INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Igor Franzoni Okuyama

TRAJECTORY PLANNING CONSIDERING ACCELERATION LIMITS FOR AN AUTONOMOUS SOCCER PLAYER

Final Paper
2017

Course of Electronics Engineering

CDU 621.38

Igor Franzoni Okuyama

TRAJECTORY PLANNING CONSIDERING ACCELERATION LIMITS FOR AN AUTONOMOUS SOCCER PLAYER

Advisor

Prof. Dr. Rubens Junqueira Magalhães Afonso (ITA)

Co-advisor

Prof. Marcos Ricardo Omena de Albuquerque Máximo (ITA)

ELECTRONICS ENGINEERING

São José dos Campos
Instituto Tecnológico de Aeronáutica

2017

Cataloging-in Publication Data
Documentation and Information Division

Franzoni Okuyama, Igor
Trajectory Planning Considering Acceleration Limits for an Autonomous Soccer Player / Igor Franzoni Okuyama.
São José dos Campos, 2017.
52f.

Final paper (Undergraduation study) – Course of Electronics Engineering– Instituto Tecnológico de Aeronáutica, 2017. Advisor: Prof. Dr. Rubens Junqueira Magalhães Afonso. Co-advisor: Prof. Marcos Ricardo Omena de Albuquerque Máximo.

1. Trajectory Planning. 2. Dynamic Systems. 3. Robotics. I. Instituto Tecnológico de Aeronáutica. II. Title.

BIBLIOGRAPHIC REFERENCE

FRANZONI OKUYAMA, Igor. **Trajectory Planning Considering Acceleration Limits for an Autonomous Soccer Player**. 2017. 52f. Final paper (Undergraduation study) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

Igor Franzoni Okuyama
R. H10E, 102
12228-700 – São José dos Campos - SP

TRAJECTORY PLANNING CONSIDERING ACCELERATION LIMITS FOR AN AUTONOMOUS SOCCER PLAYER

This publication was accepted like Final Work of Undergraduation Study

---

Igor Franzoni Okuyama

Author

Rubens Junqueira Magalhães Afonso (ITA)

Advisor

Marcos Ricardo Omena de Albuquerque Máximo (ITA)

Co-advisor

Prof. Dr. Cairo Nascimento
Course Coordinator of Electronics Engineering

São José dos Campos: November 17, 2017.

To God and all fellow friends that helped me in this work and to my family.

Acknowledgments

To ITA and all its professors for all the knowledge and experiences that helped me conclude this work and also for financial support through a scholarship.

To ITAndroids and all its members for both intellectual and equipment support.

To my advisor Prof. Dr. Rubens Junqueira Magalhães Afonso and my co-advisor Prof. Marcos Ricardo Omena de Albuquerque Máximo for the discussions and guidance that were essential for the development of this work.

Resumo

Futebol de robôs é uma competição de robôs autônomos com regras similares ao jogo com humanos, isto é, ganha quem fizer mais gols. Dessa forma, a competitividade de um time está diretamente ligada à habilidade de seus componentes. No âmbito de robôs autônomos, a habilidade está ligada aos movimentos gerados por um algoritmo de estratégia e pela capacidade de executar fielmente os comandos advindos dela. Assim, a velocidade máxima de cada jogador é essencial para definir uma estratégia mais eficiente, pois isso contribui para uma maior posse de bola e ataques mais agressivos. Contudo, ao exigir velocidades maiores dos robôs, efeitos dinâmicos como limites de aceleração devem ser levados em conta tanto no planejamento dos movimentos, quanto da execução dos mesmos. Este trabalho visa estudar a inclusão de características dinâmicas do robô durante a fase de planejamento de trajetórias e as possíveis vantagens em relação ao método clássico de utilizar apenas as características cinemáticas do sistema. Experimentos são feitos tanto em simulação, quanto em testes nos robôs reais da equipe de robótica ITAndroids, de forma a demonstrar da viabilidade da técnica proposta.

Abstract

Robot soccer is a competition of autonomous soccer robots with rules similar to the human soccer, or in other words, wins the team that scores more goals. Then, the team competitiveness is directly related to the ability of each player. In the case of autonomous robots, the ability has to do with them movements generated by a strategy algorithm e with the capacity of executing accurately the commands coming from it. Therefore, the maximum velocity of each player is essential to define a more efficient strategy, as it contributes to a better ball possession and more aggressive attacks. However, when more speed is required from the robots, dynamic effect like acceleration limits must be taken into account both in the trajectory planning and in the plan execution. This work studies the inclusion of the robot's dynamics in the trajectory planning and its possible advantages compared to the classical method of considering only the kinematic characteristics of the system. Experiments are done through simulations and tests in the real robot soccer system of the ITAndroids robotics group, in order to demonstrate the viability of the proposed technique.

List of Figures

List of Tables

Contents

1 Introduction

1.1 Motivation

Differential Drive Mobile Robots (DDMR) are a common type of robot used in many practical applications from industrial transportation to scientific research. Its popularity comes from its simple design and manufacturing, although controlling it might be harder than controlling omnidirectional robots, since it cannot move in any direction instantaneously. In addition, if it is demanded to move quickly around a field, while avoiding obstacles, a model for the system will be needed in order to design high performance controllers.

The concepts presented here, though valid for a general DDMR, are applied in the context of robot soccer, more specifically in the IEEE Very Small Size (VSS) League competition (COMPETITION, 2008) hosted at Latin American Robotics Competition (LARC). It is a soccer game consisting of 3 autonomous differential-drive robots per team. The players have a size restriction of 7.5 cm per side and have identification color on its top side. A camera placed above the field sends a video at 60 Hz to a computer, that has a vision processing algorithm. This algorithm is able to estimate the position and orientation of each robot and a game strategy program can use this to create commands to the players. The strategy might consist of many abstraction layers like: determination of the role of each player (attacker, goal keeper or defensor); definition of the desired position of each player; find a trajectory without collisions from each start state to the desired one; and finally compute the desired velocity for the agents. These last 2 abstraction layers are the main scope of this work.

Additionally, one major problem of soccer robot is the necessity of running algorithms in real time, i.e., faster than the camera capture frequency. Therefore, the possible solutions need to run quite fast, what limits the choice of algorithms. Our robotics group, ITAndroids (from Instituto Tecnológico de Aeronáutica), participates in the competition since 2014 and our robot and a game overview can be found in Fig. 1.1.

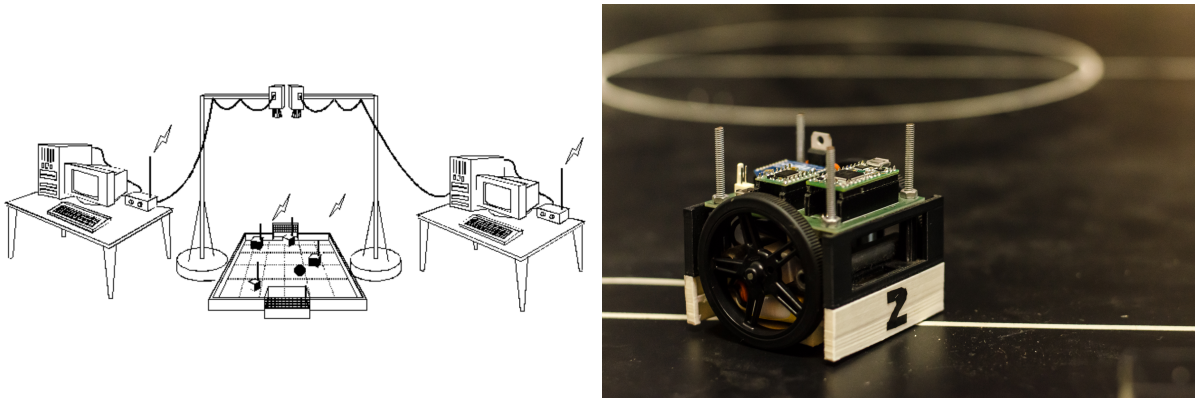


FIGURE 1.1 – To the left: the VSS game. To the right: our robot used in the 2016 competition.

1.2 Literature Review

In this chapter, a variety of algorithms that were proposed to solve the trajectory planning problem are revised, pointing out their advantages and possible disadvantages in our specific system, the DDMR.

1.2.1 Path Planning

Path planners differ from trajectory planners in the sense that they disregard the dynamic equations of the system (2.1). Here, we are just worried about finding a set of poses (positions and possibly orientation) that connect the start and goal states. After a path is computed, another algorithm must be used to make the robot actually follow the path. Therefore, the true executed trajectory may hit obstacles, since the planning does not consider acceleration limits.

A class of such algorithms is known as grid-based search, where the space is subdivided in a grid, or more generally in a graph. Is is also only permitted to move through edges of the graph. A known algorithm that is able to find the minimum path between to points in the graph is called A* (HART *et al.*, 1968). Another interesting algorithm is the Dynamic Programming, which allows for inclusion of dynamics, and was used, for instance, to solve the double integrator minimum time problem (BELLMAN, 1956). The main problem of this idea is how to deal with high-dimensionality spaces, since the number of nodes will grow exponentially, and so will the convergence time.

Another noticeable algorithm is the Artificial Potential Fields that models objects as attractive or repulsive electric charges. It has been widely used in robotics, but it does not consider the dynamics of the system. Another disadvantage is that there are many parameters to tune and they usually do not have physical meaning. However, by

employing optimization techniques like evolutionary programming, (LIM *et al.*, 2008) was able to achieve good results. Nevertheless, intensively testing the real system to generate data for the optimization might be time consuming and not possible depending on the system.

1.2.2 Sampling-Based Methods

Sampling-based algorithms rely on building up a graph (or a tree) with random samples, instead of using a predefined grid. This solves the issue of high-dimensional tasks, since it is not necessary to store a huge grid anymore and it is not needed to think about the grid resolution. In the Probabilistic Roadmap (PRM) algorithm (KAVRAKI *et al.*, 1996), a graph is created by uniformly sampling the space and connecting each sample with the k-nearest samples. Than a graph algorithm, such as A*, might be used to find the optimal path between any two nodes in the graph. If the start or goal states are not in the graph, a simple idea is to connect them to the nearest node. It is proven that this algorithm is probabilistic complete (KAVRAKI *et al.*, 1998), i.e., if there is a feasible path, it will find it, when the number of samples approaches infinity.

PRM is called a multi-query planning, since creating a graph is equivalent to solving a path planning problem for multiple start and goal states. However, if only a path between a start and a goal state is desired, multi-query planners may be too expensive to apply. For these kind of problems, a single-query algorithm might lead to faster and simpler algorithms.

An example of single-query and sample-based method is the Rapidly-Exploring Random Trees (RRT) (LAVALLE, 1998). Instead of building a graph, it tries to build a tree that connects start and goal states. The procedure for creating the tree is similar to the PRM: the root node is the start state; a point is uniformly sampled from the search space and it is connected to the nearest node of the tree. Eventually, the goal will be connected to tree and a path will be found by recursively going up the tree (from the leaf to the root).

Although, RRT is also proven to be probabilistic complete, it can also be shown that it is not optimal, even if infinite points are sampled (KARAMAN; FRAZZOLI, 2011) (the same applies to PRM). However, the same paper proposes a probabilistic optimal algorithm by ”rewiring” the tree at every iteration.

Another advantage of RRT is its ability to handle dynamics like is demonstrated at (WEBB; BERG, 2013). Other studies, utilized RRT in the case of an urban car that has nonlinear dynamics (KUWATA *et al.*, 2009). This successfully was used in the 2007 DARPA Challenge showing its applications on complex, real world systems.

Many variations of the RRT were developed through time, in order to compensate some drawbacks, for instance, some heuristics are proposed in order to make the algorithm more greedy like making two trees grow towards each other (KUFFNER; LAVALLE, 2000). (URMSON; SIMMONS, 2003) tries to use a non-uniform sampling strategy, by weighting exploitation and exploration. In the case of the optimal version, it was shown by (GAMMELL *et al.*, 2014) that after an initial path is found, one only needs to take samples from an ellipsoidal subset (it is not necessary to sample the entire space). This makes the search considerably faster, mainly if the space is not cluttered.

On the other hand, we could list some disadvantages of RRT: the first is that the algorithm in probabilistic. This means that for the same environment, start and goal objective, it might lead to different paths. The second is its complexity in comparison to the path planning methods discussed above. Some ideas were proposed in (LINDEMANN; LAVALLE, 2004) and (JANSON *et al.*, 2015) to tackle the randomization issue by using deterministic sequences . For the complexity problem, improving nearest neighbor search by means of kd-trees is reported to drastically reduce the running time (YERSHOVA; LAVALLE, 2007).

Additionally, an important finding in the last decade, is the optimal motion planning algorithm, RRT*, by (KARAMAN; FRAZZOLI, 2011). Actually, (KARAMAN *et al.*, 2011) and (JEON *et al.*, 2011) emphasize the "anytime" characteristic of RRT*: after quickly finding an initial solution with RRT, one can employ RRT* to improve it until there is no free time left for the optimization. Also, it is possible to reuse the trees if replanning is desired. This means that the RRT* can produce better trajectories, while the path is being executed.

Because of the many advantages the RRT has over the the classical path planning algorithms, we choose to apply it for the robot soccer domain. Although it has been already done in other works like (BRUCE; VELOSO, 2002), we plan to apply the many optimizations developed since this paper was published.

