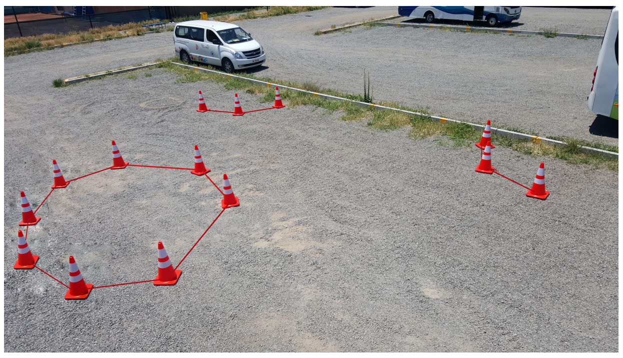


(B) Real world layout

FIGURE 5.3. Field experiment for simulation scenario 2. Traffic cones mark the vertices of the obstacles.



(A) Map



(B) Real world layout

FIGURE 5.4. Field experiment for simulation scenario 3. Traffic cones mark the vertices of the obstacles.

6. EXPERIMENTAL RESULTS WITH A ROBOTIC SKID-STEER LOADER

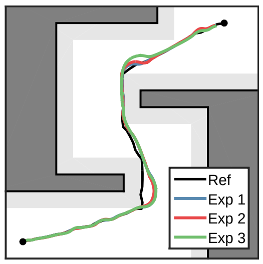
This section presents the experimental results of implementing the proposed motion planning algorithms in an automatized industrial compact loader. To minimize the effects of noise in the different parts of the systems, three repetitions were performed for each one of the tested algorithms.

The results obtained from the field experiments corresponding to the simulation scenarios 2 and 3 are presented in Figure 6.1 and Figure 6.2, respectively. For each planning strategy (RRT*, L-RRT*, LC-RRT*, and C-RRT*), the reference planned trajectory (labeled Ref) and three repetitions of the executed trajectory (labeled Exp 1, Exp 2 and Exp 3) are shown in the first column of the plots. The similitude between repetitions demonstrates the consistent behavior of the controller over time. The control signals corresponding to the longitudinal velocity v and angular velocity ω are presented in the second column of Figure 6.1 and Figure 6.2 (blue curves labeled ref). The control signals illustrate the effort of the controller. The same graphs also include the measured longitudinal and angular velocities (purple curves labeled meas). The instantaneous tracking error of the trajectory $C_{\mathrm{x,y};k}$ and the controller effort $C_{\mathrm{v},\omega;k}$ are plotted in the last column of Figure 6.1 and Figure 6.2 to compare the differences between algorithms.

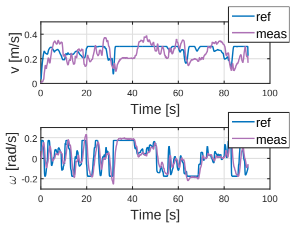
The first field experiment that implements the simulation scenario 2 poses the most challenging situation because of the tighter turns through a narrow zig-zagging passage, more complex than standard underground mining tunnels. Comparing Figure 4.2 and Figure 6.1, it is possible to confirm that the experimental results obtained are consistent with simulations. RRT* is able to find a path, but the path tracker is unable to follow the path during tight corners because it is not kinodynamically compatible. Furthermore, the oscillatory behavior of RRT* in simulations is also appreciable in real world experiments, as depicted in Figure 6.1A and Figure 6.1B. L-RRT* solves part of the oscillatory problem, but is equally unable to generate trackable turns at the tight corners, as can be seen in Figure 6.1D. On the other hand, LC-RRT* and C-RRT* generate trajectories that comply with the motion model of the robot, thus the robot is able to properly follow the trajectory, as seen in Figure 6.1G and Figure 6.1J. Comparing the instantaneous tracking errors and controller efforts, $C_{\mathrm{x,y};k}$ and $C_{\mathrm{v},\omega;k}$, of the different algorithms, it is possible to observe that LC-RRT* and C-RRT* yield low tracking errors below 0.3-0.5, while RRT* exceeds 1.0 and L-RRT* reaches 0.75 at the cornering points. It is also to be noted that LC-RRT* and C-RRT* yield lower controller efforts compared to RRT* or L-RRT*, as the executed paths follow a feasible trajectories in the case of LC-RRT* and C-RRT*.

Similar conclusions can be drawn from the second field experiment corresponding to simulation scenario 3, presented in Figure 6.2. Once again, the trajectories exectued using the RRT* approach present an oscillatory behavior, as seen in Figure 6.2A, consequently generating oscillatory control signals, depicted in Figure 6.2B. Even with a simple smoothing procedure, such as L-RRT*, the oscillatory behavior can be significantly reduced, as shown in Figure 6.2D and Figure 6.2E. However, despite being smoother, the L-RRT* trajectory has overshoots at the turns, which are negligible in the case of LC-RRT* and C-RRT*, as shown in Figure 6.2G and Figure 6.2J, because of the compatibility between the planned trajectory and the robot's motion model. The LC-RRT* paths are formed with lines and curves, thus the increases in the controller effort at the corners of the trajectory, whereas C-RRT* paths are formed using only curves, which distributes the controller effort throughout the duration of the trajectory. Although the magnitudes of the control signals for LC-RRT* and C-RRT* shown in Figure 6.2H and Figure 6.2K look similar, the fact that C-RRT* produces a trajectory that can be followed without reducing the longitudinal velocity and maintaining an angular velocity with less number of sudden changes, C-RRT* yields a trajectory that can be followed in a shorter time, almost 13 seconds less than that produced by LC-RRT*. This fact can also be observed from the figures for the first field experiment shown in Figure 6.1, for which C-RRT* produces a trajectory that can be completed almost 15 seconds faster than LC-RRT*, because C-RRT* is able to maintain and almost constant longitudinal velocity and reduces the variations in angular velocity. This is an important advantage of C-RRT*.

