UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

VINÍCIUS BRITO CARDOSO

A Model-Predictive Motion Planner for the IARA Autonomous Car

Vitória, ES
2017

VINÍCIUS BRITO CARDOSO

A Model-Predictive Motion Planner for the IARA Autonomous Car

Thesis submitted to the *Programa de Pós-Graduação em Informática* of *Centro Tecnológico of Universidade Federal do Espírito Santo,* in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Advisor: Prof. Dr. Alberto Ferreira De Souza
Co-advisor: Prof^{a}. Dr^{a}. Claudine Badue

Vitória, ES
2017

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Setorial Tecnológica,
Universidade Federal do Espírito Santo, ES, Brasil)
Sandra Mara Borges Campos – CRB-6 ES-000593/O

Cardoso, Vinícius Brito, 1992-

C268m A model-predictive motion planner for the IARA autonomous car / Vinícius Brito Cardoso. – 2017.

72 f. : il.

Orientador: Alberto Ferreira De Souza.

Coorientador: Claudine Badue.

Dissertação (Mestrado em Informática) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Veículos autônomos. 2. Robótica. 3. Controle preditivo. 4. Navegação de robôs móveis. 5. Movimento – Planejamento. 6. Planejamento de trajetória. I. Souza, Alberto Ferreira de. II. Badue, Claudine. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

A Model-Predictive Motion Planner for the IARA Autonomous Car

Vinícius Brito Cardoso

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 23 de novembro de 2017:

[illegible]
Prof. Dr. Alberto Ferreira De Souza
Orientador

Claudine Badue
Profª. Drª. Claudine Santos Badue Gonçalves
Coorientador

[illegible]
Prof. Dr. Thiago Oliveira dos Santos
Membro Interno

[illegible]
Prof. Dr. Fernando Santos Osório
Membro Externo

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória-ES, 23 de novembro de 2017.

ACKNOWLEDGEMENTS

First of all, I would like to thank God for the blessing of life, and for giving me strength in all moments.

Secondly, I would like to thank my parents, Egídio and Jirlene, for always giving me the unconditional support that was so important during this journey, for their life example that guided me till here, and for all the love that I will always try to reciprocate. My brothers for all of their support. My wonderful wife Jessica, for her patience and support throughout this time, and for understanding the moments when I was not able to be by her side. To my grandparents, uncles, and aunts, thank you!.

I would like to thank my advisor Prof. Dr. Alberto Ferreira de Souza and my co-advisor Profª. Drª. Claudine Badue for the friendship, patience, guidance, teachings, suggestions and orientation in my academic trajectory as well as in the elaboration of this work. Also I would like to thank Prof. Dr. Thiago Oliveira dos Santos, for reviews and knowledge during this time. Additionally, for all of their support for the LCAD to achieve its goals.

I would like to thank my colleagues and friends of LCAD that I made during this journey, and were very important for my personal and professional development.

I would like to thank my friends from home in Bahia, who, even from a distance, supported me and helped me conclude this stage and climb one more step on this ladder.

I would like to thank the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) for the scholarship.

I thank everyone who directly or indirectly contributed to the completion of my Master's degree.

Thank you very much!

“Apply thine heart unto instruction, and thine ears to the words of knowledge.”

Proverbs: 23:12

RESUMO

Neste trabalho, apresentamos o Model-Predictive Motion Planner (MPMP) da Intelligent Autonomous Robotic Automobile (IARA). IARA é um carro totalmente autônomo que usa um planejador de caminho para computar um caminho da sua posição atual até um destino desejado. Usando esse caminho, sua posição atual, um destino e um mapa, o MPMP pode computar trajetórias suaves da sua posição atual até o destino em menos de 50 ms. MPMP computa as poses dessas trajetórias de forma que elas sigam o caminho desejado e, ao mesmo tempo, estejam a uma distância segura de eventuais obstáculos. Nossos experimentos mostraram que o MPMP é capaz de computar trajetórias que seguem precisamente o caminho produzido por um motorista humano (com distância de 0.15 m em média) enquanto dirige suavemente a IARA a velocidades de até 32.4 km/h (9 m/s).

ABSTRACT

In this work, we present the Model-Predictive Motion Planner (MPMP) of the Intelligent Autonomous Robotic Automobile (IARA). IARA is a fully autonomous car that uses a path planner to compute a path from its current position to the desired destination. Using this path, the current position, a goal in the path and a map, IARA's MPMP is able to compute smooth trajectories from its current position to the goal in less than 50 ms. MPMP computes the poses of these trajectories so that they follow the path closely and, at the same time, are at a safe distance from occasional obstacles. Our experiments have shown that MPMP is able to compute trajectories that follow precisely a path produced by a human driver (distance of 0.15 m in average) while smoothly driving IARA at speeds of up to 32.4 km/h (9 m/s).

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

CARMEN	Carnegie Mellon Robot Navigation Toolkit
DARPA	Defense Advanced Research Projects Agency
EKF	Extended Kalman Filter
IARA	Intelligent Autonomous Robotic Automobile
IPC	Inter Process Communication
LCAD	*Laboratório de Computação de Alto Desempenho* (High Performance Computing Laboratory)
LiDAR	Light Detection and Ranging
MCL	Monte Carlo Localization
MPMP	Model-Predictive Motion Planner
RDDF	Route Data Definition File
SAE	Society of Automotive Engineers
SLAM	Simultaneous Localization And Mapping
TCP	Trajectory Control Parameters
TLT	Trajectory Look-up Table
UFES	*Universidade Federal do Espírito Santo* (Federal University of Espírito Santo)

CONTENTS

1 INTRODUCTION

Autonomous car technology has made great progress in recent years. The search for improvements in traffic safety, urban mobility and sustainable alternatives are some of the goals of this technology. In regards to traffic safety, according to the World Health Organization (WHO), 1.25 million people die each year in traffic accidents, mostly caused by human failure [WHO15]. Initiatives around the world seek to reduce this statistic. The autonomous car sensing and control technology can save thousands of lives and prevent humans from exposing themselves to dangerous activities [BUE09]. In regards to urban mobility, autonomous cars drive more precisely and need less space between them, which can optimize the traffic flow. In addition, autonomous cars can enable people who can't or don't want to drive to safely ride without needing a driver. In regards to sustainable alternatives, autonomous car control is more stable, which can reduce fuel consumption, gases emission and less wear of parts.

The research on autonomous vehicles has been happening for many years, with the first cars considered autonomous developed in 1977 [SHA13]. But the challenges of the Defense Advanced Research Projects Agency (DARPA) have become known as the accelerator of autonomous car development. The first challenges in 2004 and 2005, called the "Grand Challenge", aimed at the development of autonomous vehicles capable of traveling along a desert route, through dirt roads and rocky terrain. In the first year, none of the 15 finalists completed the challenge; in the second year, five of the 23 finalists completed the challenge, with the Stanford Race Team with the vehicle Stanley winning the challenge [BUE07]. In 2007, a new challenge was proposed, the "Urban Challenge". In this challenge, the cars should travel through a route of 96 km in urban ways, dealing with moving traffic and traffic rules, negotiating busy intersection and avoiding obstacles. Six of the eleven participants completed the challenge and the winning team was the Tartan Race Team, developed by the Carnegie Mellon University [BUE09].

From there on, there was an increase of interest by the industry as well as a rise of new researchers and companies interested in autonomous vehicles. In order to simplify communication and collaboration, the Society of Automotive Engineers (SAE) International [SAE14] divided On-Road Motor Vehicle Automated Driving Systems into six levels:

- Level 0 - No automation. All steering, acceleration and braking commands are made by a human driver, as well as the monitoring of the environment.
- Level 1 - Driver assistance system. In this level, the system is capable of partially assisting the braking or acceleration of the vehicle, but without monitoring the environment.
- Level 2 - Partial automation. In this level, the execution of steering, acceleration and braking can be executed by the system, but it still depends on the human driver to monitor the environment and take control.
- Level 3 - Conditional automation. In a specific scenario, both the execution of steering, acceleration or deceleration, as well as monitoring of the environment, can be made by the assistance system. However, the human driver must be alert and intervene or take control in situations not contemplated by the assistance system.
- Level 4 - High Automation. In a specific scenario, the system is able to drive the car and also handle the situation when the human driver does not respond appropriately to the intervention request.
- Level 5 - Full automation. The system is responsible for all aspects of driving in all scenarios, requiring no intervention from a human driver.

Many applications of autonomous cars seeking level 5 of autonomy are not for sale yet, but are already being tested by companies, such as Google[1], Mercedes[2] [ZIE14], Uber[3], Volvo[4], among others. But, cars near the level 3 of autonomy are already available for sale,

[1] www.google.com/selfdrivingcar/
[2] www.mercedes-benz.com/en/mercedes-benz/innovation/autonomous-driving/
[3] www.uber.com/cities/pittsburgh/self-driving-ubers/
[4] www.volvocars.com/intl/about/our-innovation-brands/intellisafe/autonomous-driving

made by companies like TESLA[5], since 2015. Besides companies, university projects have also brought advances on autonomous car research, such as NavLab[6], VisLab[7] and RobotCar[8]. Among Brazilian initiatives, we can cite the following projects: CADU [DIA12] at UFMG, Driving4U [HON10] at UNIFEI and UFJF, CaRINA [FER14] at USP and IARA [MUT16] at UFES.

The *Laboratório de Computação de Alto Desempenho* (LCAD) has developed a fully autonomous car, named *Intelligent Autonomous Robotic Automobile* (IARA), which is in level 4 of autonomy. Figure 1 shows IARA. We have used a Ford Escape Hybrid as hardware platform, and designed and built a full set of software modules that allow its autonomous operation in urban traffic, such as the mapper [MUT16], localizer [VER15] [VER16], path planner, high level decision maker, motion planner [CAR17], obstacle avoider [GUI16], controller [GUI17], traffic-light state detection, among others.