1.3 Objective

This work objective is to design a trajectory planning algorithm for a differential drive mobile robot (DDMR) using the RRT. It should be able to produce dynamically feasible paths, i.e., the robot should not collide with obstacles and the trajectory must consider the robot's dynamic equations. A comparison between a planner that considers only a kinematic model will be made. We also study the effects of replanning, in order to increase system robustness to model errors.

1.4 Organization of this work

The paper is organized as follows: Ch. 2 describes in detail the problem we want to solve. Ch. 3 explains the algorithms chosen and some optimizations. Then, Ch. 4 presents the models used in the planning algorithm and for simulation and a pose controller used for robot stabilization. In Ch. 5, we describe how the model and the pose controller were included in the RRT formulation, while also driving comparisons between proposed planners and experiment results with the real robot. Ch. 6 finalizes the work with final considerations and future work.

2 Problem Description

2.1 Hypothesis

To better understand the available data that will be used during planning step, see Fig. 2.1. First, the camera is responsible for detecting the poses (position and orientation) of teammate robots and position of opponent robots on the field. It is not possible to directly detect the orientation of the opponents, since their color pattern is not known previously. Next, the camera information is filtered and robots' velocities are estimated. Then, the strategy computes the desired position for a robot, which is used together with the known position of each robot (that are considered as obstacles) by the trajectory planner. Besides generating trajectory, i.e., a set of poses indexed in time, the trajectory planner also generates the actions necessary to execute the trajectory. Then, a pose controller will pick the first action and calculate speed commands that will be sent to the robots via radio link.

2.2 Problem Statement

We define the trajectory planning problem similar to (JEON *et al.*, 2011). Given the set of states, $X \subset \mathbb{R}^n$, and the set of controls, $U \subset \mathbb{R}^m$, both compact sets with $n, m \in \mathbb{Z}$, and $\mathbf{x}(t) \in X$, $\mathbf{u}(t) \in U$, such that:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{2.1}$$

We emphasize that the compact set U encodes limits in the input of the system, so that it is possible to model saturation of actuators and acceleration limits, for instance.

We would like to find a trajectory, $\mathbf{x} : [0, T] \to X$, and the controls (or actions) , $\mathbf{u} : [0, T] \to U$, that connect a start state, $\mathbf{x}_{start} = \mathbf{x}(0)$, and a goal state, $\mathbf{x}_{goal} = \mathbf{x}(T)$. More generally, the goal could be defined as a set, X_{goal}, with $\mathbf{x}_{goal} \in X_{goal}$.

If there are obstacles, we should define the set of them to be X_{obs}. The set $X_{free} =$

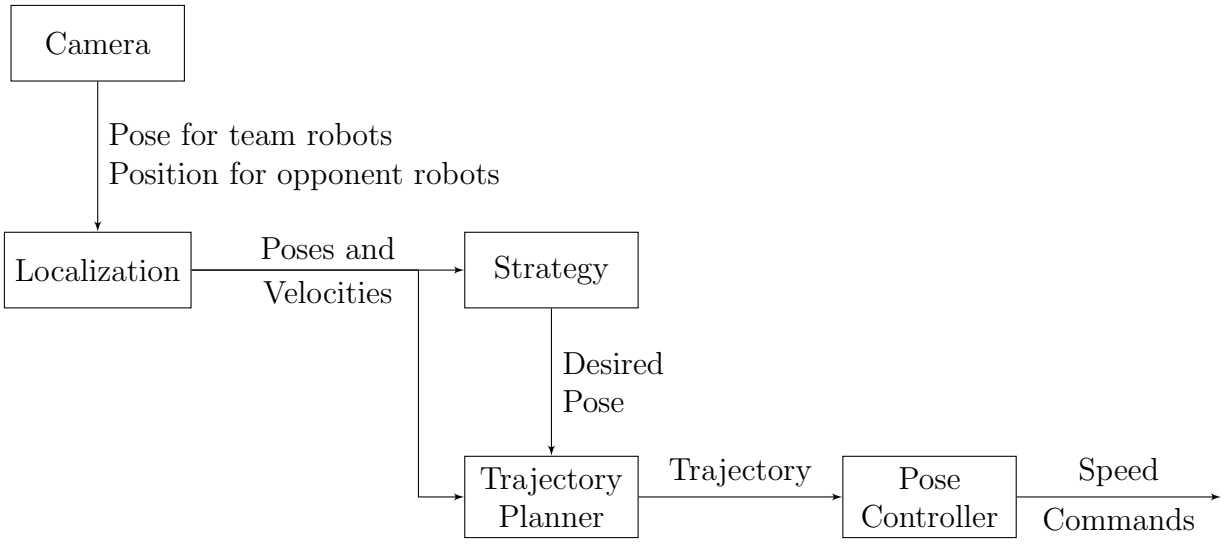


FIGURE 2.1 – VSS abstraction layers.

$X \backslash X_{obs}$ is the allowed region for the robot, and we must assure that $\mathbf{x}(t) \in X_{free}$. See Fig. 2.2.

The way the problem is defined is similar to a feasibility problem, i.e., it is only desired a feasible trajectory. If a cost function is to be minimized, we get a problem known as kinodynamic planning and there are no exact solutions for it according to (DONALD *et al.*, 1993). Even for omnidirectional robots in 2D or 3D space, there are only approximate solutions. Then, for nonlinear systems, such as the DDMR, it is more challenging to find solutions that lead to reasonable performance. Therefore, we will only try to solve the feasibility problem, although some heuristics will be applied so that the performance is not very suboptimal.

Notice that knowledge about the dynamics is necessary to generate a plan. Sec. 4.1 will present some models that will be compared. Nevertheless, there will by always disparity between models and reality and the accumulated errors add up in time. That is why we will also consider the possibility of replanning, so that the planner is robust to error models.

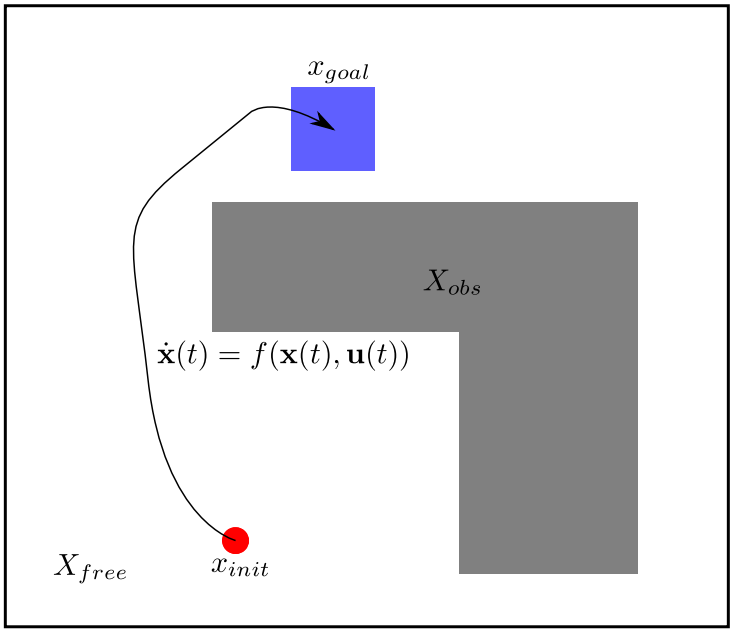


FIGURE 2.2 – Trajectory Planning Problem.

3 Rapidly-Exploring Random Trees (RRT)

In this chapter, we will explain the RRT algorithm and detail its functionality and possible heuristic optimizations. We might go through each main function and explain possible drawbacks of the original version and analyze what has been proposed to enhance it.

3.1 RRT

Alg. 1 shows the RRT pseudocode taken from its introductory paper (LAVALLE, 1998). As can be seen, the algorithm is quite simple and has just a few lines. The pseudocode as is depicted in Alg. 1 is only worried about growing a random tree from x_{init} for K iterations, although one could easily stop the growing when the tree reaches a desired goal. An example of the algorithm execution is depicted in Fig. 3.1. Next each function of the code will be further detailed.

Algorithm 1 RRT

1: **procedure** GENERATERRT($x_{init}, K, \Delta t$)
2: $\mathcal{T}.insertNode(x_{init}, \emptyset)$
3: **for** $k = 1$ to K **do**
4: $x_{rand} \leftarrow sampleSpace()$
5: $x_{nearest} \leftarrow nearestNeighbor(x_{rand}, \mathcal{T})$
6: $u \leftarrow selectInput(x_{nearest}, x_{rand})$
7: $x_{new} \leftarrow newState(x_{nearest}, u, \Delta t)$
8: $\mathcal{T}.insertNode(x_{new}, x_{nearest})$
9: **end for**
10: **return** $\mathcal{T}$
11: **end procedure**

3.1.1 Function $insertNode(x, P)$

Supposing a tree data structure like in Alg. 2, this function creates a new node with state x and a parent P.

Algorithm 2 RRT Tree Structure

```
1: Struct
2:     state;
3:     parent;
4: EndStruct
```

3.1.2 Function $sampleSpace()$

This function is responsible for sampling a point from space. In general, a uniform distribution is used as suggested by (LAVALLE, 1998), since it will make the tree grow uniformly. If the search space is defined by a hyperrectangle, then the sampling can be done as:

$$\begin{aligned} x_{rand} &= (U_1, U_2, \ldots, U_n) \\ U_i &\sim \mathcal{U}(u_i^{min}, u_i^{max}),\ i = 1, 2, \ldots, n \end{aligned} \tag{3.1}$$

where (u_i^{min}, u_i^{max}) is the range of the i-th dimension of X and $\mathcal{U}(u_i^{min}, u_i^{max})$ is the uniform distribution from u_i^{min} to u_i^{max}.

For example, for an omnidirectional 2D robot, we could have $U_1 \sim \mathcal{U}(-1, 1)$, $U_2 \sim \mathcal{U}(-1, 1)$, whilst for a differential robot, $U_1 \sim \mathcal{U}(-1, 1)$, $U_2 \sim \mathcal{U}(-1, 1)$, $U_3 \sim \mathcal{U}(0, 2\pi)$.

3.1.3 Function $nearestNeighbor(x_{rand}, \mathcal{T})$

The *Nearest Neighbor* method is responsible for finding the nearest node in the tree with respect to the sampled point, i.e., it should return:

$$x_{nearest} = \arg\min_{x \in \mathcal{T}} \|x - x_{rand}\| \tag{3.2}$$

We point out that a convenient distance metric is needed and depends on the search space. If $X = \mathbb{R}^n$, than the Euclidean distance is suitable. On the other hand, if rotations are considered, for instance, we might choose the circle space $S^1 = [0, 1]/0 \sim 1$, and a different metric should be used like $||p - q||_{S^1} = min(|p - q|, 1 - |p - q|)$.

Observe that this is usually the most costly step of the algorithm, since the tree grows at each step and it can easily have many thousands of nodes. Therefore, kd-trees are

usually chosen for nearest neighbor search as discussed in (YERSHOVA; LAVALLE, 2007). One drawback of kd-trees, on the other hand, is that it only handles Euclidean Distances, which is not a trivial metric for distances between angles, due to wrapping effects. This is easy to see, for example, considering the angles 1° and 359°. The Euclidean distance, would determine their distance to be 358°, but we desire it to be 2°. Then, we transform the 1-dimensional angle to a 2-dimensional circle that makes possible the use of Euclidean distances for angles:

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ \sin\theta \\ \cos\theta \end{bmatrix} \tag{3.3}$$

3.1.4 Function $selectInput(x_{nearest}, x_{rand})$

Also known as "extend" function, this method tries to generate an input for the system that drives it towards the sampled point. For a 2D omnidirectional robot, one only needs to generate a velocity vector that points towards the sampled state. For more complex systems, e.g. a differential robot, a simple way of doing it, is to discretize the set of possible inputs, U, simulate the system forward for time Δt and choose the input that drives the system closer to x_{rand}. Other alternatives are to use nonlinear control techniques like in (PALMIERI; ARRAS, 2014) or using closed-loop models (KUWATA *et al.*, 2009). Interesting extend functions for car-like robots are the ones that rely on optimal paths like the Dubins path (DUBINS, 1957), Reeds-Shepp curves (REEDS; SHEPP, 1990) or Balkcom-Mason curves (BALKCOM; MASON, 2002), although these path planners are only concerned with the kinematics of the robot. In Sec. 4.2, we describe a pose controller that we used as the extend function for the soccer robot.

3.1.5 Function $newState(x_{near}, u, \Delta t)$

The *New State* function returns the state of the system subject to the control input calculated by the *Select Input* procedure. For a dynamical system, this implies solving Eq. (2.1), i.e., simulating forward the system by Δt. This could be done through Euler, Runge-Kutta or linear multistep integration methods (e.g. Adams-Bashforth). In principle any nonlinear system could be used, or even a complex dynamics simulator capable of simulating active and passive objects can be used (ZICKLER; VELOSO, 2009). This is where the power of the RRT relies: if there is an appropriate extend function and a simulator available, then RRT can be used for solving trajectory planning problems. The simulation models used in this work are described in Sec. 4.1 and they use Euler

integration for simplicity.