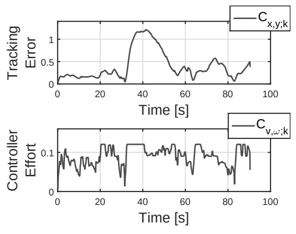
The total path tracking error and controller efforts as cost C explained in Section 4.1 obtained for the field experiments are summarized in Table 6.1. As in simulations, RRT*



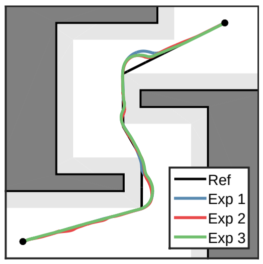
(A) RRT* Path



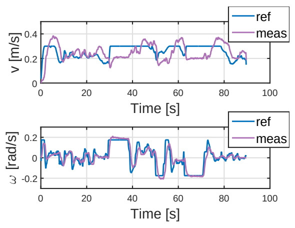
(B) RRT* Controls



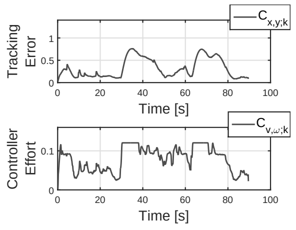
(C) RRT* Cost



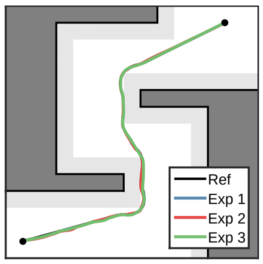
(D) L-RRT* Path



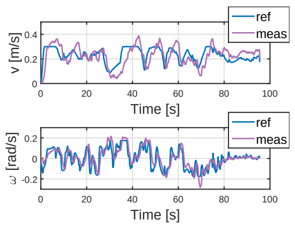
(E) L-RRT* Controls



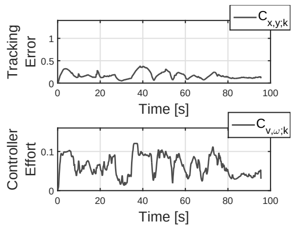
(F) L-RRT* Cost



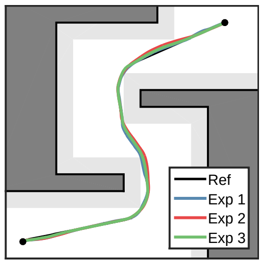
(G) LC-RRT* Path



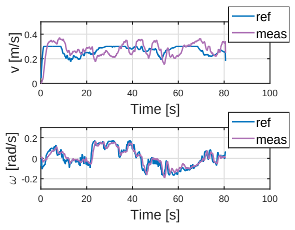
(H) LC-RRT* Controls



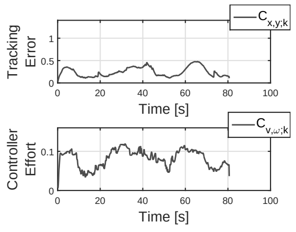
(I) LC-RRT* Cost



(J) C-RRT* Path

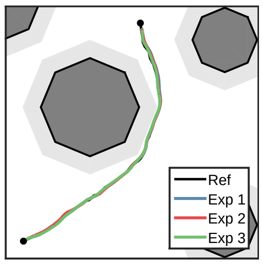


(K) C-RRT* Controls

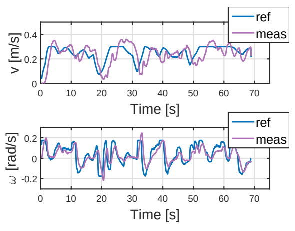


(L) C-RRT* Cost

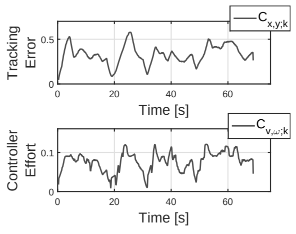
FIGURE 6.1. Experimental validation for RRT*, L-RRT*, LC-RRT*, and C-RRT* in scenario 2. Subfigures (A), (D), (G) and (J) show the reference trajectory and the real followed trajectory. Subfigures (B), (E), (H) and (K) show the control signals for every case. Subfigures (C), (F), (I) and (L) present the tracking error and controller effort for each algorithm.