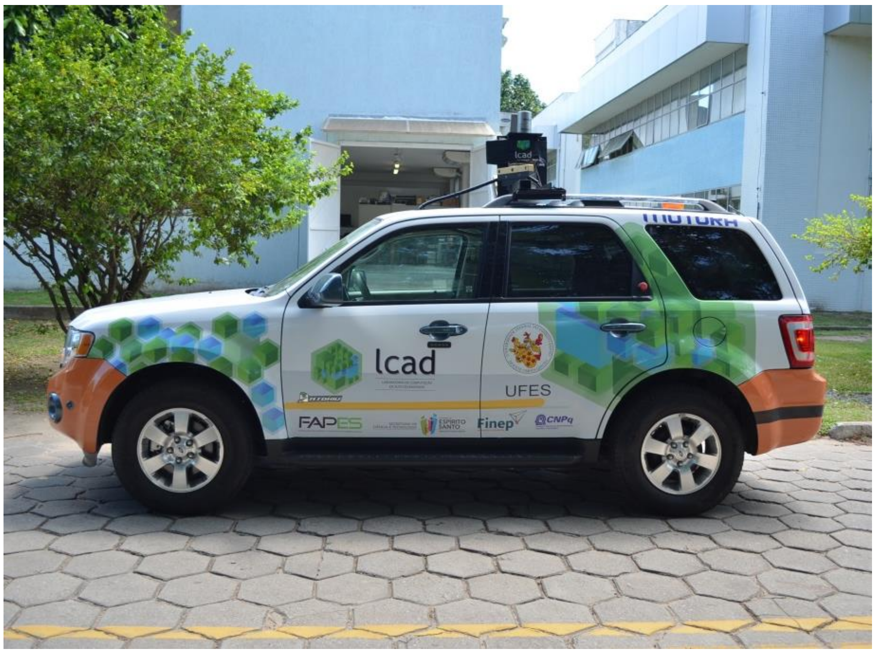


Figure 1: Intelligent Autonomous Robotic Automobile (IARA). A video of IARA's autonomous operation is available at https://goo.gl/RT1EBt. A description about the project is available at http://www.lcad.inf.ufes.br/wiki/index.php/IARA.

[5] www.tesla.com/autopilot
[6] www.cs.cmu.edu/afs/cs/project/alv/www/index.htm
[7] www.vislab.it
[8] www.robotcar.org.uk

For the navigation task, the main IARA's modules are the mapper, localizer, path planner, high level decision maker and motion planner. The mapper [MUT16] is responsible for building a map of the environment around IARA. The localizer [VER15] [VER16] is responsible for estimating IARA's state in relation to the origin of the map. The path planner is responsible for building a path from the initial pose to the end-pose, which obeys restrictions imposed by the road limits and IARA's previously defined maximum operating parameters, such as maximum speed, acceleration, rate of driving wheel turn, etc. The path is composed of poses and associated velocities. The high level decision maker is responsible for establishing a lane and a goal state in it, considering traffic rules and traffic behavior. Finally, the motion planner is responsible for planning a trajectory from the current IARA's state to the goal state, while following the lane and avoiding eventual obstacles. The trajectory is composed of velocity and steering wheel angle commands and associated execution times.

1.1 Problem

In this work, we present IARA's model-predictive motion planner, which receives a map, a lane and a goal state at a rate of 20 Hz, and computes a trajectory from the current IARA's state to a pose and velocity of the lane as close as possible to the goal state, while following the lane and avoiding occasional obstacles.

IARA's model predictive motion planner uses a parameterized model of the behavior of the hardware platform (Figure 8) that, given an initial pose, the goal state, and a set of trajectory control parameters, computes the trajectory. In order to obtain a trajectory, we have to find the right trajectory control parameters. For that, using the conjugate gradient optimization algorithm, we gradually change an initial seed of the trajectory control parameters taken from a pre-computed table indexed by indexes derived from the initial pose and goal state, so that : (i) the final pose and velocity reached is close enough to the goal state according to some optimization criteria; (ii) the poses in the trajectory are as close as possible to the poses in the lane*;* and (iii) the poses in the trajectory are as far as possible of occasional obstacles in the given map.

We have evaluated IARA's model-predictive motion planner (MPMP) experimentally by comparing it with a path followed by a Human driver (i.e., by comparing the distance between MPMP poses and Human poses and the distance between MPMP velocities and Human velocities). Experimental results show that MPMP performance compares well with Human performance – its path is smooth and very close to the Human path (average distance of 0.15, σ= 0.14) and its speeds are more stable than that of the Human driver. Currently, MPMP can safely navigate IARA in urban environments with speeds of up to 32.4 km/h (9 m/s).

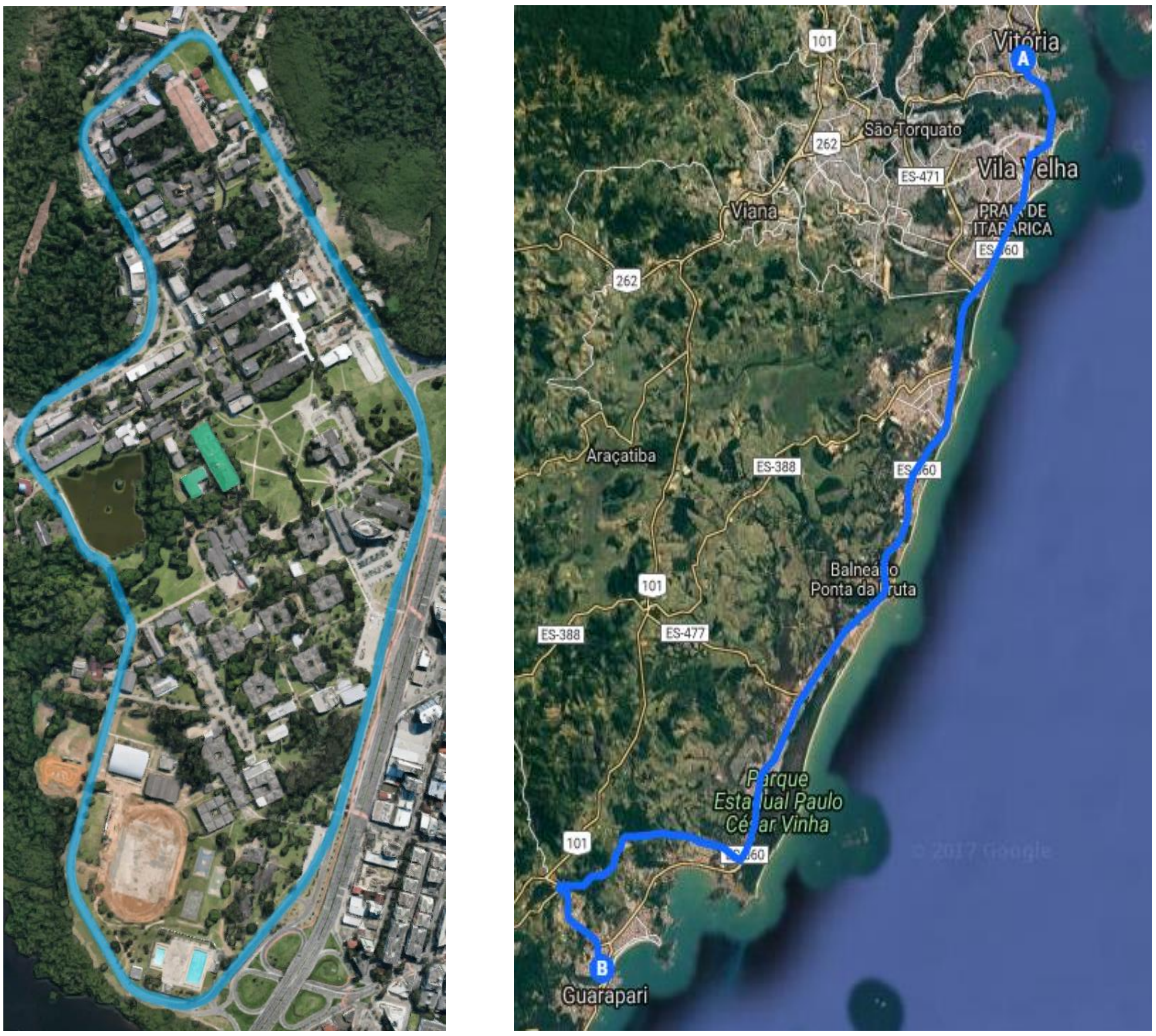
1.2 Motivation

This work is part of the IARA's project, which aims to develop a fully autonomous car. This project has been led by the *Laboratório de Computação de Alto Desempenho* (LCAD) in the *Universidade Federal do Espiríto Santo* (UFES). The laboratory long-term objective is to understand how the human brain works. Through the autonomous car project, we are able to study a dynamic real-world interaction, which requires many brain functions from drivers. Within this main objective, the motivation for this work is to extend the state of the art in the area of motion planning by investigating a new strategy of generating smooth trajectories based on splines of the steering wheel angle.

To evaluate IARA's performance, two challenges were established: the first one is to drive autonomously along the UFES beltway (Figure 2(a)), and the second one is to travel autonomously from UFES in Vitória to Guarapari (Figure 2(b)). The first course is a 3.7 kilometers ring-road that encircles the UFES main campus with urban traffic environment, however, with different road pavements and fewer traffic rules than inner city urban roads. The second course is a 74 kilometers route that crosses three cities. This route has more intense traffic and includes drive through urban environment, highway and dirt road, which require smooth trajectories that can be executed in high speed.