3.1.6 Function *collisionChecking*()

This step is implicit in the *New State* method. If there is a collision during forward simulation of the system, then the x_{new} node should not be added to the tree, since it is not a valid node. More advanced techniques might be used like (MIRTICH, 1998). Another strategy is to create a drivability map (KUWATA *et al.*, 2009) that is essentially a grid whose cells can be occupied or non-occupied by an object. This can considerably speed up the collision checking step. In our case, we opted for a simpler collision checking algorithm: we choose a small enough Δt and only verify if the new state is inside any obstacles. This leads to fast execution suitable for real time planners, besides the convenience for nonlinear robots (specially the DDMR), since it can make curved paths.

3.1.7 Heuristic Optimizations

3.1.7.1 Goal biasing

Uniform sampling the space is a good strategy for exploring the environment, although this might lead to a poor convergence rate. A common heuristic used to bias the exploration around the goal is to sample from the goal region with some probability p. This probability can be understood as a trade-off between exploration and exploitation: if p is close to 1, the search becomes very greedy, although this can lead to bad convergence rates in cluttered environments; if p is close to 0, then exploration is prioritized, although this might not be good for environments with lots of free space.

3.1.7.2 Direct Connection

An interesting way of making the search more greedy is to periodically try direct connections from some node (for example, $x_{nearest}$) to the goal. For instance, if there is an obstacle-free straight line between start and goal, then we could find this path on the first algorithm iteration. Furthermore, the direct connection also increases plan smoothness, which is an inherent problem of RRT, due to its sampling strategy. This heuristic is also particularly beneficial, in the case where replanning is done at every iteration, because a bad plan can be replaced by a direct connection from start to goal.

3.1.7.3 Extend Length

It is natural to choose Δt to match the camera iteration loop, i.e., $\Delta t = \frac{1}{60}$ s. Yet this would create trees with too many nodes, which means the nearest neighbor search could be too costly. Therefore, in each RRT iteration we simulate the system for E times (extend length) and keep only the last state as a node of tree. Larger values of E might decrease execution time, but also lead to bad and unnatural paths, whereas small values will increase execution time.

3.1.7.4 Maximum Iterations

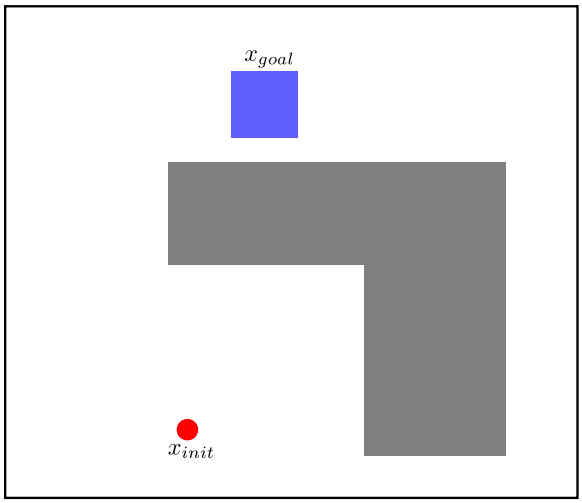
The RRT algorithm is run for K iterations as shown in the pseudocode. Therefore, the value of K can be chosen as a way of limiting the maximum running time of the algorithm, which is convenient for a real time planner. One inconvenience of this approach is that the algorithm could reach the maximum number of iterations and find no feasible solution, i.e., we would not have any commands available for the robot. The solution adopted is to find the nearest tree node with respect to the desired goal. This guarantees that the robot is at least going in the direction of the goal, although there are no guarantees of reaching it.

3.1.8 Algorithm Discussion

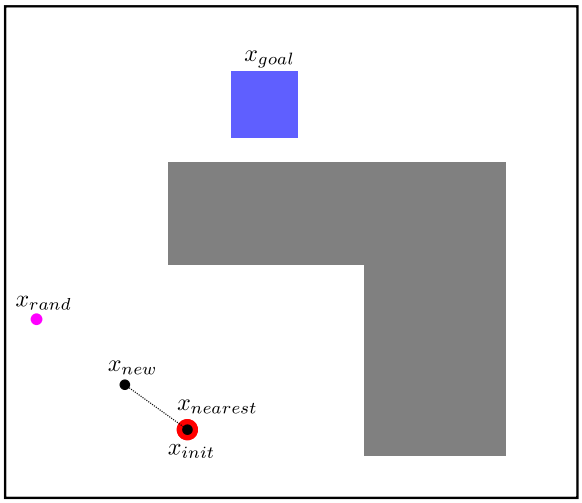
With this simple pseudocode and some optimizations discussed above, RRT is capable of solving high-dimensional piano-movers-like problems, which consist of moving a piano inside a house without colliding with the walls. For example, (YERSHOVA; LAVALLE, 2007) reports it could solve a 56-dimension problem.

Even though RRT is probabilistic complete, for a real time task, it might not find a solution given a limited computation time. Nevertheless, one could still return the best path found by searching for the nearest neighbor in the tree with respect to the goal region. The robot could start following this path and, on the next iteration, RRT could be used again with the some alterations in the tree already computed (KARAMAN *et al.*, 2011). This procedure could de repeated until RRT finds a path to the goal. We will discuss more about it later, when the replanning issue arises.

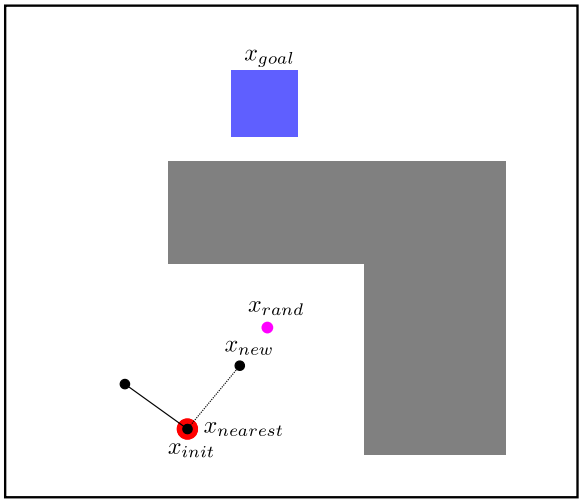
We also point out that the paths found by RRT are always suboptimal, as shown by (KARAMAN; FRAZZOLI, 2011). Therefore, even if the algorithm runs for an infinitely long time, the trajectory is never optimal. A naive approach to generate better paths, is to run the algorithm multiple times and choose the minimum cost path found. This is clearly computationally costly and inefficient, since each run of the algorithm disregards


x_{goal}
x_{init}

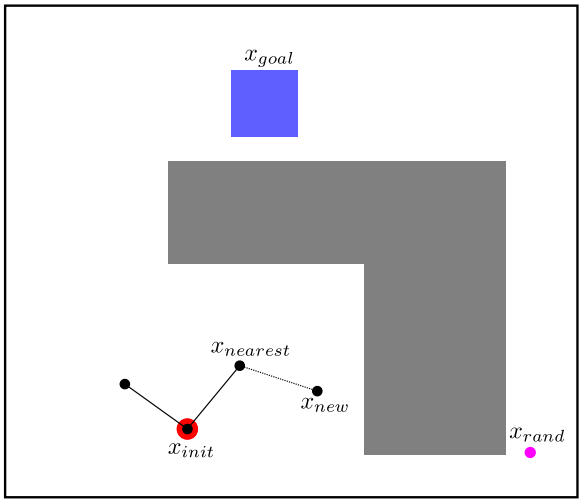
(a)


x_{goal}
x_{rand}
x_{new}
$x_{nearest}$
x_{init}

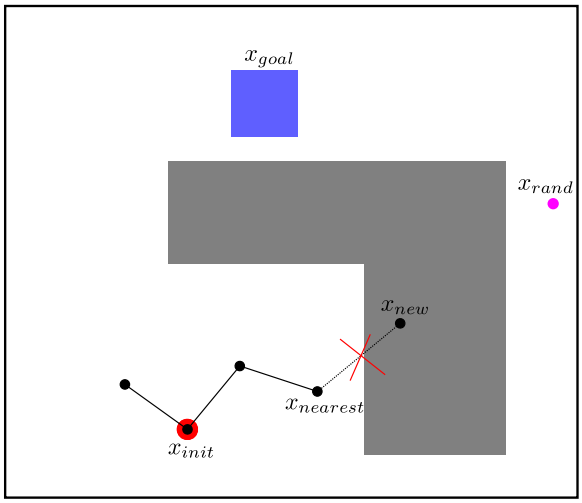
(b)


x_{goal}
x_{rand}
x_{new}
$x_{nearest}$
x_{init}

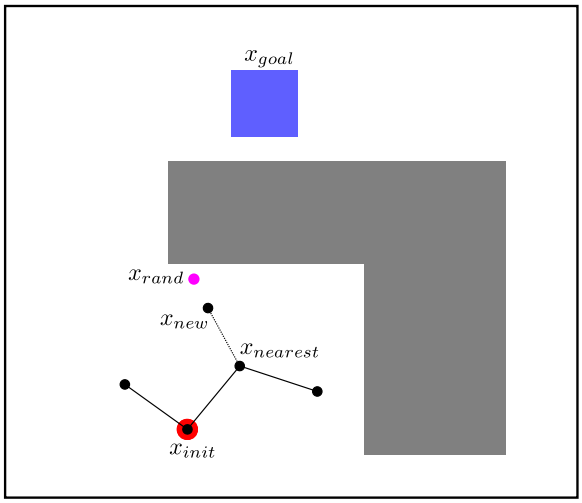
(c)


x_{goal}
$x_{nearest}$
x_{new}
x_{rand}
x_{init}

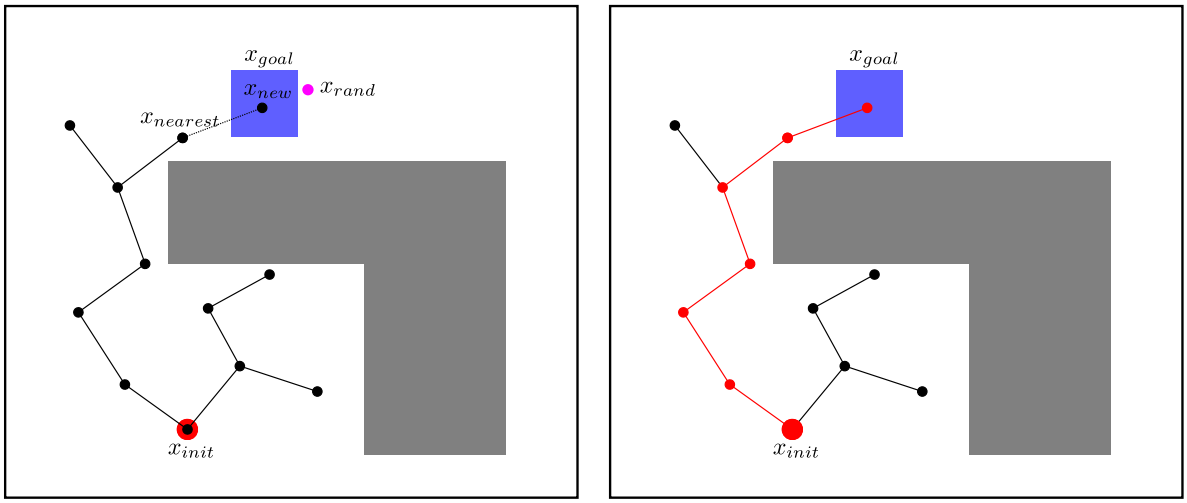
(d)


x_{goal}
x_{rand}
x_{new}
$x_{nearest}$
x_{init}

(e)


x_{goal}
x_{rand}
x_{new}
$x_{nearest}$
x_{init}

(f)



(g) (h)

FIGURE 3.1 – Step-by-step execution of the RRT algorithm.

the others, i.e., valuable information is thrown away.

A better way to do this is to use the trajectory cost of a found plan to sample more intelligently the space like is done by (OTTE; CORRELL, 2013). This paper shows that if we already know a maximum limit for the trajectory cost, we do not need to sample from the entire search space anymore: only points sampled from an ellipse can possibly generate lower cost trajectories. Then, each time a lower cost path is found, the ellipse could be shrunk. This idea is proven to generate almost sure convergence to the optimal solution.

A more efficient way of achieving optimality is to change the tree structure, if this leads to smaller paths. This is the main idea of the RRT* algorithm. On the other hand, this is a very computationally costly algorithm not suitable for the VSS fast iteration loop. Therefore, our approach to generate better quality plans is to replan at every iteration.