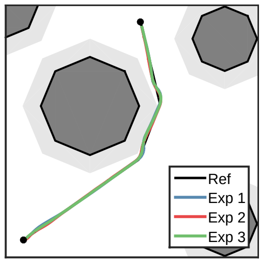
(A) RRT* Path



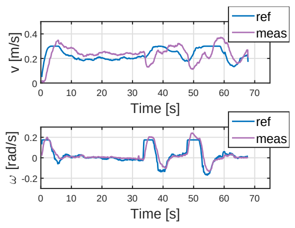
(B) RRT* Controls



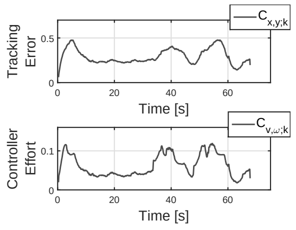
(C) RRT* Cost



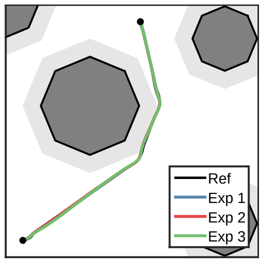
(D) L-RRT* Path



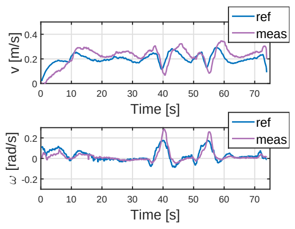
(E) L-RRT* Controls



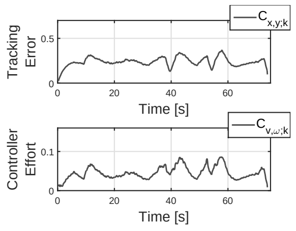
(F) L-RRT* Cost



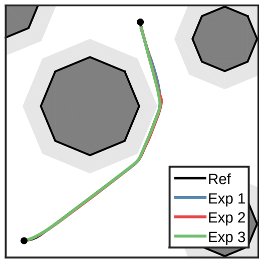
(G) LC-RRT* Path



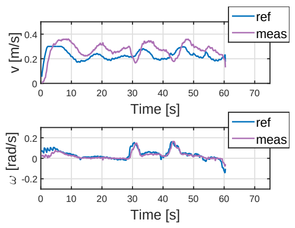
(H) LC-RRT* Controls



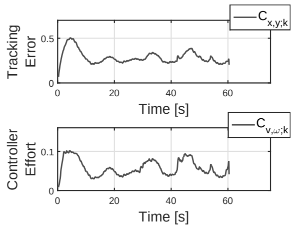
(I) LC-RRT* Cost



(J) C-RRT* Path



(K) C-RRT* Controls



(L) C-RRT* Cost

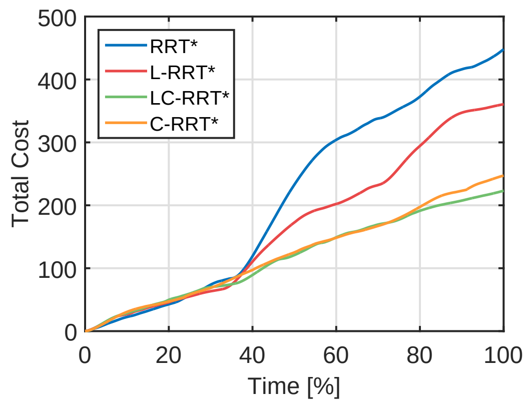
FIGURE 6.2. Experimental validation for RRT*, L-RRT*, LC-RRT*, and C-RRT* in scenario 3. Subfigures (A), (D), (G) and (J) show the reference trajectory and the real followed trajectory. Subfigures (B), (E), (H) and (K) show the control signals for every case. Subfigures (C), (F), (I) and (L) present the tracking error and controller effort for each algorithm.

TABLE 6.1. Cost comparison for real world experiments. The total cost is calculated as a combination of the tracking error and the controller effort, $C = C_{\mathrm{x,y}} + C_{\mathrm{v},\omega}$. The cost reduction compared to RRT* is calculated as $\Delta C = \frac{C^{\mathrm{RRT*}} - C^i}{C^{\mathrm{RRT*}}}$, $i =$ L-RRT*, LC-RRT*, C-RRT*.

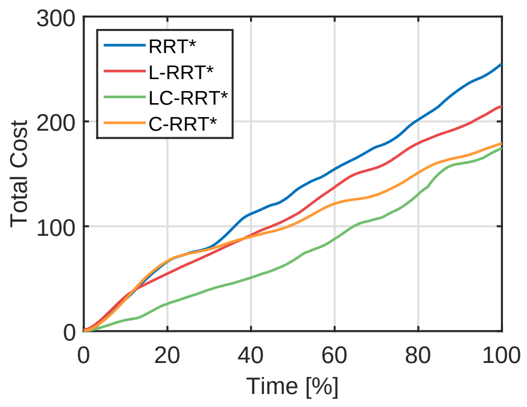
Algorithm	Scenario 2		Scenario 3	
	C	ΔC	C	ΔC
RRT*	447.8	-	254.8	-
L-RRT*	360.6	$-19.5\,\%$	214.0	$-16.0\,\%$
LC-RRT*	223.1	$-50.2\,\%$	174.4	$-31.6\,\%$
C-RRT*	247.1	$-44.8\,\%$	178.9	$-29.8\,\%$

has the highest cost C for both tested scenarios, followed by L-RRT*, with an average decrease of $17.7\,\%$. LC-RTT* and C-RRT* achieve the best performance, with an average decrease, compared to RRT*, of $40.9\,\%$ and $37.3\,\%$, respectively. Is worth mentioning that in simulations C-RRT* outperformed LC-RRT* by a close margin, a result that was inverted in the experimental tests. Average percentage decreases of the cost C in the simulations for L-RRT*, LC-RRT*, and C-RRT* are $30.9\,\%$, $65.9\,\%$, and $66.7\,\%$, respectively. Meanwhile, the decreases observed for the field experiments are on average $17.7\,\%$, $40.9\,\%$, and $37.3\,\%$, respectively. In spite of the differences between the simulations and experiments, the proposed algorithms achieved the best performance in both cases and notably C-RRT* yields a trajectory that can be executed in shorter time at almost constant velocity.