In order to complete these challenges, IARA has to be able to create a map of the environment around it, localize itself in relation to the map and navigate safely, considering traffic rules and obstacles. In navigation, it is necessary to keep a safe and smooth trajectory, achieve higher velocities and operate in real time. Unlike discrete techniques, such as RRT [RAD14], that generate uneven paths [KAT15], the model-predictive planning technique parameterizes the control using a polynomial curve to keep the movement transitions smooth and the optimization process efficient.

In this work, we develop a Model-Predictive Motion Planner (MPMP) for the IARA autonomous car. We evaluate the MPMP performance by comparing its path with that followed by a Human driver. Experimental results show that MPMP is able of planning smooth trajectories that follow a reference path while avoiding eventual obstacles, achieving speeds of up to 32 km/h and operating at a high frequency of 20 Hz.



(a) (b)

Figure 2: (a) UFES beltway. (b) Satellite route from UFES in Vitoria to Guarapari.

1.3 Objective

The objective of this work is to develop a motion planner for the IARA autonomous car, which is able to generate smooth trajectories that follow the road and avoid obstacles, reach velocities higher than 30km/h and operate in real-time.

1.4 Contributions

The main contribution of this work is a motion planner for the IARA autonomous car, which is able to generate smooth and safe trajectories, reach higher velocities and operate in real-time. Our experimental results show that the motion planner presented in this work is able to:

- Compute smooth paths very close to the Human path;
- Generate velocities more stable than that of the Human driver;
- Plan trajectories that obey restrictions imposed by obstacles in the road and IARA's performance limits;
- Achieve speeds of up to 32km/h; and
- Operate at a high frequency of 20 Hz.

1.5 Organization

This dissertation is divided as follows:

- Chapter 2 presents a review of related works.
- Chapter 3 introduces the theoretical background of this work.

- Chapter 4 details the Model-Predictive Motion Planner algorithm, the focus of this work.
- Chapter 5 explains the hardware and software platforms, test environment, test methodology and metrics used in the experiments.
- Chapter 6 describes the experiments carried out to evaluate the MPMP and discusses the results obtained.
- Chapter 7 presents conclusions and directions for future work.

2 RELATED WORKS

In this chapter, we present a review of related works. Among the autonomous car initiatives, the most popular one is the Google Self-driving car, currently developed by Waymo [WAY17], with a fleet that already traveled more than three million miles in autonomous mode. In Brazil the initiatives are being developed since 2009 and are advancing the research area. All these following projects adapted cars with actuators and sensors to allow the autonomous navigation, and tested them in open areas. In the project CADU [DIA12], a Chevrolet Astra navigates using a composition of vector field and dynamic window approaches [DEL13]. In the project Driving4U [HON10], a Chevrolet Zafira navigates using a fuzzy maneuver control approach [HON10]. In the CaRINA project [FER14], a Fiat Palio Adventure navigates based on the state lattice approach [MAG13].

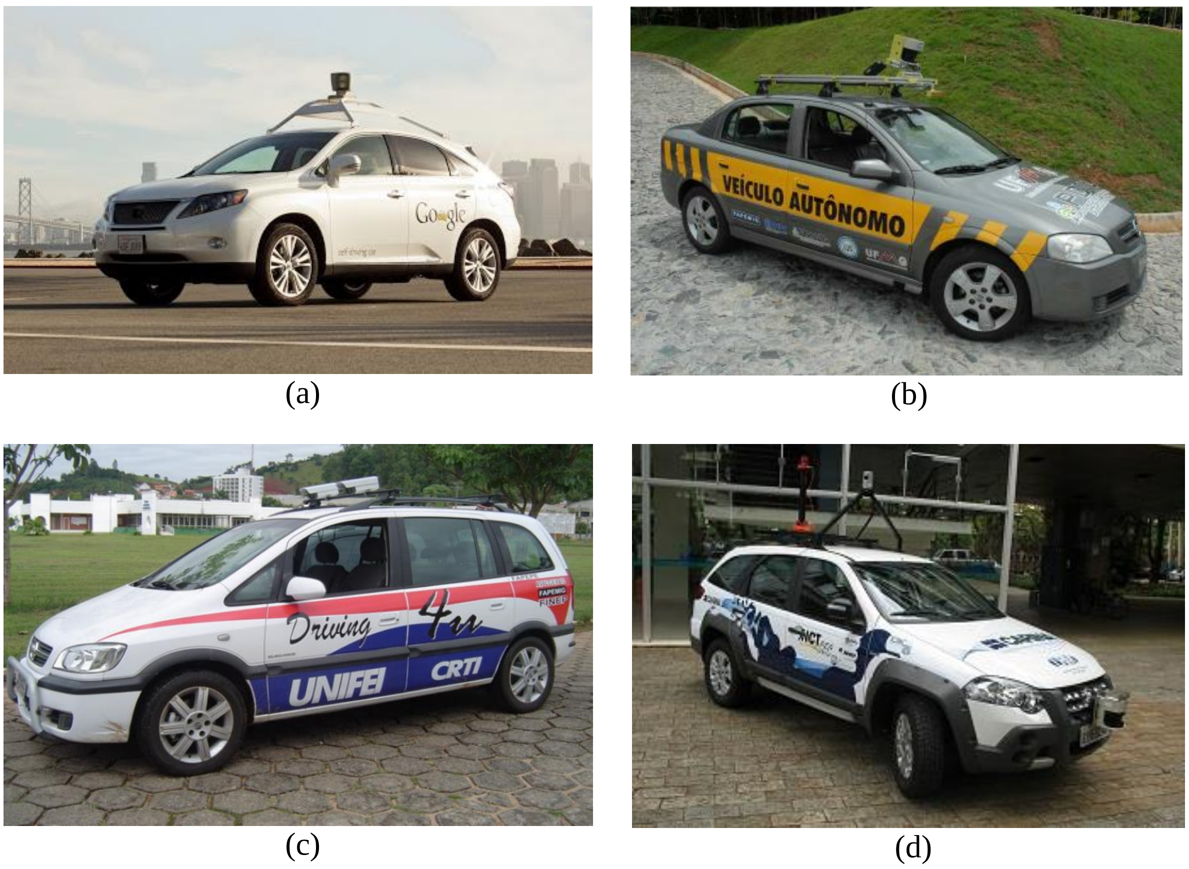
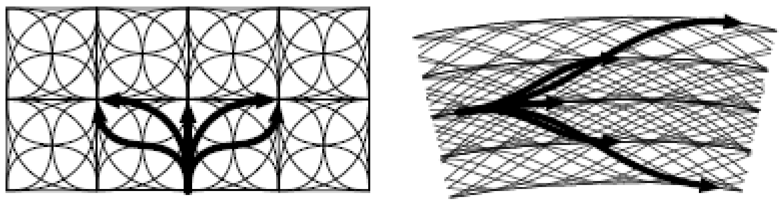


Figure 3: Autonomous cars projects. (a) Google Self-driving car. (b) CADU [DIA12]. (c) Driving4U [HON10]. (d) CaRINA (http://lrm.icmc.usp.br/).

There are several methods in the literature that address the *on-road motion planning problem for autonomous cars*. We refer to the problem of *motion planning* that aims at planning a trajectory – list of commands of velocity and steering angle along with respective execution times – considering kinematic and dynamic constraints of the autonomous car, which contrast to the problem of *path planning*, that aims at planning a path – list of waypoints – considering only kinematic constraints. We also refer to the problem of *on-road motion planning* that aims at planning a trajectory that follows a desired lane, which differs from the problem of *unstructured motion planning* in which there are no lanes and, thus, the trajectory is far less constrained.

Among the works on on-road motion planning that were evaluated experimentally using real-world autonomous car, on-road motion planning methods employ mainly state lattice, rapidly-exploring random tree (RRT), interpolation, optimization, and model predictive techniques [BAU15].

In methods based on state lattice [MCN11] [XU12] [MAG13], trajectories between initial and desired goal states are searched for in a state lattice that is adapted for on road motion planning, such that only states that are a priori likely to be in the solution are represented (Figure 4). Possible trajectories are evaluated by a cost function that considers the car's dynamic, environmental and behavioral constraints, among others. The major disadvantage of these methods is that they are computationally costly due to the evaluation of every possible solution in the graph.



(a) (b)

Figure 4: State lattice representations. (a) In an unstructured environment. (b) In a structured environment such as on-road environments [MCN11].

In methods based on RRT [KUW09] [RAD14], a search tree is built incrementally from the car's initial state using random states. For each random state, a control command is applied to the nearest state of the tree for creating a new state as close as possible to the random state. A trajectory is found when a state reaches the goal state. Candidate trajectories are evaluated according to various criteria (Figure 5). The main weakness of these methods is that solutions are not continuous and, therefore, jerky.

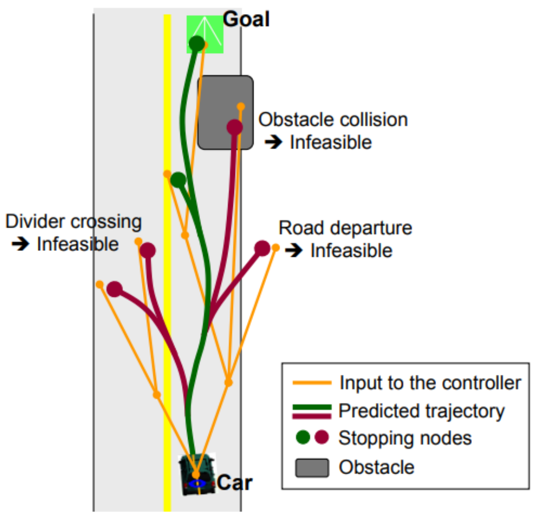


Figure 5: Illustration of RRT trees and trajectory evaluation [KUW09]

In methods based on optimization [KOG06] [ZIE14], trajectories are computed using an optimization process, which is run over trajectory parameters and aims at minimizing an objective function that considers trajectory constraints, such as position, velocity, acceleration, and jerk (Figure 6). The shortcoming of these methods is that they are time consuming since the optimization process takes place at each motion state.