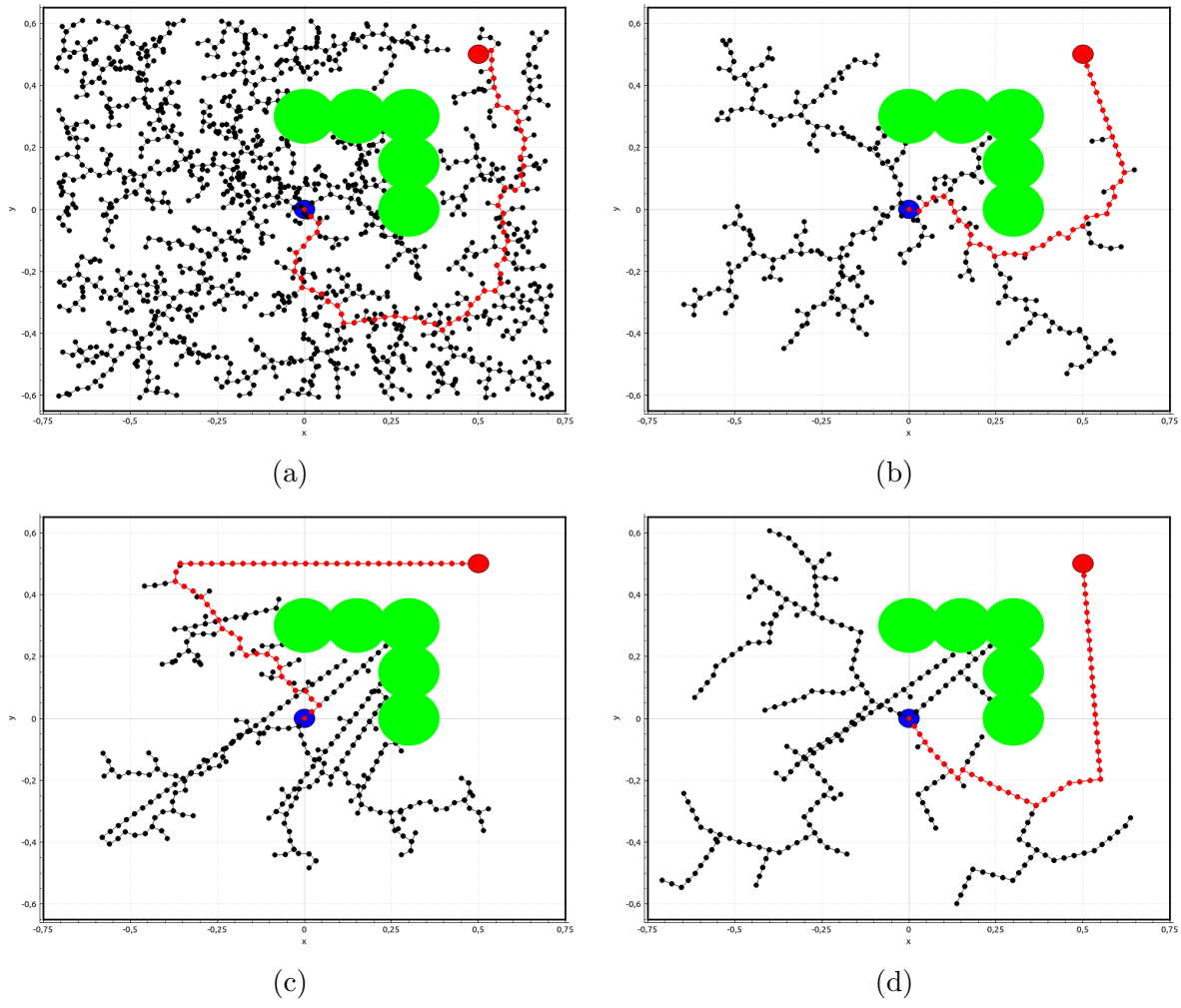
3.2 Experimental Results with RRT for Omnidirectional Planning

In this section, we show that RRT can achieve the real time performence desired and generate reasonable paths. For this, we present the simulation results for different implementations of RRT. For now, we will consider the omnidirectional robot, with a maximum velocity of 1 m/s. We will compare 4 possible heuristic optimizations:

- Original RRT, $E = 1$, $\Delta t = 30\,\text{ms}$
- RRT with goal bias of $p = 0.5$, $E = 1$, $\Delta t = 30\,\text{ms}$

- RRT with goal bias of $p = 0.5$, direct connection, $E = 1$, $\Delta t = 30\,\text{ms}$.
- RRT with goal bias of $p = 0.5$, direct connection, $E = 4$, $\Delta t = 30\,\text{ms}$.

The running time of these cases can be found in Table 3.1. Also a picture of the grown tree for each case is available in Fig. 3.2. See that the planning time is around 1 ms, i.e., within use for real time tasks. However, one may observe that the original RRT leads to winding paths and hard curves. The goal bias and direct connection heuristics help to diminish these effects. Besides, it is clear that the paths are always suboptimal.



(a) (b)

(c) (d)

FIGURE 3.2 – Examples of RRT planning for different heuristics. Fig. 3.2a is the original RRT without any heuristics. Fig. 3.2b is the RRT with goal bias of 0.5. Fig. 3.2c shows the RRT with goal bias and direct connection. Fig. 3.2d uses goal bias, direct connection and extend length of 120 ms.

TABLE 3.1 – Average of 1000 Runs of RRT with Different Heuristics

	Average Time (ms)
Original	1.89872
Goal Bias	0.910247
Goal Bias + Direct Connection	0.515399
Goal Bias + Direct Connection + Extend Length	0.294509

4 Mobile Robot Model and Pose Controller

4.1 Mobile Robot Model

This section will present 3 different DDMR models that will be used in this work for simulation purposes or for movement prediction in the RRT simulation step. First, we talk about the kinematics model, a simplified representation of the robot that disregards dynamics, i.e., it assumes that velocity can be controlled directly. Then, we show a full dynamics model, that encodes robot's inertia, a DC motor model and friction. Finally, we simplify this model assuming wheels' acceleration can be controlled directly.

4.1.1 Kinematics Model

The kinematics equations of a unicycle are shown in Eq. (4.1). The robot's position is (x, y), the orientation is θ, the linear and angular velocity are respectively v and ω. This model is an abstraction of a mobile robot and assumes that the robot is a point with controllable v and ω. However, in the real robot (see Fig. 4.1), we actually have control over the wheels' angular velocities (ω_r and ω_l) and Eq. (4.2) can be used for conversion of the unicycle model to the DDMR to obtain Eq. (4.3). It is important to note that these equations comes from the hypothesis that there are no slip effects over the wheels.

This model is quite useful if the hypothesis that one can directly control the wheels' velocities (even for step references, or in other words, the motors have infinite acceleration) is reasonable. This might be true for slow speeds, but it may not be a good approximation if the robot is running fast, since acceleration limits will lead to significant model errors.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix} \tag{4.1}$$

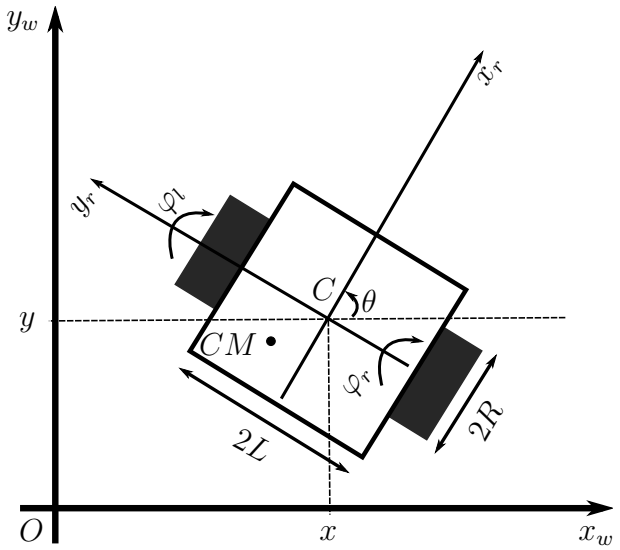


FIGURE 4.1 – Robot and world frames.

$$\begin{cases} v = \dfrac{(\omega_r + \omega_l)R}{2} \\ w = \dfrac{(\omega_r - \omega_l)R}{2L} \end{cases} \tag{4.2}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{R}{2}\cos(\theta) & \frac{R}{2}\cos(\theta) \\ \frac{R}{2}\sin(\theta) & \frac{R}{2}\sin(\theta) \\ \frac{R}{2L} & -\frac{R}{2L} \end{bmatrix}}_{\boldsymbol{S}} \underbrace{\begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix}}_{\boldsymbol{\eta}} \tag{4.3}$$

4.1.2 Full Dynamics Model

Next, Eq. (4.4) presents a high-fidelity model, that was obtained through a system identification of the robot (OKUYAMA *et al.*,). It will be used mainly for simulation of the system and we consider it to be our ground truth in all simulations experiments. This equation represents the dynamics of the wheels. The model takes into account the robot's mass and inertia, DC motor parameters like gear reduction, viscous friction, rotor inertia and torque constant, and static and kinetic friction (which are represented by $\boldsymbol{F}$). Note also that the system input is the voltage on each motor, $\boldsymbol{V}$. For a complete explanation of the parameters see (OKUYAMA *et al.*,). For now, it suffices to say that the system can

be written in state-space form as in Eq. (4.5).

$$
\begin{aligned}
&\bar{\boldsymbol{H}}_m\dot{\boldsymbol{\eta}} + \bar{\boldsymbol{C}}_m\boldsymbol{\eta} = \bar{\boldsymbol{B}}_m(\boldsymbol{V} - \boldsymbol{F}) \\
&\bar{\boldsymbol{H}}_m = \begin{bmatrix} I_1 + N^2 I_m & I_2 \\ I_2 & I_3 + N^2 I_m \end{bmatrix} \\
&\bar{\boldsymbol{C}}_m = \begin{bmatrix} eN^2\left(\frac{K^2}{Res} + b\right) & \frac{R^2}{2L} m x_{cm}\dot{\theta} \\ -\frac{R^2}{2L} m x_{cm}\dot{\theta} & eN^2\left(\frac{K^2}{Res} + b\right) \end{bmatrix} = \\
&\qquad = \begin{bmatrix} eN^2\left(\frac{K^2}{R_{es}} + b\right) & \frac{R^3}{4L^2} m x_{cm}\left(\dot{\varphi}_r - \dot{\varphi}_l\right) \\ -\frac{R^3}{4L^2} m x_{cm}\left(\dot{\varphi}_r - \dot{\varphi}_l\right) & eN^2\left(\frac{K^2}{R_{es}} + b\right) \end{bmatrix} \\
&\bar{\boldsymbol{B}}_m = \begin{bmatrix} \frac{eNK}{R_{es}} & 0 \\ 0 & \frac{eNK}{R_{es}} \end{bmatrix}
\end{aligned}
\tag{4.4}
$$

$$
\begin{aligned}
\dot{\boldsymbol{\eta}} &= \boldsymbol{A_c}\boldsymbol{\eta} + \boldsymbol{B_c}(\boldsymbol{V} - \boldsymbol{F}) \\
\boldsymbol{A_c} &= -\bar{\boldsymbol{H}}_m^{-1}\bar{\boldsymbol{C}}_m \\
\boldsymbol{B_c} &= \quad \bar{\boldsymbol{H}}_m^{-1}\bar{\boldsymbol{B}}_m
\end{aligned}
\tag{4.5}
$$

By joining Eqs. (4.5) and (4.3), we can find the dynamics equations for the full state $\boldsymbol{q} = \begin{bmatrix} x & y & \theta & \omega_r & \omega_l \end{bmatrix}^T$:

$$
\dot{\boldsymbol{q}} = \underbrace{\begin{bmatrix} \boldsymbol{0}_{3\times 3} & \boldsymbol{S} \\ \boldsymbol{0}_{2\times 3} & \boldsymbol{A_c} \end{bmatrix}}_{\boldsymbol{A}} \boldsymbol{q} + \underbrace{\begin{bmatrix} \boldsymbol{0}_{3\times 2} \\ \boldsymbol{B_c} \end{bmatrix}}_{\boldsymbol{B}} (\boldsymbol{V} - \boldsymbol{F}) \tag{4.6}
$$

Finally, the robot has one speed controller for each wheel, implemented by means of PI controllers. This is done, since the feedback loop greatly attenuates the effects of modeling errors.

4.1.2.1 Reference Filtering

One important detail is that one must assure that the hypothesis of no wheel slip is valid. Then, we must limit the maximum acceleration of the robot, so that the motors do not imply a force greater than the maximum friction between wheels and ground. Since, the robots receive a step reference from the external computer, the PI controllers might produce very high accelerations. Therefore, we filter the reference to make it have a ramp shape, due to the fact that the ramp angle represents the acceleration. See Fig. 4.2 to

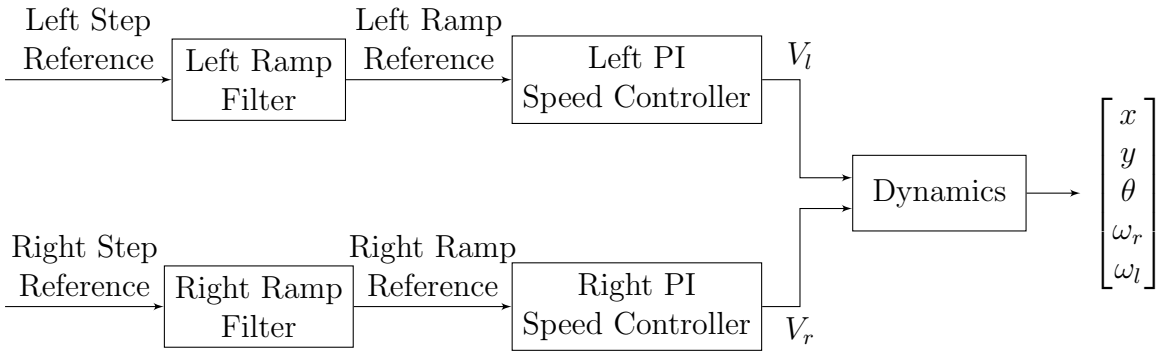


FIGURE 4.2 – Diagram block showing how the robot is simulated.

see the complete diagram block for simulation of the robot.