The difference between simulations and experiments is explained mainly due to measurement delays in the real sensors, actuator delays and the fact that the full dynamics of the skid-steer machine is more difficult to accurately describe, especially due to varying traction on uneven terrain with changes between compacted soil and loose gravel. An effort was made to replicate sensor and actuator noise and delays in the simulations, but the stochastic wheel-terrain interaction dynamics was neglected as the optimal control TPBVP was solved as a deterministic problem for practical reasons. The existence of delays and inevitable model uncertainty, which explain some of the differences between simulations and experiments, can be observed in the difference between the reference and measured velocities shown in Figure 6.1(B)(E)(H)(K) and Figure 6.2(B)(E)(H)(K).



(A) Cumulative cost for scenario 2



(B) Cumulative cost for scenario 3

FIGURE 6.3. Cumulative cost vs time. It shows the evolution of the path tracker over time and highlights the differences between the tested methods. The time is normalized to simplify the comparison.

Finally, the total cumulative cost $C = C_{\mathrm{x,y}} + C_{\mathrm{v},\omega}$ versus time for each of the algorithms are presented in Figure 6.3A and Figure 6.3B for the first and second field experiment corresponding to simulation scenarios 2 and 3, respectively. Since each trajectory takes a different amount of time to complete, the time here was normalized in order to simplify the comparison between algorithms. The lowest cost is achieved by LC-RRT*, closely followed by C-RRT* in both experiments. L-RRT* follows with a higher cost and RRT* has the highest cost. The cumulative cost curves can be misleading as they do not reflect the fact that C-RRT* yields trajectories that can be completed in shortest time, as mentioned earlier. Thus C-RRT* is the most convenient approach despite having a cost C slightly higher than LC-RRT* in practice. These results also show that combining RRT* with an optimal control solution between intermediate points of the path can provide a good practical solution that to overcome the fact that the plain RRT* is asymptotically optimal, requiring an infinite time to converge, and that it does not consider the feasibility of the path with respect to the motion dynamics of the robot. Therefore, LC-RRT* and C-RRT*, provide practical alternatives that yield significant improvements in the quality of the path considering the ability of the robot to properly follow the trajectory.

7. CONCLUSION

7.1. Lessons Learned and Recommendations

The recommendations and lessons learned during the development and testing of the proposed motion planning strategies can be summarized in:

(i) Combining a spatial sampling approach like RRT* with a strategy to simplify the path and solve a finite number of two-point boundary optimal control problems is possible in practice with the standard computational power of current CPUs. The proposed LC-RRT* and C-RRT* approaches allow to generate feasible trajectories that comply with the motion model of the robot and simultaneously obtain the control signals. Thus the resulting trajectory can be executed without the oscillatory behaviour nor the overshoots induced by RRT* or L-RRT*.

(ii) Initially, LC-RRT* was developed believing that the solution of shorter TPBVPs located at the corners of a polygonal path would be faster than solving only TPBVPs piecewise between points of the polygonal trajectory. In simulation and in practice, the results showed that processing time was not affected significantly by the choice of strategy. The observation that LC-RRT* was producing trajectories that would take more time to execute than RRT* and L-RRT*, motivated the development of C-RRT*, which produced nice control signals in the sense that controls have less variations. In several cases the manipulated variable for the longitudinal velocity was kept almost constant and only gradual changes in the manipulated variable for the angular rate (trapezoidal transitions) were performed. Because of this, trajectories generated by C-RRT* can be executed faster than those generated by the other approaches. The limitation of LC-RRT* is in that is causes the longitudinal velocity to drop every time the straight motion switches into a curve, and both the manipulated variables for the longitudinal and angular velocities become a combination of trapezoidal signals. During the development, having trapezoidal shaped control inputs did not seem terrible, as many robots use this type of velocity profiles. However, solving optimal control problems piecewise grouping three consecutive points of the polygonal trajectory revealed that the longitudinal velocity should be kept as constant as possible and only changes in heading should be executed. In retrospect, this seems an obvious way of driving the robot to its goal considering its dynamics is similar to that of a car with Ackermann steering. Certainly, the motion dynamics imposes optimal ways of steering. If the robot would not be underactuated and could be capable of instantaneously producing a velocity in any direction of its configuration space like a holonomic omnidirectional robot, then other control trajectories could arise. This aspect possibly deserves further investigation, and was not treated here because the focus was on industrial skid-steer excavators and similar machines.

(iii) A practical issue that has to be considered in the implementation of all the motion planning strategies is the separation of points in the trajectory and how fast the reference trajectory is passed to the controller. Finely sampled spaces increases the computational burden and do not provide any gains if the resolution of the navigation sensors is comparatively low. On the other hand, limitations in the velocity of the robot, especially of slow heavy machinery, imply that the planner and controller have to be synchronized in some way that ensures that the planner waits for the controller to complete part of the motion before sending a new reference point of the trajectory. See (Auat Cheein, 2015) for a in depth discussion about this issue.

(iv) We also tested a strategy in which the two-point boundary value problem (TP-BVP) solver was embedded into the RRT* algorithm in an attempt to consider also the costs of traversing terrains with different terramechanical properties, but this approach of generating a trajectory and obtaining the control inputs proved to be impractical for operation in real-time due to the great computational burden.

7.2. Review of the Results and General Remarks

Two motion planning strategies, LC-RRT* and C-RRT*, were proposed in this paper. The proposed approaches combine the sampling-based path planning strategy of RRT* with the solution of an optimal control problem that takes into account the motion model of the robot, thus yielding kinodynamically feasible trajectories and the corresponding control inputs. The novel contribution can be summarized in that the proposed approaches provide a way to simplify the RRT* path and solve piecewise over a finite number of discrete intervals a two-point boundary optimal control problem. The proposed strategy is computationally feasible and can be implemented for operation in real-time, as it does not require to wait a large number of iterations for RRT* to converge to a smoother path, nor to solve a large optimal control problem with constraints at once.