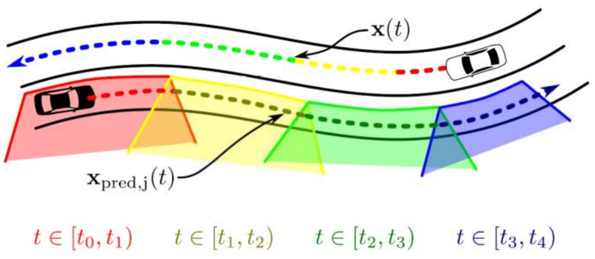
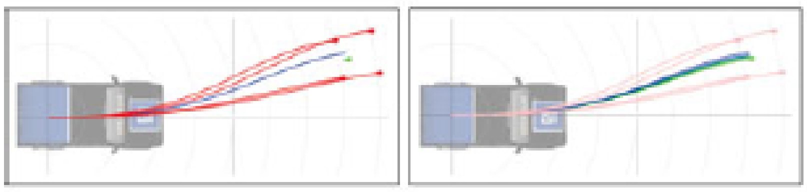


Figure 6: Illustration of a optimization based trajectory (x(t)) considering moving obstacles by [ZIE14].

In model predictive methods [FER08], a model predictive trajectory generation algorithm is used to compute trajectories that satisfy desired goal state constraints. Trajectories are computed via an optimization procedure that is run over control parameters. The constraint equation is defined as the difference between the goal state constraints and the integral of the car's model dynamics. The constrained trajectory generation algorithm determines the control parameters that minimize the constraint equation. An approximate mapping from the state space to the parameter space is precomputed offline, in order to seed the constrained optimization process. Resulting trajectories are evaluated using several criteria (Figure 7).



(a) (b)

Figure 7: Illustration of the trajectory generation process by [FER08]. (a) First a trajectory (in blue) is interpolated from the precomputed trajectories (in red). (b) After that, using the blue trajectory as seed, a new trajectory (in green) is optimized to reach the goal.

Amidst previous works cited above, the most similar to ours is the motion planner proposed by Ferguson *et al*. [FER08]. However, our work differs from that of Ferguson *et al.* in three main aspects. First, our planner is able to compute more complex trajectories, since it uses a steering angle spline with one extra knot point (four in total) to parameterize the car control, while that of Ferguson *et al.* uses a three knot curvature spline. Second, our planner can generate trajectories that are optimized for curvy roads with obstacles, since its cost function considers the desired road as well as obstacles, while that of Ferguson *et al.* generates several alternative trajectories and select a collision-free one. Third, we present the algorithm that we use for computing the table of the trajectory control parameters (**tcp**) seeds.

3 THEORETICAL BACKGROUND

In this chapter, we present the theoretical background related to this work. In Section 3.1, we present definitions of the general motion planning problem. In Section 3.2, we describe the mapping problem, and the representation used for localization and the motion planner. In Section 3.3, we describe the robot localization problem and the method used for this work. In Section 3.4, we describe the path planning problem, and how our reference path is built. In Section 3.5, we describe the general Model-Predictive Planner approach for trajectory generation.

3.1 Motion Planning

A motion planner has to be able to automatically compile the information of a high-level language task, into a set of low-level motion primitives, or feedback controllers. The task normally is to find a collision-free motion for the robot, from one configuration to another [CHO05]. The configuration of the robot or state is its position, orientation and velocities in the environment at an instance of time. A set of all possible states that the robot can be in is called a state space [HOW09]. Here the state is represented as the set $\mathbf{x} = (x,\ y, \theta, v, \varphi, t)$, where $x,\ y$ are the 2-Dimension position; θ the orientation, v the speed, φ the steering wheel angle, and t the time. The initial state is represented by $\mathbf{x_o}$ and $\mathbf{x_g}$ represents the goal state.

To move from one state to another, the robot constraints have to be considered. Nonholonomic robots, as a car-like, are subjected to velocity constraints, and all degrees of freedom aren’t controllable (e.g. a car cannot translate sideways). If no constraints are applied the robot is called holonomic [HOW09] [CHO05]. So, to estimate the new position of a robot from one state to another we need a model that considers these constraints. The model of the robot can be represented by kinematic equations, with velocity as control, or using dynamic equations of motion, with forces as controls [CHO05]. In a car-like, the Ackerman geometry model is commonly applied. With this model, it is possible to estimate the new position of a robot from one state to another, with some precision in low speeds. The Figure 8 shows a modified model representation, also known as bicycle model, where the steering angle φ is approximated by the average between the two front wheels; x and y are the position of the vehicle located in the middle of the rear wheels; θ his orientation; and l is the wheelbase [CHO05][KAT15].

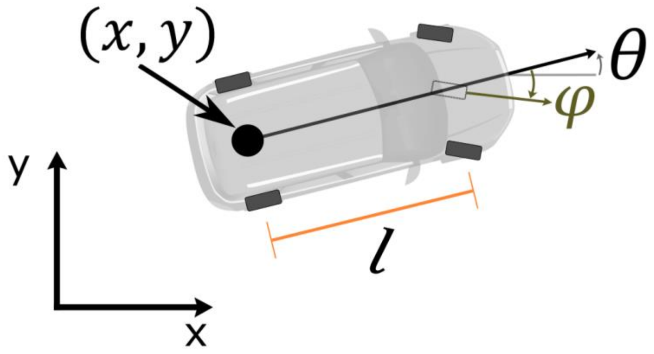


Figure 8: Kinematic model of a car-like vehicle.

The kinematic model of motion is given by the equations:

$$x_{t+1} = x_t + \Delta t\, v_t \cos\theta_t, \quad (1)$$

$$y_{t+1} = y_t + \Delta t\, v_t \sin\theta_t, \quad (2)$$

$$\theta_{t+1} = \theta_t + \Delta t\, \frac{tan\varphi_t}{l} \quad (3)$$

Considering the robot constraints, to move the robot, we need to command an *action*, such as acceleration, steering angle or wheels torques. Common types of actions are controls and controllers. The first uses the action as a function of time, and the second determines inputs as a function of state and time. Controls and controllers can be parameterized to approximate the continuous constraints by a finite number of constraints. For example, parameters could be represented by spline curves or clothoid curves [HOW09] [CHO05]. In this work, the control $\boldsymbol{u}$ is defined by the command of velocity and steering angle during a specific time variation, the time and steering angle control are parameterized as a spline. As shown below.

$$\boldsymbol{u} = [v, \varphi, t] \quad (4)$$

$$\varphi = spline(\Delta t) \tag{5}$$

The motion planner can also be addressed as trajectory planning problem [CHO05]. The set of associated states and times that defines the robot movement from initial state ($\mathbf{x}_o$) to goal state ($\mathbf{x}_g$) is a trajectory (T). The trajectory generated has to satisfy the vehicle's kinematic constraints, consider the road boundaries, and also, at the same time, avoid collision with obstacles. In order to achieve that, good representations of the environment, such as a map, are required. The map, or maps, can be used for localization, obstacle detection, or road boundaries detection, as well as to provide the information about traffic rules. Thus, the decision making process can choose a goal considering traffic rules and eventual obstacles and ask to the motion planner to generate a trajectory to this goal, considering the current localization and obstacles in the map. In case of failure of the trajectory generation, or localization problems, an obstacle avoider module takes control and stops the car.

3.2 Mapping

To properly localize and navigate, the robot needs a representation of the environment where it is. This representation is acquired by sensing and extracting the main features structures of the environment and processing it building a map [ROM14].

According to Thrun et al. [THR05], the main problems to acquire maps with robot mapping is: (i) the dimensionality, in which the more complex the representation, more processing, and storage will be required; (ii) the problem of simultaneous localization and mapping (SLAM), where the localization is necessary to build a map, and a map is necessary to localize the robot, thus the robot has to do both, estimate a map and localize itself in relation to this map; Beyond those, it is possible to cite other problems, such as: size of the environment in relation to the range of the sensors, noise in perception and actuation, and cycles in the environment which increase odometry errors.

The common map representations are feature-based and location-based. In feature-based, the map stores point landmarks, it is common to identify lines, corners, hallways, and intersection. The location-based stores the proprieties of locations of the world, for example, if a region is free, or if there is an obstacle in that location, one example is the occupancy grid map [THR05].

The Occupancy grid mapping is a popular family of algorithms. This approach assumes that the pose of the robot is known, and used to generate consistent maps from noisy and uncertain sensor measurement data [THR05]. To reduce complexity, this map can be a 2-D representation of the 3-D world; the map is a set of cells that describe real-world position, building a grid. Given the robot pose and the measurement data, a probability of map is computed, each cell stores the probability of occupation, which can be represented as a value, for instance, when occupied the value is 1, when free the value is 0, for unknown cells the value is 0.5. Figure 9 below shows an example of a map visualization, where the occupied cells are in black, free cells in white and unknown cells in gray.

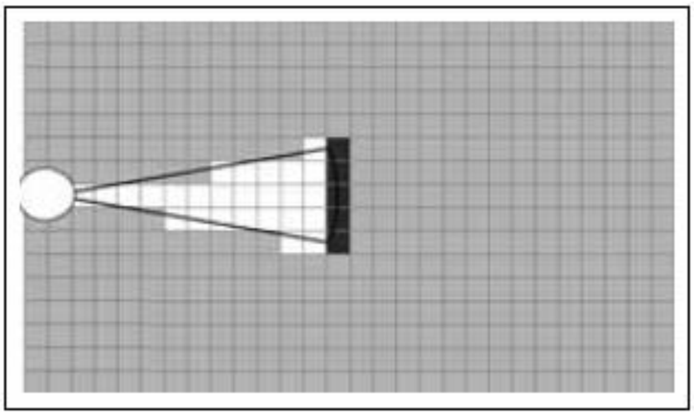


Figure 9: Occupancy grid map cell representation from a measuring range [THR05].