4.1.3 Simplified Dynamics Model

Although the model presented on the last section is very close to the real robot, its complexity might not suit a real time planner like RRT, since we need to integrate it multiple times to generate a trajectory. In addition, the PI controllers needs memory for the integrator, i.e., the integrator value should also be included in the robot state, which adds a bit more complexity on the implementation of the algorithm. Then, considering the PI controllers guarantee a fast rise time and we are ramp filtering the reference, we could suppose we are actually controlling the acceleration of each wheel. Therefore, we developed the following simplified dynamics model:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\omega_r} \\ \dot{\omega_l} \end{bmatrix} = \begin{bmatrix} \frac{(\omega_r+\omega_l)R}{2}\cos(\theta) \\ \frac{(\omega_r+\omega_l)R}{2}\sin(\theta) \\ \frac{(\omega_r-\omega_l)R}{2L} \\ \alpha_r \\ \alpha_l \end{bmatrix} \tag{4.7}$$

where α_r and α_l are the angular accelerations of the right and left wheel respectively. In this model, we assume these variables are the control inputs, which are limited in order to avoid wheel slip. We use the same strategy of ramp filtering to respect the acceleration constraints:

$$\begin{cases} -\alpha_{max} \leq \alpha_r \leq \alpha_{max} \\ -\alpha_{max} \leq \alpha_l \leq \alpha_{max} \end{cases} \tag{4.8}$$

Finally, we remark that curve effects like centripetal forces or movement of the center of pressure also affect the wheel's maximum angular acceleration, although they are

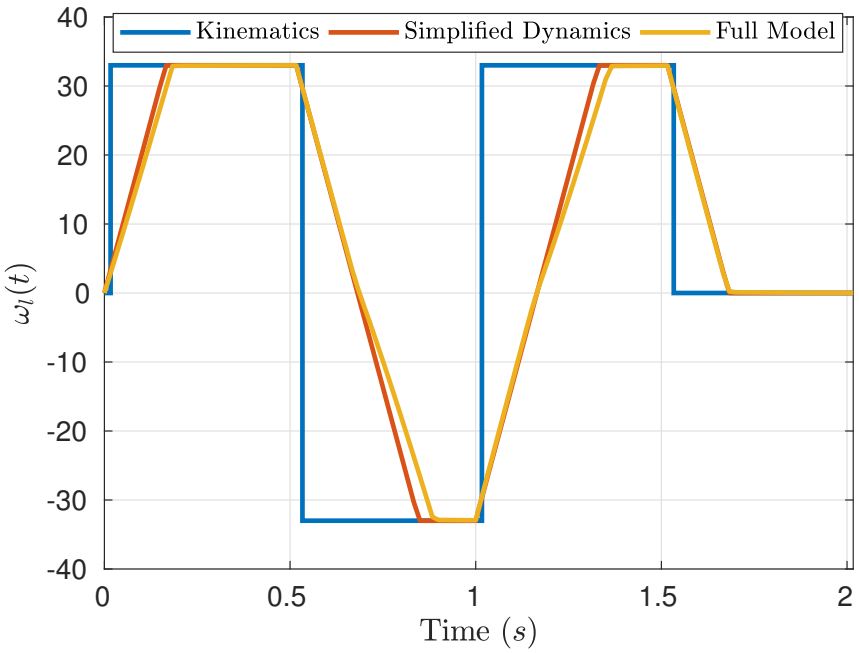


FIGURE 4.3 – Velocity response comparison of the left wheel angular velocity of the 3 DDMR models, due to step references.

TABLE 4.1 – Model Parameters

$R =$	$3.00 \times 10^{-2}\,\mathrm{m}$	$\alpha_{max} =$	$200\,\mathrm{rad/s^2}$
$L =$	$3.31 \times 10^{-2}\,\mathrm{m}$	$V_{max} =$	$7\,\mathrm{V}$
$\boldsymbol{A_c} =$	$\begin{bmatrix} -6.1585 & 0.8842 \\ 0.8842 & -6.1585 \end{bmatrix}$	$\boldsymbol{F} =$	$\begin{bmatrix} 0.7\tanh(2.5\omega_r) - 0.3\tanh(0.4\omega_r) \\ 0.7\tanh(2.5\omega_l) - 0.3\tanh(0.4\omega_l) \end{bmatrix}$
$\boldsymbol{B_c} =$	$\begin{bmatrix} 67.7331 & -7.0182 \\ -7.0182 & 67.7331 \end{bmatrix}$		

disregarded in this analysis for simplicity.

For a comparison of the 3 presented models, see Fig. 4.3. Observe that the simplified dynamics model closely approximates the full model, although the full model accelerates a little less than the simplified one. The little discrepancies might be attributed to the non-linearity of friction, since there are two distinct case: friction might be acting against the controller, when it has positive velocity and is accelerating, or in favor for it, when it has positive velocity ans is decelerating. The kinematics model, however, goes instantaneously to the reference, not respecting acceleration limits.

4.2 Pose Controller

A pose controller is responsible for driving the robot from a starting pose $P_S = (x_S, y_S, \theta_S)$ to a desired pose $P_D = (x_D, y_D, \theta_D)$ like Fig. 4.4 demonstrates. We chose

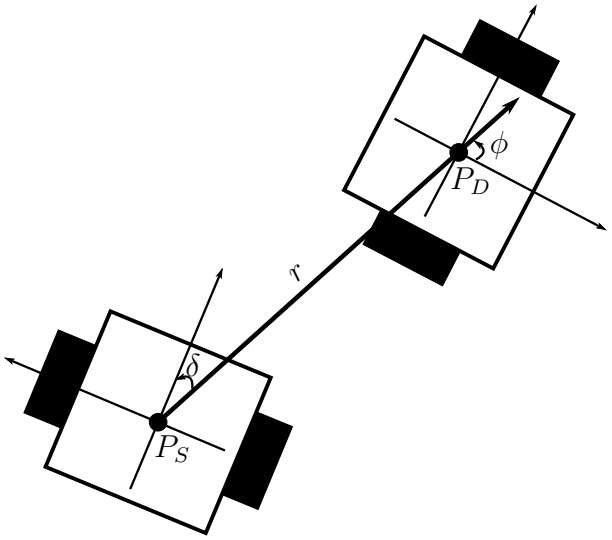


FIGURE 4.4 – Robot and world frames.

the following controller described in (PARK, 2016):

$$\omega = \kappa(r, \phi, \delta)v = -\frac{v}{r}\left[k_\delta\left(\delta - \arctan(-k_\phi\phi) + \left(1 + \frac{k_\phi}{1 + (k_\phi\phi)^2}\right)\sin\delta\right)\right] \tag{4.9}$$

where k_ϕ and k_δ are constants. This controller has the interesting property that it has stability guarantee for any choice of velocity profile. Furthermore, if $v \to 0$, then the convergence is asymptotic. We also point out that the controller generates paths that are independent from the velocity profile, since it chooses a desired curvature, κ, for each pose.

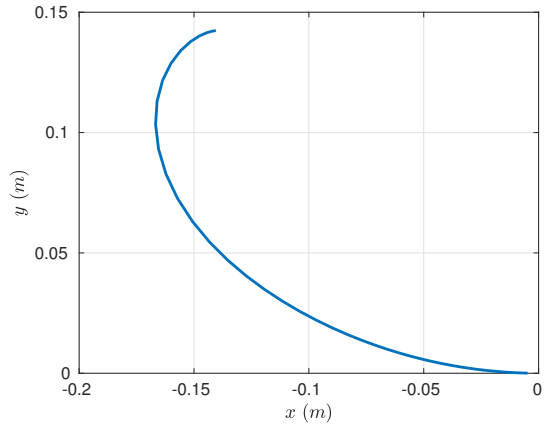
Therefore, in order to have a pose controller that stabilizes in a desired pose, we chose the following velocity profile:

$$v = v_{max}\tanh(k_t r) \tag{4.10}$$

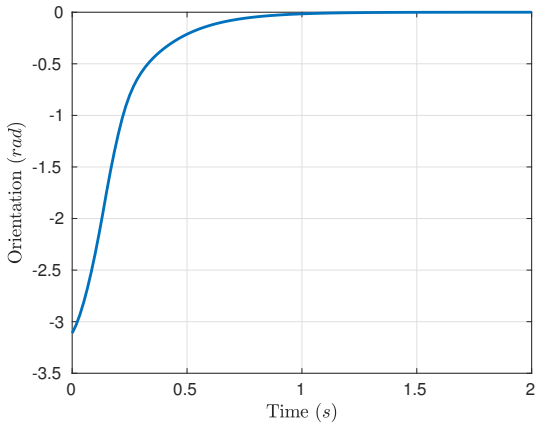
In Fig. 4.5, one can see an example of the response of the pose controller. The results were generated simulating the interaction between the controller and the full dynamics model. The controller parameters are in Table 4.2.

TABLE 4.2 – Pose controller parameters.

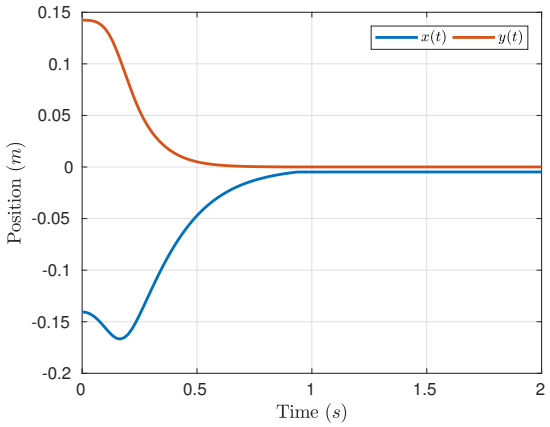
k_ϕ	k_δ	k_t	v_{max}
1	4	5	1 m/s



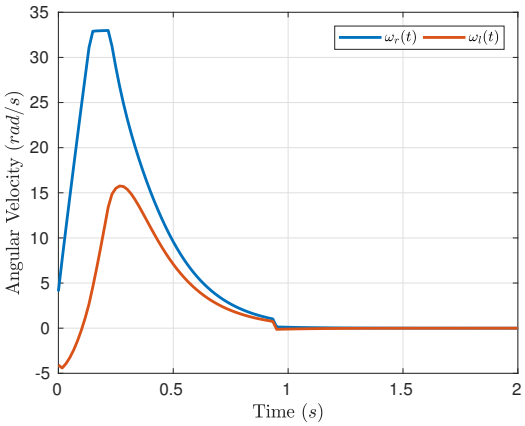
(a) Path outcome.



(b) Position convergence.



(c) Orientation convergence.



(d) Velocity profile.

FIGURE 4.5 – Example of a run of the pose controller. The initial state is $x = -14 \times 10^{-2}$ m, $y = 14 \times 10^{-2}$ m, $\theta = -\pi$, $\omega_r = 0$, $\omega_l = 0$ and the desired pose is the origin.

5 RRT for the DDMR

As discussed in Ch. 3, two functions are needed to adapt the original RRT algorithm to the DDMR: an extend function, and a robot simulator. In Ch. 4, we presented 3 possible simulation models and a pose controller that is a natural choice for the extend function. In fact, only the kinematics model and the simplified dynamics model are used as RRT simulation functions, while the full dynamics model is used for execution of the plan, because it is a good approximation of the real robot. Then, it is possible to assess the advantages of including a dynamic model in the planning step, as will be shown in the next sections. From now on, we name the RRT algorithm with the kinematics model and the simplified dynamics model as Kinematic RRT and Dynamic RRT, respectively.

5.1 Open Loop Planning Execution

This section is focused on measuring how good the generated plans are considering the mismatch between the model used for planning and the full dynamics model. For that we consider two scenarios:

- Scenario 1 (Random Obstacles): The robot needs to traverse the soccer field, while avoiding 6 obstacles that are randomly generated on the field. This is a common scenario in soccer robot games. The robot's initial state is $x_S = -0.6\,\text{m}$, $y_S = 0\,\text{m}$, $\theta_S = \frac{\pi}{2}\,\text{rad}$, $\omega_{r,S} = 0\,\text{rad/s}$, $\omega_{l,S} = 0\,\text{rad/s}$ and the desired pose is $x_D = 0.6\,\text{m}$, $y_D = 0\,\text{m}$, $\theta_D = \frac{\pi}{2}\,\text{rad}$.

- Scenario 2 (Going into Obstacle): The robot start very close to an obstacle and needs to go the a position that is also near it. Here, we can evaluate the finite acceleration effects, since transient will play a major role in the planning success. The robot's initial state is $x_S = -0.2\,\text{m}$, $y_S = 0\,\text{m}$, $\theta_S = 0\,\text{rad}$, $\omega_{r,S} = 0\,\text{rad/s}$, $\omega_{l,S} = 0\,\text{rad/s}$ and the desired pose is $x_D = 0.3\,\text{m}$, $y_D = 0\,\text{m}$, $\theta_D = 0\,\text{rad}$. The obstacle on in the origin.