LC-RRT* and C-RRT* yield trajectories that can be executed more accurately by the trajectory tracking controller. As a consequence, this approaches ensure that the trajectory will avoid maneuvers that can cause collisions that cannot be foreseen by RRT* or L-RRT*. Moreover, C-RRT* produces motion plans that reduce the variability of the manipulated variables. In the simulations and experiments, the longitudinal velocity was kept almost constant and variations in the angular velocity occurred gradually. This in turn means that the trajectory generated by C-RRT* can be accomplished in less time than with the other approaches.

The proposed strategies were first evaluated in simulations and then tested in field experiments using a robotic skid-steer excavator implemented by modifying a Caterpillar CAT 262C2 compact loader. The results obtained in simulations and the experiments are consistent. In both cases, the proposed approaches LC-RRT* and C-RRT* reduce the cost index composed of the tracking error and controller effort between $30-70\,\%$ with respect to RRT* because the paths are smoother and comply with the motion model of the robot. The major benefits of using the proposed approaches can be obtained in situations that involve narrow passages with tight turns. The results also show that C-RRT* can reduce by $10-15\,\%$ the time required to execute the planned trajectory.

The approaches were developed considering the maneuvering challenges of underground mining, but can also be employed for navigation in agricultural environments that have obstacles, like tree groves. In these applications, the planning of feasible collision free trajectories that can be executed rapidly is very important for safety and economic reasons.

References

Aguilera, S., Torres-Torriti, M., & Auat, F. (2014, Sept). Modeling of skid-steer mobile manipulators using spatial vector algebra and experimental validation with a compact loader. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on* (p. 1649-1655). doi: 10.1109/IROS.2014.6942776

Aguilera, S., Torres-Torriti, M., & Auat, F. (2015, Dec.). General dynamic model for skid-steer mobile manipulators with wheel-ground interactions. *Submitted to IEEE Trans. on Mechatronics*, 1-8.

Ardiyanto, I., & Miura, J. (2011). Heuristically arrival time field-biased (heat) random tree: An online path planning algorithm for mobile robot considering kinodynamic constraints. In *ROBIO* (p. 360-365). IEEE. Retrieved from http://dblp.uni-trier.de/db/conf/robio/robio2011.html#ArdiyantoM11

Auat Cheein, F. (2015). Intelligent sampling technique for path tracking controllers. *Control Systems Technology, IEEE Transactions on*, *PP*(99), 1-1. doi: 10.1109/TCST.2015.2450180

Barraquand, J., Langlois, B., & Latombe, J.-C. (1991, June). Numerical potential field techniques for robot path planning. In *Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on* (p. 1012-1017 vol.2). doi: 10.1109/ICAR.1991.240539

Canny, J. (1985, Mar). A Voronoi method for the piano-movers problem. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on* (Vol. 2, p. 530-535). doi: 10.1109/ROBOT.1985.1087297

Cheein, F. A., & Scaglia, G. (2014). Trajectory tracking controller design for unmanned vehicles: A new methodology. *Journal of Field Robotics*, *31*(6), 861–887.

Retrieved from http://dx.doi.org/10.1002/rob.21492 doi: 10.1002/rob.21492

De Oliveira Vaz, D. A. B., Inoue, R. S., & Grassi, V. (2010). Kinodynamic motion planning of a skid-steering mobile robot using RRTs. *Proceedings - 2010 Latin American Robotics Symposium and Intelligent Robotics Meeting, LARS 2010*, 73–78. doi: 10.1109/LARS.2010.27

Donald, B., Xavier, P., Canny, J., & Reif, J. (1993, November). Kinodynamic motion planning. *J. ACM*, *40*(5), 1048–1066. Retrieved from http://doi.acm.org/10.1145/174147.174150 doi: 10.1145/174147.174150

Dubins, L. E. (1957). On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, *79*(3), pp. 497-516. Retrieved from http://www.jstor.org/stable/2372560

Elbanhawi, M., & Simic, M. (2014). Sampling-based robot motion planning: A review. *IEEE Access*, *2*, 56-77. Retrieved from http://dblp.uni-trier.de/db/journals/access/access2.html#ElbanhawiS14

Gammell, J. D., Srinivasa, S. S., & Barfoot, T. D. (2014a). BIT*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs. *CoRR*, *abs/1405.5848*. Retrieved from http://arxiv.org/abs/1405.5848

Gammell, J. D., Srinivasa, S. S., & Barfoot, T. D. (2014b). Informed RRT*: Optimal incremental path planning focused through an admissible ellipsoidal heuristic. *CoRR*, *abs/1404.2334*. Retrieved from http://arxiv.org/abs/1404.2334

Ha, J.-S., Lee, J.-J., & Choi, H.-L. (2013). A successive approximation-based approach for optimal kinodynamic motion planning with nonlinear differential constraints. In *CDC* (p. 3623-3628). IEEE. Retrieved from http://dblp.uni-trier.de/db/conf/cdc/cdc2013.html#HaLC13

Hart, P., Nilsson, N., & Raphael, B. (1968, July). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, *4*(2), 100-107. doi: 10.1109/TSSC.1968.300136

Houska, B., Ferreau, H., & Diehl, M. (2011). ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, *32*(3), 298–312.