When poses of the robot are unknown, SLAM algorithms are used to estimates the robot pose and build a map. The methods employed to solve this problem are mainly offline, given the high computational cost, like EFK Slam and GraphSlam [MUT16], but there are also technics for online, such as the FastSlam. In GraphSlam example, the odometry and sensor data of the environment are logged, and offline, the poses of the robot are computed to fit the real position of the robot, given the movement and sensors data, and matching actual and posteriors predictions to correct the estimation. After that, a mapper (e.g. occupancy grid mapping) use the predicted poses to build a precise map indicating where the robot was.

In the IARA autonomous car, the global map is an Occupancy grid-map. To build this map, a human drives IARA through the region of interest and these data are logged. Then the Large-Scale Environment Mapping System (LEMS) [MUT16] based on GraphSlam sends the calculated LiDAR point clouds and their poses for the mapping subsystem that outputs an Occupancy grid-map. This process is computed offline. In autonomous navigation mode, this offline occupancy grid map is used to localization and is updated with the real-world data by LiDAR sweep and the Occupancy grid-map algorithm. Figure 10 below, shows the visualization of one part map, where each cell has 0.2 meters discretization and the total size of the image represents 210x210 meters in a real world map. The occupied cells are in black, free cells in white, and unknown cells in blue.

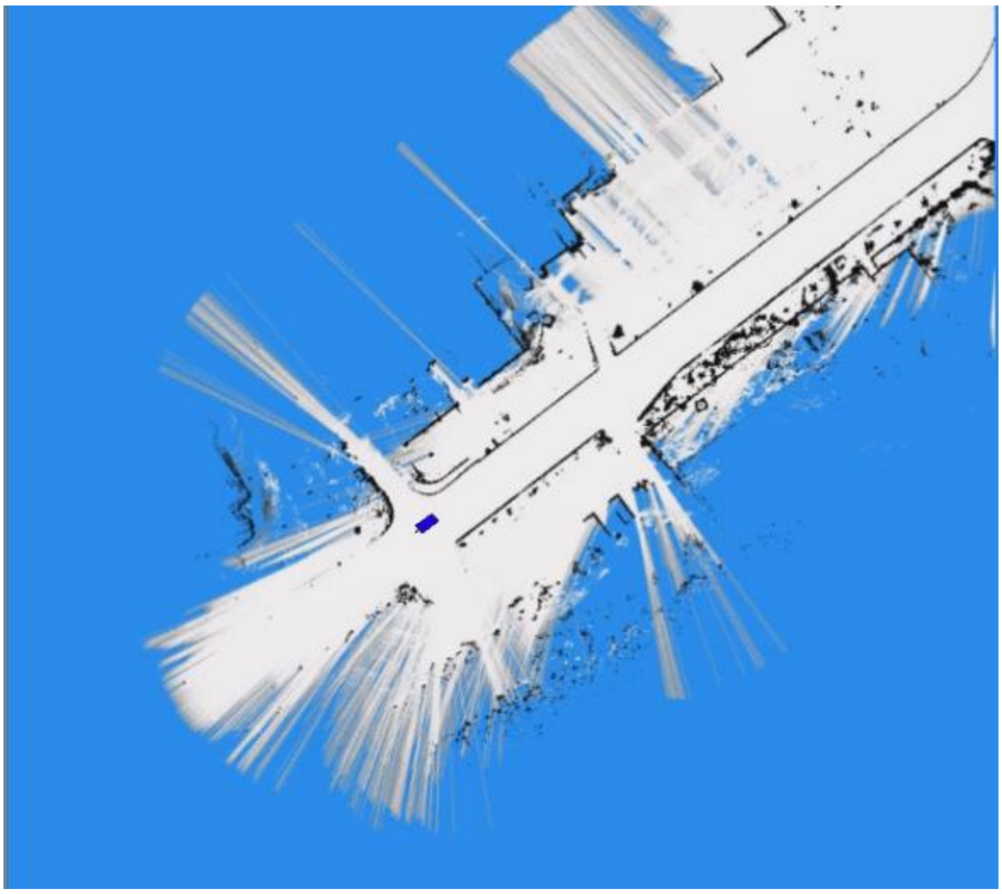


Figure 10: Occupancy grid map with 210x210m and 0.2m of resolution. In this visualization, the blue area is the unknown cells, the occupied cells are in black, and the free cells in white. The blue rectangle is the robot (IARA).

Besides, it is possible to use other representations that will only be used in the motion planning task, helping in the fast collision detection, and reducing the complexity of the search space, for example, Occupancy grid cost maps, state lattices, drive corridors and Voronoi diagrams [KAT15].

3.3 Robot Localization

Localization is an important function in navigation task, responsible for estimating the *state* of the robot. In dynamic environments, the robot has to be able to estimate its own pose relative to a given map of the environment and to deal with the changes in the environment, motion uncertainty, and sensors noise.

The localization problem can be divided in global localization, pose tracking and kidnapped robot problem. In global localization problem, the initial pose of the robot is unknown, normally when initially somewhere in its environment, without knowledge of where it is; the pose tracking problem, assumes that the robot know the global pose, and dealing with the motion noise, track the robot position over time for estimate a precise local pose; the kidnapped robot problem evaluate the robot ability of recover from failures, by simulating that the robot was kidnapped and testing if the robot recover its new pose without the transition information, this is a variant of the global localization problem [THR05].

Localization methods are based on a loop of movement and perception. Each time, the robot senses the environment, and when it moves, using a motion model and the previous information about the environment, tries to estimate the new state [ROM14]. The perception can be done by using for example sensors as LiDAR, cameras or radar, the movement can be updated using the robot odometers and a motion model (Figure 8, Equations (1) to (3)).

To localize the robot, a representation of the environment is required. Normally a map representation is used, as feature-based and location-based (e.g. occupancy grid maps) [THR05]. With a map, mainly approaches applied in autonomous cars use probabilistic methods, for example Extended Kalman filter (EKF) and Particle Filter (PF) [VER16]. These localization methods are based on Bayes filter, which in each time, aims to update the belief of the robot localization based on a probability distribution. The probability density $p(\mathbf{X}_t|z_t, \boldsymbol{u}_t)$ translate the belief of the robot about the possible pose $\mathbf{X}_t$ that the robot can be at time t, taking into account the measurements of sensors z_t and applied commands $\boldsymbol{u}_t$. Due to sensor uncertainties, the robot's belief in its measurements z_t and its movement, are represented by probabilistic models [ROM14]. In robotics localization problem, the application of the Bayes filter is called Markov localization [THR05].

The Extended Kalman Filter algorithm applies Kalman Filter to the localization problem, and extends it to nonlinear problems, approximating the state of the system, whose probability density is Gaussian [THR05]. EKF predicts the new pose of the robot using the knowledge of the previous pose and the sensor measurements, finding the parameters of mean and covariance matrix of the estimation error. Alternating steps of prediction and correction [THR05] [ROM14]. The EKF application typically uses feature map as can be seen in [LYR15].

The Particle filter approach instead of representing the pose with a single hypothesis uses many samples that represent the probability density of the robot's current pose. In robot localization, the particle filter applies the principle of the Monte Carlo techniques, where the simulation of phenomena is used to estimate parameters in the model. This approach gave rise to the Monte Carlo Localization (MCL) technique [ROM14]. In MCL, the localization of the robot is initialized with initial particles sampled from a uniform distribution in the free area of the map, when the robot moves and observes new areas, its belief about its location is corrected. Each particle uses a motion model to predict the new pose after the initialization, based on the last pose and the applied command ($\boldsymbol{u}$) with addition Gaussian noise, creating assumptions of possible new poses. To correct the predicted pose, a weight is computed to each particle based on the measurements of sensors (e.g. LiDAR data) compared with the map information. The particles are drawn by importance, where the set of particles that represents the belief, is those with high weights [THR05]. The algorithm stays in a loop between the prediction, correction and resampling steps. In the course of time, the most likely position for the robot to be will be where the particles concentrate.

The approach applied in IARA use a GPS data as an initial global localization, and pose tracking based on a Particle Filter localization approach with Occupancy grid map and online LiDAR data. After the initialization, the GPS information is no longer needed [VER16]. The

PF approach uses the odometry data for prediction phase, and applies map-matching with the offline global map (Section 3.2) and the currently local map. With this technique, the MPMP is able to receive the state of the robot at 20Hz and update the plan.

3.4 Path Planning

The path planner differs from motion planner in the addition of the time and velocities in the model, where the problem of finding the sequence of states from initial state to a goal state is considered only geometric [ROM14]. Thus, the task of a path planning is to generate a geometric *path*, considering a simplified model of the robot constraints, from an initial state to a desired final goal state, without taking into account the time and velocities. Commonly used algorithms are graph searching algorithms like A* [LIK09] and Field D* [FER05]. Among those, for on-road navigation, a path can be extracted using the information about the road lanes [ZIE14] [FER08] or using a Route Data Definition File (RDDF) [THR06]. The generated path is used for the motion planner as a reference to restrict the state space and follow the traffic rules.

In this work, we use an RDDF to build a global plan. This file contains waypoints that specify the route for IARA. This method stores a set of poses generated by a human driver and corrected by the SLAM algorithm (LEMS [MUT16]). Therefore, a path will be processed and will become the reference path for the motion planner. The path represents the lane centerline. With this approach, we avoid problems with lane marking detection, for example, in unpaved roads, poor lane marking, and noisy paths. The RDDF guarantee a path trusty and fast to access. Figure 11 shows, as lane visualization, a stretch of RDDF post-processed between the car and an end-goal defined by the user. For visualization purposes, the lane is an estimation of the lane width where the RDDF path is the centerline.