After computing the plan, we make the full model follow it by executing the plan's

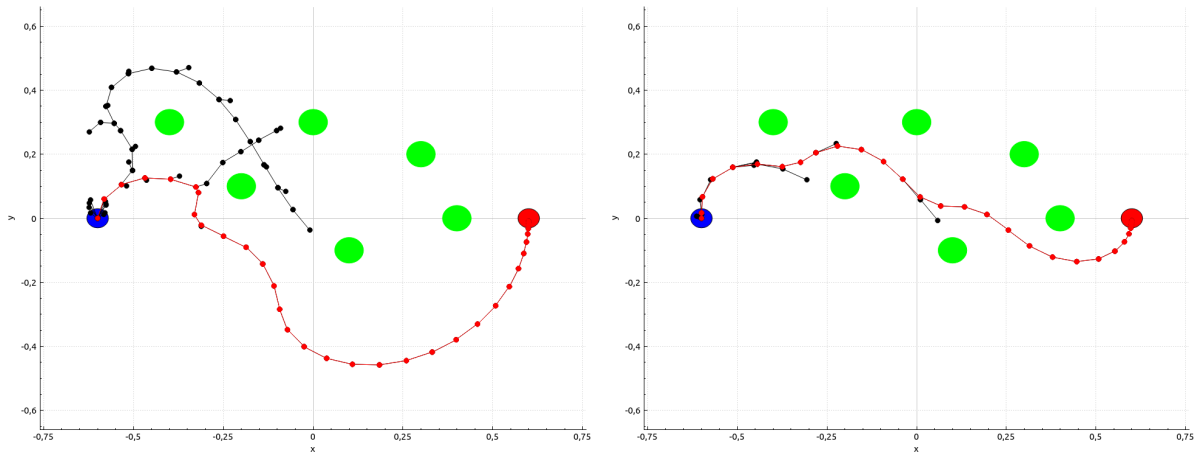
actions. In this case, there is no replanning and we call it an open loop execution of the plan.

We use the following measures to compare the planners:

- Computational Time: is the time of generating a plan. This is important, since we want a real-time planner, that runs faster than our camera loop (60 Hz).
- Mean Path Length: is the length of a path. Smaller paths might be better, although it might not be achievable by a real robot.
- Trajectory Following Error: is the average position error, i.e., the error between the position of the full dynamics model and the planned position. This indicates how good the planner model is.
- Collision Rate: is the number of plans that generated collisions when executed over the total number of plans. This gives a notion of planning safety.

We point out that, due to the probability characteristic of the RRT, we run the same problem over 10000 times and take the average. Also, the RRT parameters used are the same for both planners (maximum iterations = 1000, probability of goal sampling = 0.5, extend length = 5, direct connection period = 30).

The Tables 5.1 and 5.2 shows the results for the experiments with Scenarios 1 and 2. Additionally, we also show an example of the grown RRT in both scenarios in Figs. 5.1 and 5.2.

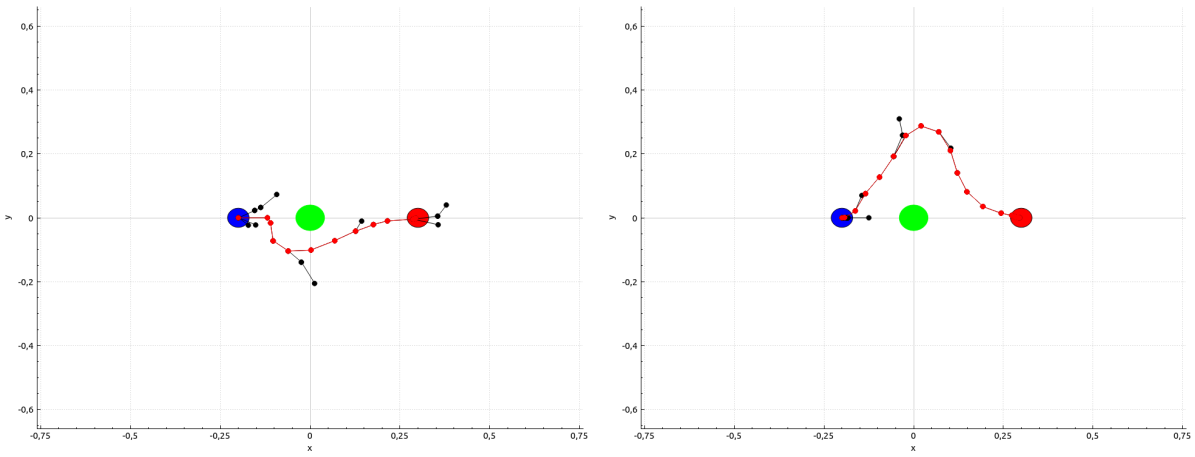


(a) Kinematics model. (b) Simplified Dynamics Model.

FIGURE 5.1 – Example of a run of the Kinematic and Dynamic RRT algorithm for Scenario 1. Blue is the start position and red is the target.

TABLE 5.1 – Comparison of RRT generated plans for the Random Obstacles case (average over 10000 experiments).

	Dynamic RRT	Kinematic RRT
Computational Time	1.01 ms	0.499 ms
Mean Path Length	1.76 m	1.60 m
Trajectory Following Error	0.029 m	0.072 m
Collision Rate	10.72 %	23.98 %



(a) Kinematics model. (b) Simplified Dynamics Model.

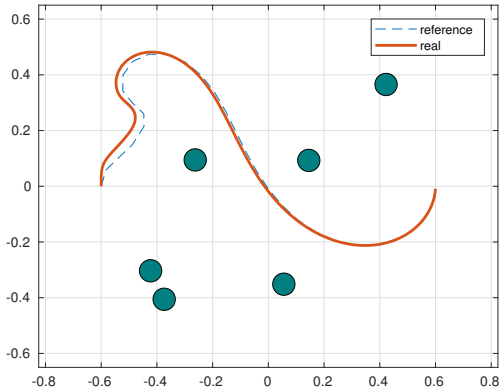
FIGURE 5.2 – Example of a run of the Kinematic and Dynamic RRT algorithm for Scenario 2. Blue is the start position and red is the target.

TABLE 5.2 – Comparison of RRT generated plans for the Going Into Obstacle case (average over 10000 experiments).

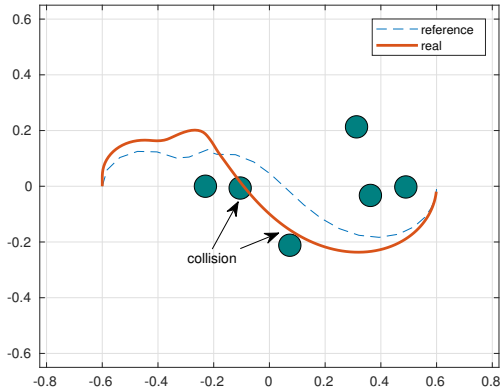
	Dynamic RRT	Kinematic RRT
Computational Time	0.619 ms	0.244 ms
Mean Path Length	1.00 m	0.71 m
Trajectory Following Error	0.025 m	0.037 m
Collision Rate	2.66 %	25.42 %

We can see that the Dynamic RRT takes more than the double of computational time with respect to the Kinematic version. This is expected, since the dynamics model has more states, which also means the Dynamic RRT is solving a harder problem. However, the running time is still admissible. Notice also that the planned path length and time are smaller for the Kinematic RRT, since this model assume infinite acceleration. The advantages of the Dynamic RRT are clear, on the other, if we look at the Trajectory Following Error and the collision rate. In the Scenario 1, the Trajectory Following Error is 2 times greater for the Kinematic RRT and we remark that the error is of about the size of the robot (robot's length = 0.075 m). The collision rate of the Kinematic RRT is also very high and more than 2 times greater than the Dynamic version. Looking at Scenario

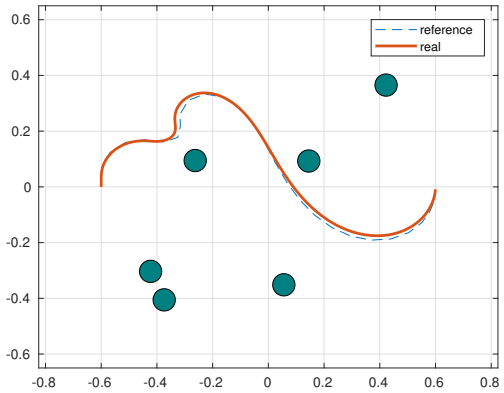
2, we see that the Trajectory Error decreased for both planners, which is expected, since the plans are smaller. Nevertheless, the collision rate is very interesting: the Dynamic RRT is about 6 times safer than the other planner. This shows that considering the dynamics of the model boosts the plan safety, mainly in the cases of imminent collision. Some examples of the plan execution can be found in Figs. 5.3 and 5.4.



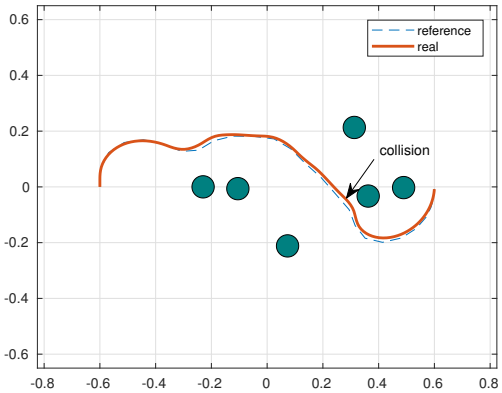
(a) Kinematics model.



(b) Kinematics model.

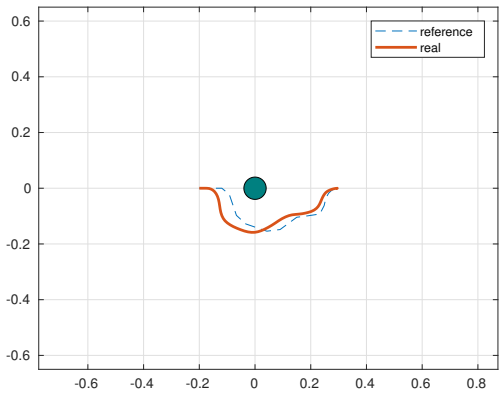


(c) Simplified Dynamics model.

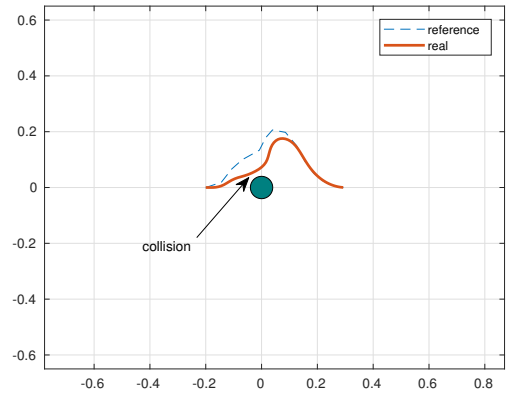


(d) Simplified Dynamics model.

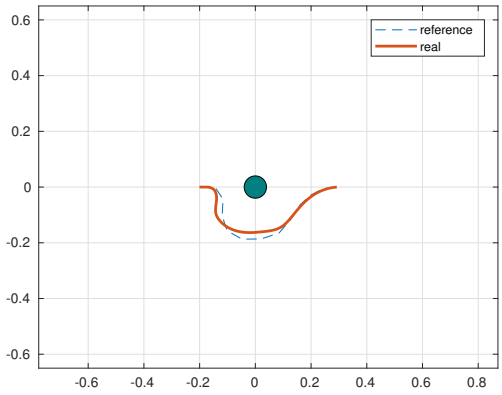
FIGURE 5.3 – Example of executions of RRT plans for the Random Obstacles Scenario.



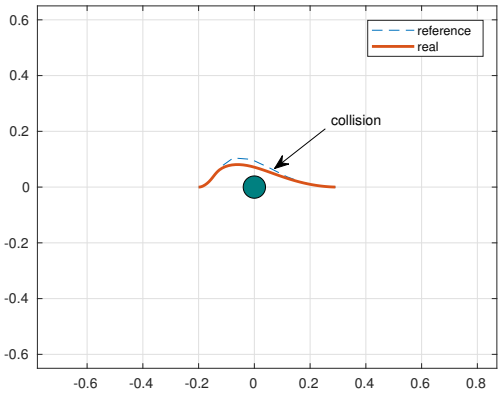
(a) Kinematics model.



(b) Kinematics model.



(c) Simplified Dynamics model.



(d) Simplified Dynamics model.

FIGURE 5.4 – Example of executions of RRT plans for the Going Into Obstacle Scenario.

5.2 Replanning

In the last section, the generated plans were executed directly, i.e., without any replanning. One could argue that the replanning effect would be quite similar to feedback in control systems. In other word, replanning adds robustness to the system. Therefore, in order to test this hypothesis, we did the use the same scenarios from last section, but the planner creates a new plan at every time step. This is possible, since the execution time is quite small.

Regarding the quality measures, we only used the collision rate and the mean path length, since the average computational time is not expected to change and it does not make sense to use the Trajectory Tracking Error, because there is no reference trajectory, due to the replanning. Also, we remark that the mean path length in this case is the

length that the real robot executed, instead of the planned length.