Islam, F., Nasir, J., Malik, U., Ayaz, Y., & Hasan, O. (2012, Aug). RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution. In *Mechatronics and Automation (icma), 2012 International Conference on* (p. 1651-1656). doi: 10.1109/ICMA.2012.6284384

Janson, L., Schmerling, E., Clark, A., & Pavone, M. (2015). Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research*, *34*(7), 883-921. Retrieved from http://ijr.sagepub.com/content/34/7/883.abstract doi: 10.1177/0278364915577958

Jolly, K. G., Sreerama Kumar, R., & Vijayakumar, R. (2009, January). A bezier curve based path planning in a multi-agent robot soccer system without violating the acceleration limits. *Robot. Auton. Syst.*, *57*(1), 23–33. Retrieved from http://dx.doi.org/10.1016/j.robot.2008.03.009 doi: 10.1016/j.robot.2008.03.009

Kanayama, Y. J., & Hartman, B. I. (1997, June). Smooth local-path planning for autonomous vehicles. *Int. J. Rob. Res.*, *16*(3), 263–284. Retrieved from http://dx.doi.org/10.1177/027836499701600301 doi: 10.1177/027836499701600301

Karaman, S., & Frazzoli, E. (2010a, June). Incremental sampling-based algorithms for optimal motion planning. In *Proceedings of Robotics: Science and Systems.* Zaragoza, Spain.

Karaman, S., & Frazzoli, E. (2010b, Dec). Optimal kinodynamic motion planning using incremental sampling-based methods. In *Decision and Control (CDC), 2010 49th IEEE Conference on* (p. 7681-7687). doi: 10.1109/CDC.2010.5717430

Karaman, S., & Frazzoli, E. (2011, June). Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, *30*(7), 846–894. Retrieved from http://dx.doi.org/10.1177/0278364911406761 doi: 10.1177/0278364911406761

Karaman, S., & Frazzoli, E. (2013). Sampling-based optimal motion planning for non-holonomic dynamical systems. In *IEEE International Conference on Robotics and Automation.* Karlsruhe, Germany. Retrieved from http://ares.lids.mit.edu/papers/Karaman.Frazzoli.ICRA13.pdf

Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., & Teller, S. J. (2011). Anytime motion planning using the RRT. In *ICRA* (p. 1478-1483). IEEE. Retrieved from http://dblp.uni-trier.de/db/conf/icra/icra2011.html#KaramanWPFT11

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

Khatib, O. (1985, Mar). Real-time obstacle avoidance for manipulators and mobile robots. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on* (Vol. 2, p. 500-505). doi: 10.1109/ROBOT.1985.1087247

Kunchev, V., Jain, L., Ivancevic, V., & Finn, A. (2006). Path planning and obstacle avoidance for autonomous mobile robots: A review. In B. Gabrys, R. Howlett, & L. Jain (Eds.), *Knowledge-Based Intelligent Information and Engineering Systems* (Vol. 4252, p. 537-544). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/11893004_70 doi: 10.1007/11893004_70

Latombe, J.-C. (1999). Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, *18*, 1119–1128.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge, U.K.: Cambridge University Press. (Available at http://planning.cs.uiuc.edu/)

LaValle, S. M., & Kuffner, J. J. (1999). Randomized kinodynamic planning. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on* (Vol. 1, p. 473-479 vol.1). doi: 10.1109/ROBOT.1999.770022

Li, Y., Littlefield, Z., & Bekris, K. E. (2014). Asymptotically optimal sampling-based kinodynamic planning. *CoRR*, *abs/1407.2896*. Retrieved from http://arxiv.org/abs/1407.2896

Lozano-Pérez, T., & Wesley, M. A. (1979, October). An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, *22*(10), 560–570. Retrieved from http://doi.acm.org/10.1145/359156.359164 doi: 10.1145/359156.359164

Maekawa, T., Noda, T., Tamura, S., Ozaki, T., & Machida, K.-i. (2010, April). Curvature continuous path generation for autonomous vehicle using b-spline curves. *Comput. Aided Des.*, *42*(4), 350–359. Retrieved from http://dx.doi.org/10.1016/j.cad.2009.12.007 doi: 10.1016/j.cad.2009.12.007

Noble, F. (2011-2014). *SBP: Swift Navigation Binary Protocol.* https://github.com/swift-nav/libsbp.

Seward, D. W., Pace, C., & Agate, R. (2007). Safe and effective navigation of autonomous robots in hazardous environments. *Auton. Robots*, *22*(3), 223-242. Retrieved from http://dblp.uni-trier.de/db/journals/arobots/arobots22.html#SewardPA07

Shan, E., Dai, B., Song, J., & Sun, Z. (2009, Dec). A dynamic RRT path planning algorithm based on B-spline. In *Computational Intelligence and Design, 2009. ISCID '09. Second International Symposium on* (Vol. 2, p. 25-29). doi: 10.1109/ISCID.2009.155

Stenning, B. E., McManus, C., & Barfoot, T. (2013). Planning using a network of reusable paths: A physical embodiment of a rapidly exploring random tree. *Journal of Field Robotics*, *30*(6), 916–950. Retrieved from http://dx.doi.org/10.1002/rob.21474 doi: 10.1002/rob.21474

Stentz, A. T. (1994, May). Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '94)* (Vol. 4, p. 3310 - 3317).