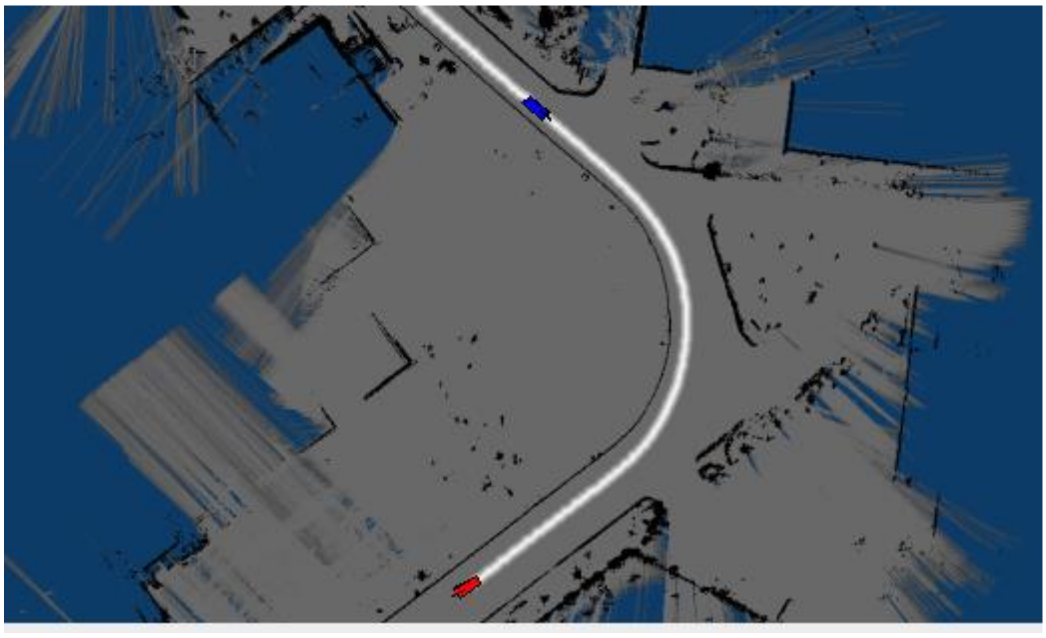


Figure 11: Lane map visualization estimated from centerline given by the RDDF path. The lane is shown in white. The blue rectangle represents the robot (IARA) and the red rectangle represents the end-pose.

3.5 Model-Predictive Planner

A model-predictive planner is an approach used to solve the problem of generating a set of parameterized control commands to take a robot from an initial state to a goal state. The set of parameterized control commands represent the actions performed by the robot along the way. A predictive motion model maps the actions to state response. To estimate the control parameters, an optimizer is applied, and then the best parameters that approximate the estimated final state to goal state are computed [HOW09].

The choice of action parameterization directly impacts the efficiency of the system. It is necessary to take into account the boundary state constraints and motion model used, to avoid for example indeterminate solutions, minimum local and sudden control variations.

The predictive motion model provides the estimative of the change of state of the robot given the applied actions. Therefore, higher the fidelity of this model, better the prediction and stability of the system, but also higher the computational complexity, especially in real-time applications such as on-road navigation.

The optimization process aims to estimate the best parameters that minimize the cost of the predicted state reaches the goal state. The cost function can consider the pose, the proximity to the reference path, and others constraints. To reduce the all possible parameters values, normally a precomputed set of parameters seeds the optimizer initial parameters.

Model Predictive Planning (MPP) algorithm is able to generate feasible solutions to execute, thanks to the addition of the prediction model and compensation. In this work, we propose a Model-Predictive Motion Planner (MPMP) for the IARA autonomous car. Using the map, the current position by a precise localization, the path from its current position to the desired destination and a goal in the path, IARA's MPMP is able to compute, in real time, smooth trajectories that follow the path closely and, at the same time, are at a safe distance from obstacles.

In this work, the representation to obstacle detection is a Distance Map. With the Occupancy grid map and the online sensor data, the IARA's mapping system build a Distance Map, where each cell, instead of the probability of occupation, stores the distance to the nearest obstacle detected in the occupancy grid map. So, it is possible to check the nearest obstacle to the car only consulting the cell's data of interest. With Distance Map, we are able to perform faster collision detection, which will be considered by the optimization process of MPMP trajectory generation. This map is updated at 20Hz. In that way, when we specify a *map*, we refer to the Distance Map.

4 IARA'S MODEL-PREDICTIVE MOTION PLANNER

In this chapter, we present the proposed system. In Section 4.1, we describe an overview of the system. In Section 4.2, we describe the parameterization of the hardware platform model. In Section 4.3, we describe the minimization problem of the optimization process, detailing the process of obtaining the costs function. In Section 4.4, we describe the general algorithm of the IARA's Model-Predictive Motion Planner. Finally, In Section 4.5, we describe how is computed ours Trajectory Look-up Table to seed the optimization process.

4.1 System Overview

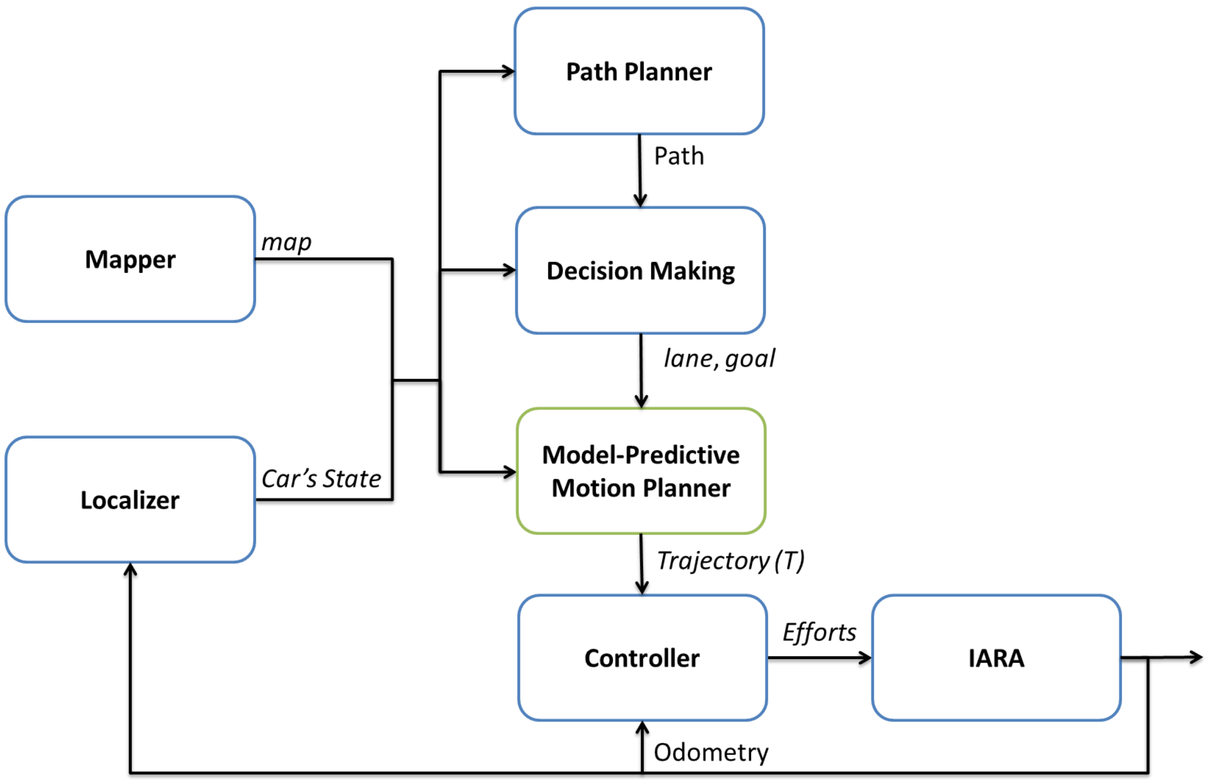


Figure 12: Block diagram of the IARA main navigation modules.

In a typical mission, IARA is at an initial pose (*origin*), $\mathbf{p}_o = (x_o, y_o, \theta_o)$, and the user defines an end-pose, $\mathbf{p}_e = (x_e, y_e, \theta_e)$, and asks it to go from $\mathbf{p}_o$ to $\mathbf{p}_e$. Once asked to perform such a mission, IARA's path planner builds a path, $P = (\mathbf{pv}_o, \dots, \mathbf{pv}_i, \dots, \mathbf{pv}_e)$, from $\mathbf{p}_o$ to $\mathbf{p}_e$, composed of a vector of poses, about 0.5 m apart, and associated velocities, $\mathbf{pv}_i = (x_i, y_i, \theta_i, v_i)$, that goes from $\mathbf{pv}_o = (x_o, y_o, \theta_o, v_o)$ to $\mathbf{pv}_e = (x_e, y_e, \theta_e, v_e)$, while obeying restrictions imposed by the road limits and the platform's previously defined maximum operating parameters, such as maximum speed, acceleration, rate of driving wheel turn ($d\varphi/dt$), etc. IARA's high level decision making module periodically (20 Hz) slices this path and takes a small portion of it of about 100 m that we call *lane*, or $L = (\mathbf{pv}_1, \cdots, \mathbf{pv}_g, \cdots, \mathbf{pv}_{|L|})$, and establish a *goal* in it, $\mathbf{pv}_g = \left(x_g, y_g, \theta_g, v_g\right)$, about 5s in front of IARA's current position. To choose this *goal*, the high level decision making module considers traffic rules and eventual obstacles.