Tables 5.3 and 5.4 show the results of the replanning experiments. We can see that the collision rate dropped drastically in comparison with the open loop execution. However, the Dynamic RRT is still considerably safer than the Kinematic RRT. It is also interesting to see that the planning length is the same for both planners. This can be explained by the fact that the same dynamic model is used for plan execution and the same pose controller is used in the RRT. Therefore, the replanning indeed increases the robustness to collisions of the system.

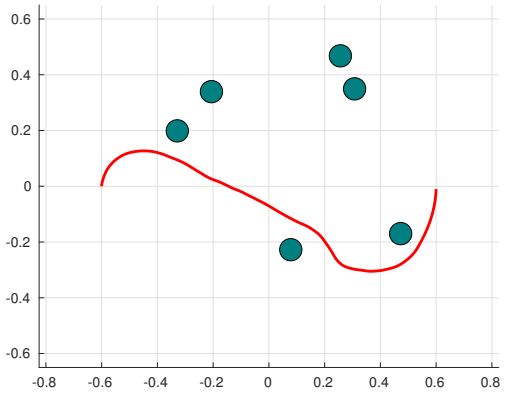
TABLE 5.3 – Comparison of the Replanning RRT for the Random Obstacles case (average over 10000 experiments).

	Dynamic RRT	Kinematic RRT
Mean Path Length	1.70 m	1.70 m
Collision Rate	0.10 %	5.59 %

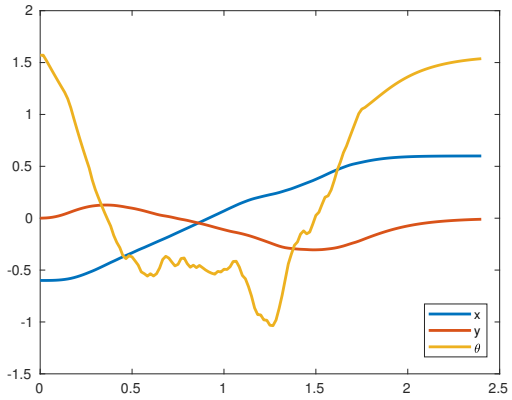
TABLE 5.4 – Comparison of the Replanning RRT for the Going Into Obstacle case (average over 10000 experiments).

	Dynamic RRT	Kinematic RRT
Mean Path Length	0.69 m	0.69 m
Collision Rate	0.24 %	1.68 %

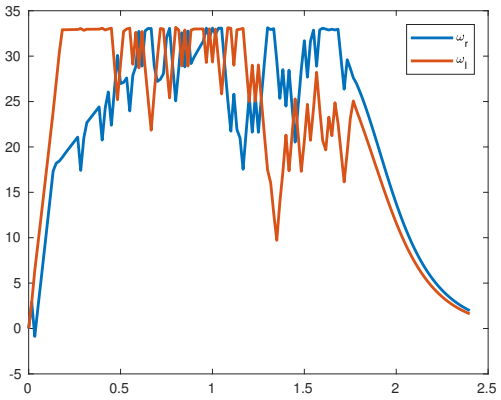
Two examples of the execution of the Dynamic RRT with replanning are shown in Figs. 5.5 and 5.6. It is noticeable that the angular velocity is quite noisy and this reflects on the robot's orientation. This is caused by the randomness of the RRT algorithm, which is repeated at every iteration. This could be avoided by a more intelligent algorithm for replanning. For example, instead of running the RRT again at every iteration, one could only replan if the robot deviated above some threshold from the planned trajectory.



(a) Executed path.

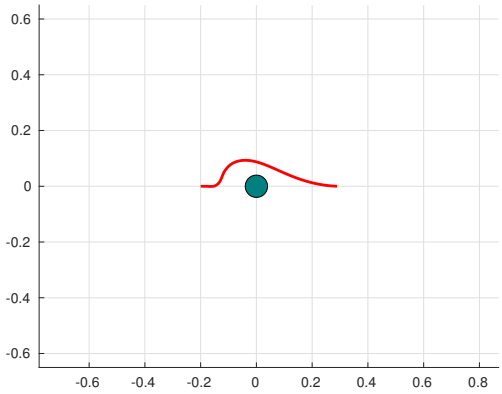


(b) Pose trajectory.

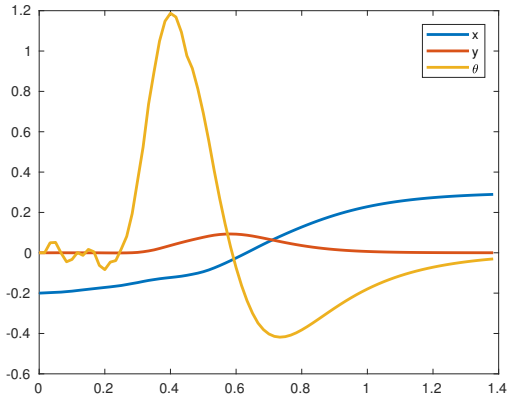


(c) Velocity profile.

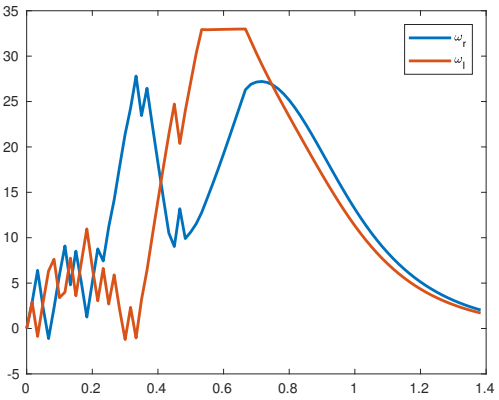
FIGURE 5.5 – Example of executions of Dynamic Replanning RRT for Random Obstacles Scenario.



(a) Executed path.



(b) Pose trajectory.



(c) Velocity profile.

FIGURE 5.6 – Example of executions of Dynamic Replanning RRT for Going Into Obstacle Scenario.

5.3 Experiments with the Real Robot

This section is dedicated to analyze the performance of the Dynamic RRT algorithm, when running in the real VSS system, which has unmodeled quantities such as camera latency, communication delay, noise measurements and modeling errors. These effects might potentially affect the stability and safety of the algorithm and the maximum possible velocity, since we only have access to noisy and delayed observations. In our case, the observation latency is about 2 camera frames (OKUYAMA *et al.*,), i.e., approximately 33 ms, and even though the position and orientation noise are quite low, the linear and angular velocities are measured by simple numerical differentiation. The radio communication is quite fast, since it has a transmission rate of 200 kbps, and its effects are negligible in front of the camera delay.

Regarding model errors, Fig. 5.7 shows a comparison between the response of the

simplified dynamics model and the real robot to the same reference. The real robot data was taken from the encoders coupled to the motor. One can see that the velocities are very close, meaning that the robot can track reasonably well the ramp references.

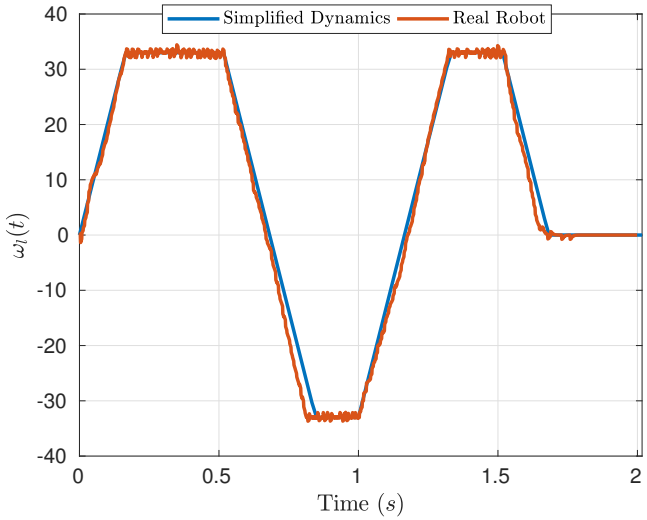
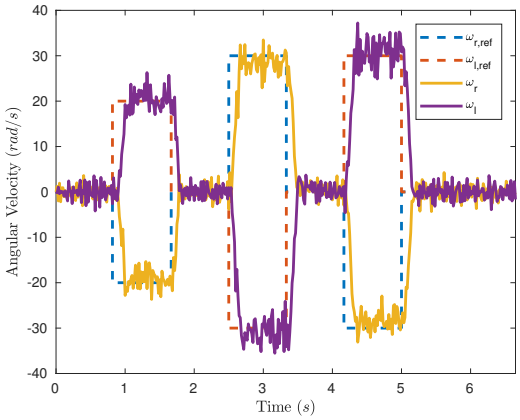


FIGURE 5.7 – Real robot tracking ramp references. Data taken from encoders.

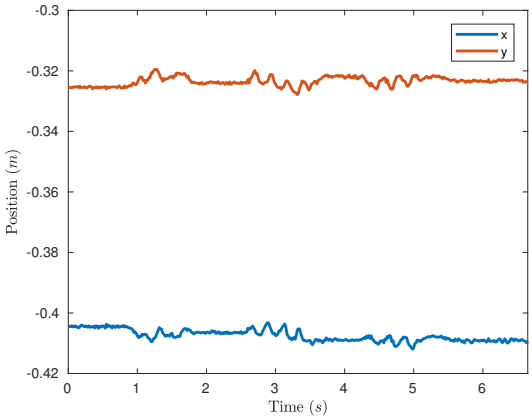
In order to assess the camera measurement quality and delay, we conducted an experiment of sending a command to the robot of rotating around its center. The results can be seen in Fig. 5.8. It is noticeable that even though the pose estimates are quite accurate, the differentiation amplifies the noise and results in noisy speed estimates. Observe that even when the robot is stopped, the velocity noise is considerable. This impairs the planner, since it uses the estimates as the initial state of the search.

Then, we ran several experiments of the Dynamic RRT with replanning in our VSS system. We could not use the same parameters from the simulations of Secs. 5.1 and 5.2. Mainly due to the noisy observations and latency, we had to decrease the maximum velocity to 0.75 m/s in order to make the system stable.

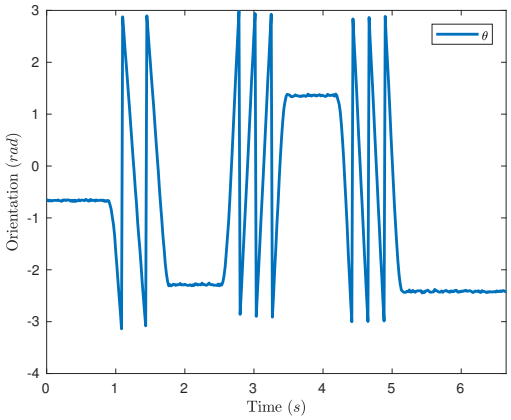
We chose 2 experiments to analyze and their results are displayed in Figs. 5.9 and 5.10. In the experiment 1, the planner can find a smooth path from the start to goal, and it can succeed without any collisions. Observe that the velocity decays smoothly, since we chose the tanh function to generate the velocity profile in the pose controller. In experiment 2, on the other hand, one can see that the velocity is much noisier than in experiment 1. The cause of this is the constant replanning and the probabilistic characteristic or RRT. In this case, the robot also collided, which is recognizable by the discontinuity in the path (Fig. 5.10d). Nevertheless, it could quickly recover from collision and reach the target successfully. If higher top speed is required, then it is necessary to utilize localization techniques like the Extended Kalman Filter (PINTO; BRUNO,), to filter the noise and consider the camera delay.



(a) Angular velocity references and estimates.

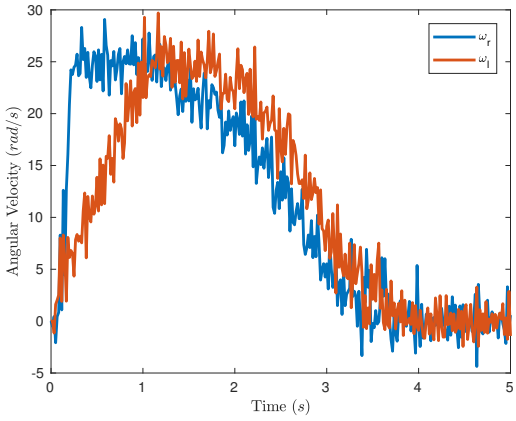


(b) Position estimates.

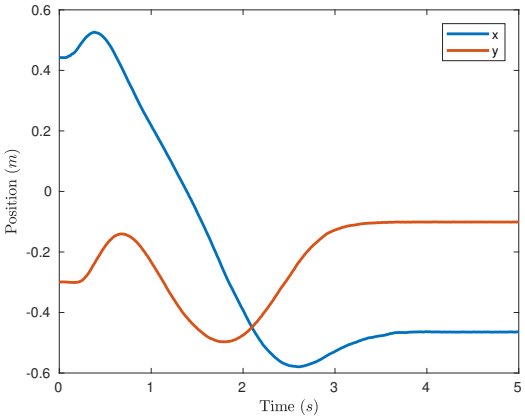


(c) Orientation estimates.