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., ... Mahoney, P. (2006, September). Stanley: The robot that won the darpa grand challenge: Research articles. *J. Robot. Syst.*, *23*(9), 661–692. Retrieved from http://dx.doi.org/10.1002/rob.v23:9 doi: 10.1002/rob.v23:9

Webb, D. J., & van den Berg, J. (2012). Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints. *CoRR*, *abs/1205.5088*. Retrieved from http://arxiv.org/abs/1205.5088

Xie, C., van den Berg, J. P., Patil, S., & Abbeel, P. (2015). Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015* (pp. 4187–4194). Retrieved from http://dx.doi.org/10.1109/ICRA.2015.7139776 doi: 10.1109/ICRA.2015.7139776

Yang, K., Moon, S., Yoo, S., Kang, J., Doh, N., Kim, H., & Joo, S. (2014). Spline-based RRT path planner for non-holonomic robots. *Journal of Intelligent & Robotic Systems*, *73*(1-4), 763-782. Retrieved from http://dx.doi.org/10.1007/s10846-013-9963-y doi: 10.1007/s10846-013-9963-y

Yang, K., & Sukkarieh, S. (2010, June). An analytical continuous-curvature path-smoothing algorithm. *Trans. Rob.*, *26*(3), 561–568. Retrieved from http://dx.doi.org/10.1109/TRO.2010.2042990 doi: 10.1109/TRO.2010.2042990

APPENDIXES

APPENDIX A. ADDITIONAL RESOURCES

A.1. Notation

X	Set of state space
$x(t)$	state, belongs to X
$\mathcal{X}$	Set of state space trajectories
x	trajectory, belongs to $\mathcal{X}$
U	Set of allowed controls
$u(t)$	control, belongs to U
$\mathcal{U}$	Set of control trajectories
u	control trajectory, belongs to $\mathcal{U}$
$\mathcal{C}$	Configuration Space
X_{free}	Free space
X_{obs}	Obstacle region, $X_{\text{obs}} = X \setminus X_{\text{free}}$
X_{goal}	Goal region, $X_{\text{goal}} \subset X_{\text{free}}$
V	Vertex structure
E	Edge structure
Q	Set of vertices
q_i	Vertex
x_i	Trajectory between vertices
L	Path formed by vertices q_i
Σ	Kinodynamic model of the robot
q_i	State $[\mathrm{x}_i\ \mathrm{y}_i\ \theta_i]$
x	Position in the X axis
y	Position in the Y axis
θ	Orientation with respect to the X axis
v	Longitudinal velocity
ω	Angular velocity

$C_{\mathrm{x,y}}$	Total error cost, $\sum\limits_{k=0} (\mathrm{x}_{\mathrm{ref},k} - \mathrm{x}_k)^2 + (\mathrm{y}_{\mathrm{ref},k} - \mathrm{y}_k)^2$
$C_{\mathrm{v},\omega}$	Effort of the control law, $\sum\limits_{k=0} \mathrm{v}_k^2 + \omega_k^2$
C	Total cost, $C_{\mathrm{x,y}} + C_{\mathrm{v},\omega}$

APPENDIX B. ROBOT COMMUNICATION PROTOCOL

B.1. Design Philosophy

The *Robust Real-Time Robot Communication Protocol* (RCP) is the communication protocol for RAL's robots. The main features of RCP are:

(i) Simplicity: Simple protocol structure.

(ii) Real-Time Operation: Light-weight messages with adjustable time-out settings.

(iii) Robustness: Error detection through cyclic-redundancy codes and watchdog timers.

(iv) Modularity: Settable parameters and open architecture for additional user-defined functions.

The design of RCP is inspired in the SwiftNav Binary Protocol (SBP) (Noble, 2011-2014), developed by Swift Navigation Inc. for the communication of the Piksi GPS devices.

B.2. Message Structure

The RCP message consists of a 4 byte header section, a variable size data field, and a 16-bit CRC value[1]. Table B.1 summarizes the message structure.

B.3. Message List

The complete message list to date can be found in the following tables. The messages are divided into 2 categories: messages going from the PC to the MasterBoard R3, Table B.2, and messages going from the MasterBoard R3 to the PC, Table B.3. For every message from the PC there will be a corresponding response message.

Messages with EVEN ID are messages going from the PC to the MasterBoard R3.

Messages with ODD ID are messages going from the MasterBoard R3 to the PC.

[1]CCITT 16-bit CRC implementation is used. More implementation details can be found in the source code.

TABLE B.1. RCP message structure

Offset (bytes)	Size (bytes)	Name	Description
0	2	Start	Denotes the start of the message. First byte = `0x55`. Second byte = `0x33`
2	1	Message Type	Identifies the data contents.
3	1	Length	Data contents length in bytes. The length is a number $N \in [0, 255]$.
4	N	Data	Binary message contents of length N.
4+N	2	CRC	Cyclic redundancy check of the data from the *Message Type* up to the end of the *Data* (does not include the message *Start* bytes).
	$N + 8$		Total message length.

TABLE B.2. Message Types

Name	ID	Size (bytes)	Description
`RCP_HEARTBEAT`	`0x12`	0	System heartbeat message
`RCP_SET_PWM`	`0x22`	4	Set control signals
`RCP_GET_ENC`	`0x42`	0	Get encoder's readings
`RCP_SET_PWM_GET_ENC`	`0x62`	4	Set control signals and get encoder's readings with the same message
`RCP_SET_PARAMETERS`	`0xf2`	5	Set parameters of the system
`RCP_GET_PARAMETERS`	`0xf4`	1	Get parameters of the system

B.3.1. `RCP_MSG_LOG`

This message contains a human-readable payload string from the device containing errors, warnings and informational messages.

B.3.2. `RCP_HEARTBEAT` and `RCP_HEARTBEAT_R`

The heartbeat message is used to check if the system is still running. The response message contains the program running time in milliseconds.