IARA's model-predictive motion planning module, receives a map, a lane and a goal at a rate of 20 Hz, and computes a trajectory, $T = (\mathbf{x}_o, \dots, \mathbf{x}_i, \dots, \mathbf{x}_f)$, from the current car state, $\mathbf{x}_o = (x_o, y_o, \theta_o, v_o, \varphi_o, t_o)$, to a pose and velocity as close as possible to $\mathbf{pv}_g$, while following the lane and avoiding occasional obstacles (Figure 13, Figure 14). The value φ_o in $\mathbf{x}_o$ is the front-wheels' steering angle of the current car state (Figure 8).

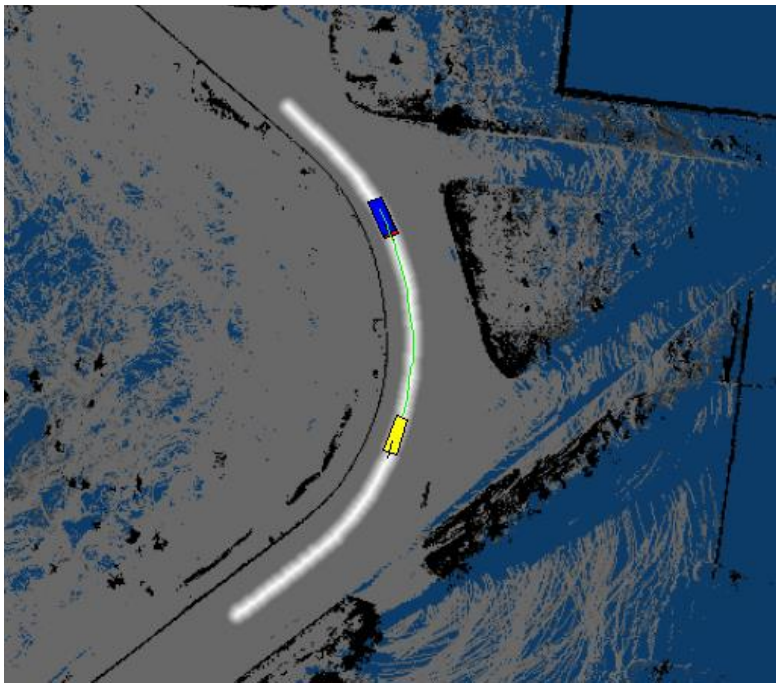


Figure 13: IARA's map, lane and motion plan. IARA is the blue rectangle, while the goal is the yellow rectangle. The lane (L) is shown in white and the trajectory (T) in green. The image is in the cells map low resolution.

In Figure 13, the trajectory seems irregular. However, it is not. It seems irregular because this representation, which is very useful for us, has the resolution of the map. This representation in the resolution of the map is important to analyze how the car interacts with the map, which is fundamental for security. The trajectory is very smooth, as shown in Figure 14.

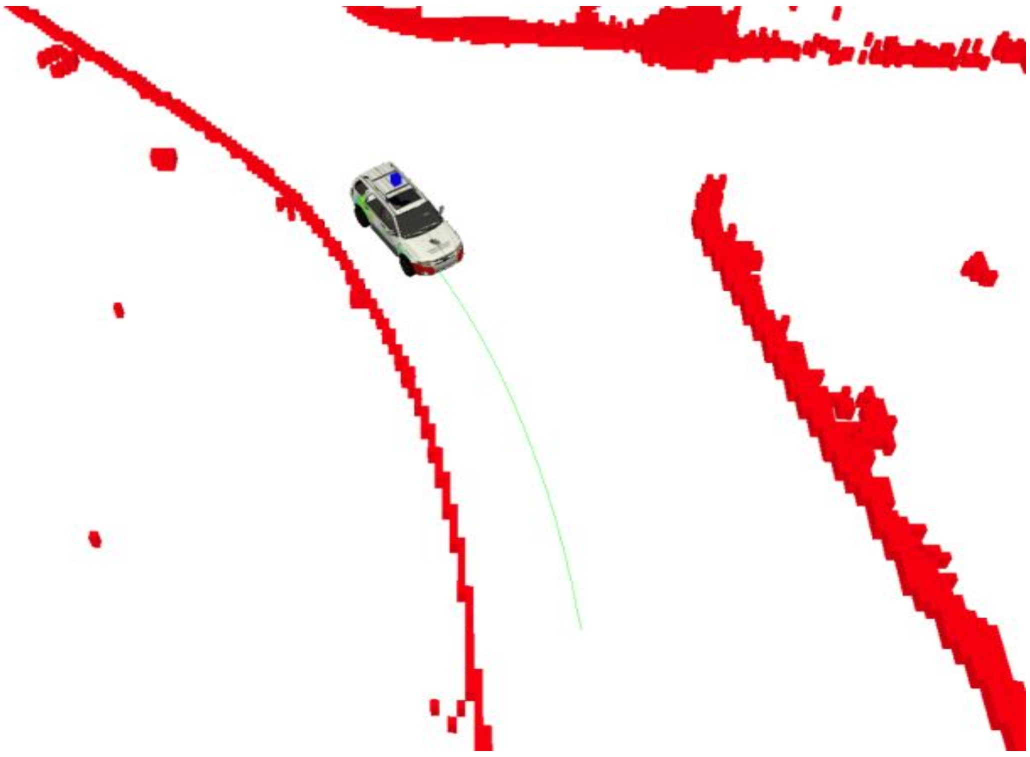


Figure 14: 3D representation in better resolution of IARA, map, goal and motion plan. The trajectory T is shown in green and the goal is at the end of the trajectory.

IARA's model predictive motion planner uses a parameterized model of the behavior of the hardware platform, $M(.)$, that, given an initial state, $\mathbf{x}_o$, the goal, $\mathbf{pv}_g$, and a set of *trajectory control parameters*, $\mathbf{tcp}$, computes the trajectory $T = M(\mathbf{x}_o, \mathbf{pv}_g, \mathbf{tcp})$. In order to obtain T, we have to find the right $\mathbf{tcp}$. For that, using the conjugate gradient optimization algorithm, we gradually change an initial $\mathbf{tcp}$ seed taken from a pre-computed table indexed by indexes derived from $\mathbf{x}_o$ and $\mathbf{pv}_g$, so that: (i) $\mathbf{pv}_f = (x_f, y_f, \theta_f, v_f)$, composed of the final pose and velocity at state $\mathbf{x}_f$, is close enough to $\mathbf{pv}_g$ according to some optimization criteria; (ii) the poses in T are as close as possible to the poses in L; and (iii) the poses in T are as far as possible of occasional obstacles in the given map, map.

4.2 Hardware Platform Model

IARA's model-predictive motion planner uses a parameterized model of the behavior of the hardware platform, $M(.)$, that, given an initial state, $\mathbf{x}_o = (x_o, y_o, \theta_o, v_o, \varphi_o, t_o)$, the goal, $\mathbf{pv}_\mathrm{g} = (x_g, y_g, \theta_g, v_g)$, and a set of trajectory control parameters, **tcp**, computes a trajectory $T = M(\mathbf{x_o}, \mathbf{pv}_g, \mathbf{tcp})$, which is a vector of states, $T = (\mathbf{x_o}, \dots, \mathbf{x_i}, \dots, \mathbf{x_f})$. The elements of **tcp** are the total time of the trajectory, tt, and three knot points, k_1, k_2 and k_3, $\mathbf{tcp} = (tt, k_1, k_2, k_3)$. The three knot points, together with φ_o, define a cubic spline that specifies the evolution of the steering angle during tt (Figure 15).

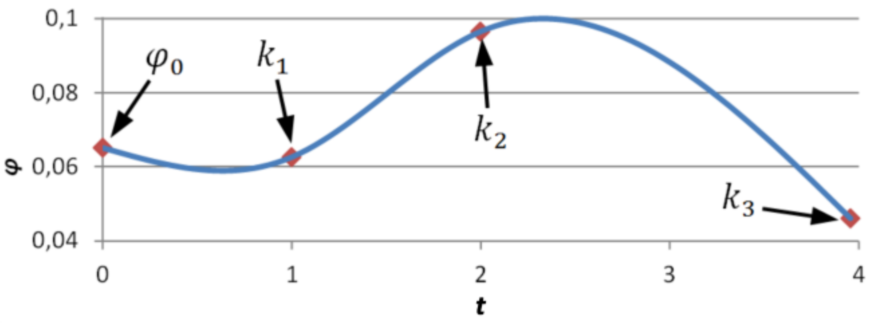


Figure 15: Cubic spline, defined by φ_0, and k_1, k_2 and k_3, that specifies the evolution of the steering angle during tt, for tt = 3.96 s. The knot point k_1 is defined at $\mathrm{t} = \mathrm{tt}/4$, k_2 at $\mathrm{t} = \mathrm{tt}/2$ and k_3 at $\mathrm{t} = \mathrm{tt}$.

Our car model, $M(.)$, is currently a bicycle kinematic model (Figure 8) modified to consider an understeer coefficient, u, and is defined by the equations:

$$x_{t+1} = x_t + \Delta t\, v_t \cos\theta_t, \tag{6}$$

$$y_{t+1} = y_t + \Delta t\, v_t \sin\theta_t, \tag{7}$$

$$\theta_{t+1} = \theta_t + \Delta t\, v_t c_t \tag{8}$$

$$v_{t+1} = v_t + \Delta t\, a \tag{9}$$

$$\varphi_{t+1} = spline(\Delta t(t+1)), \text{and} \tag{10}$$

$$c_t = \frac{\tan\dfrac{\varphi_t}{1 + u v_t^2}}{l} \tag{11}$$

where c_t is the car curvature that is directly used to control the car [TOR10]. The relationship between φ_t and c_t, given by Equation (11), together with equations (6) to (10), constitute a simplification of the full Ackerman car model and it was obtained experimentally [CUR08]. For our hardware platform, the understeer coefficient, u, is equal to 0.0015 [TOR10]. This car model can be improved in the future without a significant impact on the remainder of the planner.