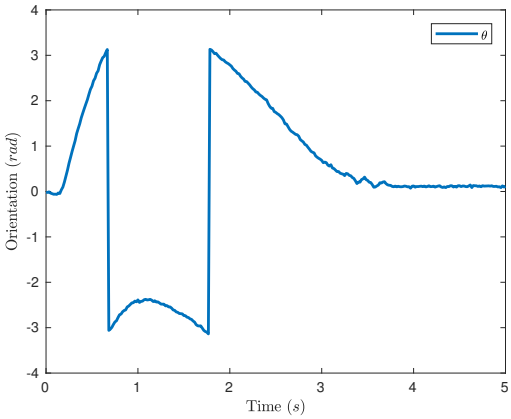
FIGURE 5.8 – Camera measurements.



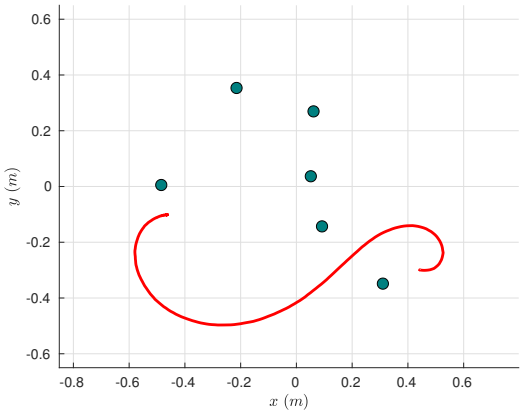
(a) Angular velocity references and estimates.



(b) Position estimates.

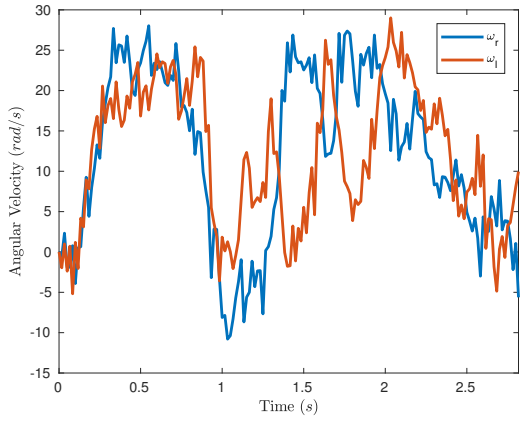


(c) Orientation estimates.

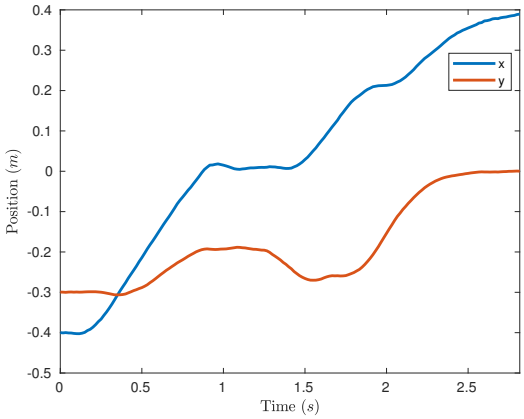


(d) Followed trajectory.

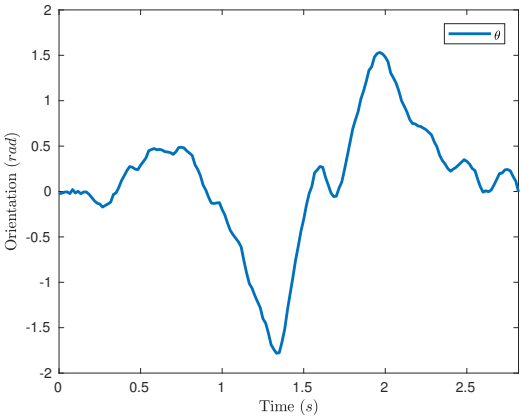
FIGURE 5.9 – Results from real robot experiment 1. The robot goes smoothly to the objective.



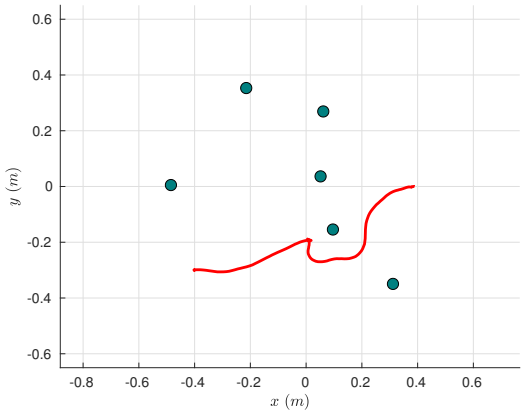
(a) Angular velocity references and estimates.



(b) Position estimates.



(c) Orientation estimates.



(d) Followed trajectory.

FIGURE 5.10 – Results from real robot experiment 2. The robot collides, but it is able to recover and reach the objective.

6 Conclusion and Future Work

This work showed the possibility of including dynamics of a DDMR in the formulation of RRT. We demonstrated through simulations that the Dynamic RRT offers better safety and dynamic compatibility with the real robot than the Kinematic RRT. It is particularly more suitable when the robot is moving fast, and acceleration limits become important for safe movement. However, we point out the commands to the robot were quite noisy, since the RRT was replanned at every iteration. A possible future work is to develop a smarter replan procedure, that tries to reuse old plans like in (KUWATA *et al.*, 2009). Another future work is to include optimality guarantees maybe through the RRT*, although it is a challenge to run it in real-time .

We also executed experiment on a real soccer robot, but we had to decrease its maximum speed, since the planner was not fed with information about sensor noise or latency. Future work includes testing the system integrated with some filtering technique like the EKF and evaluating if the RRT performance is enhanced.

Bibliography

BALKCOM, D. J.; MASON, M. T. Time optimal trajectories for bounded velocity differential drive vehicles. **The International Journal of Robotics Research**, SAGE Publications Sage UK: London, England, v. 21, n. 3, p. 199–217, 2002.

BELLMAN, R. Dynamic programming and lagrange multipliers. **Proceedings of the National Academy of Sciences**, National Acad Sciences, v. 42, n. 10, p. 767–769, 1956.

BRUCE, J.; VELOSO, M. Real-time randomized path planning for robot navigation. In: IEEE. **Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on**. [S.l.], 2002. v. 3, p. 2383–2388.

COMPETITION 7th L. A. I. R. **Rules for the IEEE Very Small Competition**. 2008. `http://www.cbrobotica.org/wp-content/uploads/2014/03/VerySmall2008_en.pdf`. Accessed: Apr 6th, 2017.

DONALD, B.; XAVIER, P.; CANNY, J.; REIF, J. Kinodynamic motion planning. **Journal of the ACM (JACM)**, ACM, v. 40, n. 5, p. 1048–1066, 1993.

DUBINS, L. E. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. **American Journal of mathematics**, JSTOR, v. 79, n. 3, p. 497–516, 1957.

GAMMELL, J. D.; SRINIVASA, S. S.; BARFOOT, T. D. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In: IEEE. **Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on**. [S.l.], 2014. p. 2997–3004.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE transactions on Systems Science and Cybernetics**, IEEE, v. 4, n. 2, p. 100–107, 1968.

JANSON, L.; ICHTER, B.; PAVONE, M. Deterministic sampling-based motion planning: Optimality, complexity, and performance. **arXiv preprint arXiv:1505.00023**, 2015.

JEON, J. hwan; KARAMAN, S.; FRAZZOLI, E. Anytime computation of time-optimal off-road vehicle maneuvers using the rrt. In: IEEE. **Decision and Control and**

European Control Conference (CDC-ECC), 2011 50th IEEE Conference on. [S.l.], 2011. p. 3276–3282.

KARAMAN, S.; FRAZZOLI, E. Sampling-based algorithms for optimal motion planning. **The international journal of robotics research**, Sage Publications Sage UK: London, England, v. 30, n. 7, p. 846–894, 2011.

KARAMAN, S.; WALTER, M. R.; PEREZ, A.; FRAZZOLI, E.; TELLER, S. Anytime motion planning using the rrt. In: IEEE. **Robotics and Automation (ICRA), 2011 IEEE International Conference on**. [S.l.], 2011. p. 1478–1483.

KAVRAKI, L. E.; KOLOUNTZAKIS, M. N.; LATOMBE, J.-C. Analysis of probabilistic roadmaps for path planning. **IEEE Transactions on Robotics and Automation**, IEEE, v. 14, n. 1, p. 166–171, 1998.

KAVRAKI, L. E.; SVESTKA, P.; LATOMBE, J.-C.; OVERMARS, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. **IEEE transactions on Robotics and Automation**, IEEE, v. 12, n. 4, p. 566–580, 1996.

KUFFNER, J. J.; LAVALLE, S. M. Rrt-connect: An efficient approach to single-query path planning. In: IEEE. **Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on**. [S.l.], 2000. v. 2, p. 995–1001.

KUWATA, Y.; TEO, J.; FIORE, G.; KARAMAN, S.; FRAZZOLI, E.; HOW, J. P. Real-time motion planning with applications to autonomous urban driving. **IEEE Transactions on Control Systems Technology**, IEEE, v. 17, n. 5, p. 1105–1118, 2009.

LAVALLE, S. M. Rapidly-exploring random trees: A new tool for path planning. Citeseer, 1998.

LIM, Y.; CHOI, S.-H.; KIM, J.-H.; KIM, D.-H. Evolutionary univector field-based navigation with collision avoidance for mobile robot. **IFAC Proceedings Volumes**, Elsevier, v. 41, n. 2, p. 12787–12792, 2008.

LINDEMANN, S. R.; LAVALLE, S. M. Steps toward derandomizing rrts. In: IEEE. **Robot Motion and Control, 2004. RoMoCo'04. Proceedings of the Fourth International Workshop on**. [S.l.], 2004. p. 271–277.

MIRTICH, B. V-clip: Fast and robust polyhedral collision detection. **ACM Transactions On Graphics (TOG)**, ACM, v. 17, n. 3, p. 177–208, 1998.

OKUYAMA, I. F.; MAXIMO, M. R.; CAVALCANTI, A. L.; JM, R. Nonlinear grey-box identification of a differential drive mobile robot.

OTTE, M.; CORRELL, N. C-forest: Parallel shortest path planning with superlinear speedup. **IEEE Transactions on Robotics**, IEEE, v. 29, n. 3, p. 798–806, 2013.

PALMIERI, L.; ARRAS, K. O. A novel rrt extend function for efficient and smooth mobile robot motion planning. In: IEEE. **Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on**. [S.l.], 2014. p. 205–211.

PARK, J. J. Graceful navigation for mobile robots in dynamic and uncertain environments. 2016.

PINTO, M. R. O. A. M. S. C.; BRUNO, M. G. S. Comparison of nonlinear stochastic filters performance for differential drive robots localization.

REEDS, J.; SHEPP, L. Optimal paths for a car that goes both forwards and backwards. **Pacific journal of mathematics**, Mathematical Sciences Publishers, v. 145, n. 2, p. 367–393, 1990.

URMSON, C.; SIMMONS, R. Approaches for heuristically biasing rrt growth. In: IEEE. **Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on**. [S.l.], 2003. v. 2, p. 1178–1183.

WEBB, D. J.; BERG, J. van den. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In: IEEE. **Robotics and Automation (ICRA), 2013 IEEE International Conference on**. [S.l.], 2013. p. 5054–5061.

YERSHOVA, A.; LAVALLE, S. M. Improving motion-planning algorithms by efficient nearest-neighbor searching. **IEEE Transactions on Robotics**, IEEE, v. 23, n. 1, p. 151–157, 2007.

ZICKLER, S.; VELOSO, M. Efficient physics-based planning: sampling search via non-deterministic tactics and skills. In: INTERNATIONAL FOUNDATION FOR AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS. **Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1**. [S.l.], 2009. p. 27–33.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO Nº	4. Nº DE PÁGINAS
TC	November, 17th, 2017	DCTA/ITA/DM-018/2017	52

5. TÍTULO E SUBTÍTULO:

Trajectory Planning Considering Acceleration Limits for an Autonomous Soccer Player

6. AUTOR(ES):

Igor Franzoni Okuyama

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica – ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Palavras chave do autor..

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Palavras chave resultantes..

10. APRESENTAÇÃO: (**X**) **Nacional** ( ) **Internacional**

Trabalho de Graduação, ITA, São José dos Campos, 2017. 52 páginas.

11. RESUMO:

Futebol de robôs é uma competição de robôs autônomos com regras similares ao jogo com humanos, isto é, ganha quem fizer mais gols. Dessa forma, a competitividade de um time está diretamente ligada à habilidade de seus componentes. No âmbito de robôs autônomos, a habilidade está ligada aos movimentos gerados por um algoritmo de estratégia e pela capacidade de executar fielmente os comandos advindos dela. Assim, a velocidade máxima de cada jogador é essencial para definir uma estratégia mais eficiente, pois isso contribui para uma maior posse de bola e ataques mais agressivos. Contudo, ao exigir velocidades maiores dos robôs, efeitos dinâmicos como limites de aceleração devem ser levados em conta tanto no planejamento dos movimentos, quanto da execução dos mesmos. Este trabalho visa estudar a inclusão de características dinâmicas do robô durante a fase de planejamento de trajetórias e as possíveis vantagens em relação ao método clássico de utilizar apenas as características cinemáticas do sistema. Experimentos são feitos tanto em simulação, quanto em testes nos robôs reais da equipe de robótica ITAndroids, de forma a demonstrar da viabilidade da técnica proposta.

12. GRAU DE SIGILO:

(**X**) **OSTENSIVO** ( ) **RESERVADO** ( ) **SECRETO**