TABLE B.3. Response Messages Types

Name	ID	Size (bytes)	Description
RCP_MSG_LOG	0x11	N	Plaintext logging messages
RCP_HEARTBEAT_R	0x13	4	System heartbeat message
RCP_SET_PWM_R	0x23	4	Set control signals
RCP_GET_ENC_R	0x43	8	Get encoder's readings
RCP_SET_PWM_GET_ENC_R	0x63	8	Set control signals and get encoder's readings with the same message
RCP_SET_PARAMETERS_R	0xf3	N	Set parameters of the system
RCP_GET_PARAMETERS_R	0xf5	N	Get parameters of the system

TABLE B.4. `RCP_MSG_LOG - 0x11` response message structure

Offset (bytes)	Size (bytes)	Format	Units	Name	Description
0	N	string		`text`	Human-readable message
	N				Total message length.

TABLE B.5. `RCP_HEARTBEAT - 0x12` message structure

Offset (bytes)	Size (bytes)	Format	Units	Name	Description
	0				Total message length.

TABLE B.6. `RCP_HEARTBEAT_R - 0x13` response message structure

Offset (bytes)	Size (bytes)	Format	Units	Name	Description
0	4	u32	ms	`info`	System time
	4				Total message length.

B.3.3. `RCP_SET_PWM` and `RCP_SET_PWM_R`

Set the duty cycle of the 4 PWMs that control the movement of the CAT. Each value must be in the range $20\% - 80\%$ of $[0, 255]$ to be a valid, i.e. $[52, 203]$. Conversion of bytes to duty cycles is straight forward following the Arduino conveniton $0 = 0\%$ duty cycle, $255 = 100\%$ duty cycle. Beware that the R3 robot only considers duty cycles values between 20% to 80% to be valid. If the user sends a duty cycle value out of this range, the robot automatically stops.

The response message contains the values to which the PWMs were set.

TABLE B.7. `RCP_SET_PWM - 0x22` message structure

Offset (bytes)	Size (bytes)	Format	Units	Name	Description
0	1	u8		`pmw1`	Forward/backward control signal
1	1	u8		`pmw2`	Left/right control signal
2	1	u8		`pmw3`	Arm up/down control signal
3	1	u8		`pmw4`	Bucket up/down control signal
	4				Total message length.

TABLE B.8. `RCP_SET_PWM_R - 0x23` response message structure

Offset (bytes)	Size (bytes)	Format	Units	Name	Description
0	1	u8		`pmw1`	Forward/backward control signal
1	1	u8		`pmw2`	Left/right control signal
2	1	u8		`pmw3`	Arm up/down control signal
3	1	u8		`pmw4`	Bucket up/down control signal
	4				Total message length.

B.3.4. `RCP_GET_ENC` and `RCP_GET_ENC_R`

Get the readings of both encoders. The response message contains 2 floats with the values. Use the function `rcp_get_enc_callback()` to cast these values to a `msg_get_enc_t` struct.

B.3.5. `RCP_SET_PWM_GET_ENC` and `RCP_SET_PWM_GET_ENC_R`

Combines the `RCP_SET_PWM` message with the response message `RCP_GET_ENC_R`.

TABLE B.9. `RCP_GET_ENC - 0x42` message structure

Offset (bytes)	Size (bytes)	Format	Units	Name	Description
	0				Total message length.

TABLE B.10. `RCP_GET_ENC_R - 0x43` response message structure

Offset (bytes)	Size (bytes)	Format	Units	Name	Description
0	4	float	$\mathrm{rad\,s^{-1}}$	`wl`	Left side wheels speed
4	4	float	$\mathrm{rad\,s^{-1}}$	`wr`	Right side wheels speed
	8				Total message length.

B.3.6. `RCP_SET_PARAMETERS` and `RCP_SET_PARAMETERS_R`

Set one of the parameters of the device to a desired value. To date the available system parameters are:

(i) `PARAM_TIMEOUT`: Max allowed time without messages before stopping the machine. Initially set to $100\,\mathrm{ms}$.
(ii) `PARAM_PWM`: Sets the response for the `RCP_SET_PWM` message, 1 for response and 0 for no response. Initially set to 1.
(iii) `PARAM_ENC`: Time interval between encoder's readings. Not implemented.
(iv) `PARAM_HEARTBEAT`: Time interval between hearbeat messages. Not implemented.

TABLE B.11. `RCP_SET_PARAMETERS - 0xf2` message structure

Offset (bytes)	Size (bytes)	Format	Units	Name	Description
0	1	u8		`id`	Parameter ID
1	4	u32		`value`	Value to change, it can be type casted
	5				Total message length.

TABLE B.12. `RCP_SET_PARAMETERS_R - 0xf3` response message structure

Offset (bytes)	Size (bytes)	Format	Units	Name	Description
0	N	string		`text`	Human-readable message
	N				Total message length.

B.3.7. `RCP_GET_PARAMETERS` and `RCP_GET_PARAMETERS_R`

Get one of the device parameters in a human-readable string.

TABLE B.13. `RCP_GET_PARAMETERS - 0xf4` message structure

Offset (bytes)	Size (bytes)	Format	Units	Name	Description
0	1	u8		`id`	Parameter ID
	1				Total message length.

TABLE B.14. `RCP_GET_PARAMETERS_R - 0xf5` response message structure

Offset (bytes)	Size (bytes)	Format	Units	Name	Description
0	N	string		`text`	Human-readable message
	N				Total message length.