To obtain T, $M(.)$ is interactively used for computing each state, $\mathbf{x}_{t+1}$. During each planning cycle the model starts with $t = 0$ and $a = (v_g - v_0)/tt$, and it is run from t = 0 to $t = tt/\Delta t$. The velocity of the car evolves according to the acceleration (a), so that the velocity in the end of the trajectory is the desired velocity for that pose.

4.3 Minimization Problem

At a rate of 20 Hz, our model-predictive planner computes a trajectory, T, that starts at the current car state, $\mathbf{x}_o = (x_o, y_o, \theta_o, v_o, \varphi_o, t_o)$ and finishes at a state, $\mathbf{x}_f = (x_f, y_f, \theta_f, v_f, \varphi_f, t_f)$, whose pose and velocity, $\mathbf{pv}_f = (x_f, y_f, \theta_f, v_f)$, is as close as possible to a given goal pose and velocity, $\mathbf{pv}_g = (x_g, y_g, \theta_g, v_g)$. However, it has to do that while keeping the car at a safe distance from obstacles and as close as possible to the poses of the given lane $L = (\mathbf{pv}_1, \cdots, \mathbf{pv}_g, \cdots, \mathbf{pv}_{|L|})$, part of the path computed by the IARA's path planner. To compute T, we solve the following minimization problem:

$$\arg\min_{\mathbf{tcp}} f(M(\mathbf{x_o}, \mathbf{pv}_g, \mathbf{tcp}), L, map) \tag{12}$$

where

$$f(.) = \sqrt[2]{w_1 \Delta\lambda^2 + w_2 \Delta\theta^2 + w_3 \Delta\phi^2 + w_4 {D^O}^2 + w_5 {D^L}^2}, \tag{13}$$

and $w_1$to w_5 are weights.

To minimize $f(.)$, we have to find the **tcp** that minimizes the square root of the weighted sum of the squares of the values: (i) $\Delta\lambda$, (ii) $\Delta\theta$ and (iii) $\Delta\phi$, which together, measure the distance from the end of T, $\mathbf{pv}_f$, to the desired goal, $\mathbf{pv}_g$; (iv) D^O, which summarizes the distance of each point in T to its nearest obstacle; and (v) D^L, which summarizes the distances between each point in L and its nearest point in T.

The value $\Delta\lambda$ is the difference between the magnitude of the two vectors, $\mathbf{s}_g = (x_g, y_g)$ and $\mathbf{s}_f = (x_f, y_f)$, that connect the poses associated with $\mathbf{x}_o$ and $\mathbf{pv}_g$ ($\mathbf{s}_g$), and the poses associated with $\mathbf{x}_o$ and $\mathbf{pv}_f$ ($\mathbf{s}_f$), and it is computed by the equation (Figure 16):

$$\Delta\lambda = \|(x_g - x_0, y_g - y_0)\| - \|(x_f - x_0, y_f - y_0)\| \tag{14}$$

The value $\Delta\theta$ is the difference between the car orientation at the goal and the car orientation at the end of the trajectory, and is computed by the equation (Figure 16):

$$\Delta\theta = \theta_g - \theta_f \tag{15}$$

The value $\Delta\phi$ is the angle between $\mathbf{s}_g$ and $\mathbf{s}_f$, and is given by the equation (Figure 16):

$$\Delta\phi = \operatorname{atan2}((y_g - y_o)/(x_g - x_o)) - \operatorname{atan2}((y_f - y_o)/(x_f - x_o)) \tag{16}$$

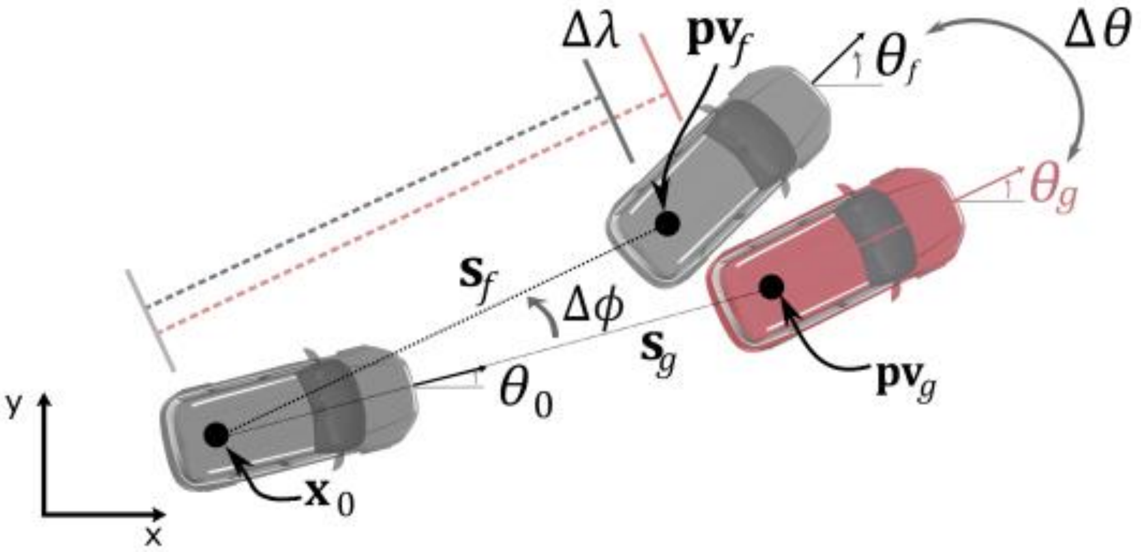


Figure 16: Variables associated with $\Delta\lambda$, $\Delta\theta$ and $\Delta\phi$, which together, measure the distance from the end of the T, $\mathbf{pv}_f$, to the desired goal, $\mathbf{pv}_g$.

The value D^O is the summation of the differences between d^{min} and $\|\mathbf{x}_k - \mathbf{o}_k\|$, where d^{min} is the smallest allowed distance between the car and an obstacle (about half the car width (Figure 17)), and $\|\mathbf{x}_k - \mathbf{o}_k\|$ is the distance between the pose $\mathbf{x}_k$ and its nearest obstacle, $\mathbf{o}_k$, in the distance grid map, map. The pose of $\mathbf{x}_k$, is a displacement of each $\mathbf{x}_i \in T$, where k is distributed in n circles to cover all the car area, and check all obstacles considering the car size. D^O is computed by the equation:

$$D^O = \sum_{i=0}^{f} \sum_{k=0}^{n} max\ \left(0, d^{min} - \|\mathbf{x}_k - \mathbf{o}_k\|\right) \tag{17}$$

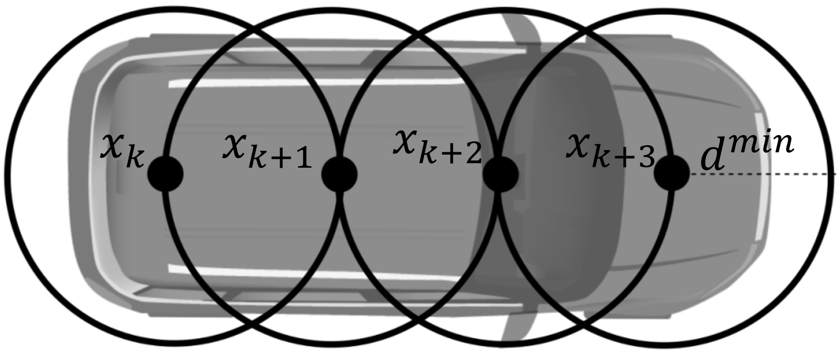


Figure 17: Illustration of the collision detection area where $\mathbf{x}_k$ is the center of the circle and the pose displaced from each $\mathbf{x}_i \in T$; and d^{min} represent the circles radius, that is, the smallest allowed distance between the car and an obstacle.

Finally, the value D^L is the summation of $\|\mathbf{pv}_k - \mathbf{x}_i\|$ from $\mathbf{pv}_j$ (the pose in L nearest to $\mathbf{x}_o$) to $\mathbf{pv}_g$ (the goal pose in L), where $\|\mathbf{pv}_k - \mathbf{x}_i\|$ is distance between the pose of each element $\mathbf{pv}_k$ of L to the pose of $\mathbf{x}_i \in T$ nearest to $\mathbf{pv}_k$, and is calculated by the equation (Figure 18):

$$D^L = \sum_{k=j}^{g} \|\mathbf{pv}_k - \mathbf{x}_i\| \tag{18}$$

To solve the minimization problem, we start with an initial guess for $\mathbf{tcp}$, $\mathbf{tcp}^{seed}$, taken from a 5-dimension pre-computed table, named *trajectory look-up table*, TLT, which is indexed by indexes computed from $\mathbf{x}_o$ and $\mathbf{pv}_g$, named *discrete trajectory descriptors*,

$\mathbf{dtd} = (\lambda_g^d, \phi_g^d, \theta_g^d, v_o^d, \varphi_o^d)$, as described in Section 4.5 below. The indexes λ_g^d, ϕ_g^d, θ_g^d, v_o^d and φ_o^d are computed from λ_g, ϕ_g, θ_g, v_o and φ_o, respectively, which, as a tuple, are called *trajectory descriptors*, or $\mathbf{td} = (\lambda_g, \phi_g, \theta_g, v_o, \varphi_o)$ (Section 4.5). We then use the conjugate gradient optimization algorithm to minimize $f(.)$ by manipulating the $\mathbf{tcp}^{seed}$ elements (the trajectory total time, tt, and three knot points, k_1, k_2 and k_3, of the cubic spline that specifies the evolution of the steering angle during tt).

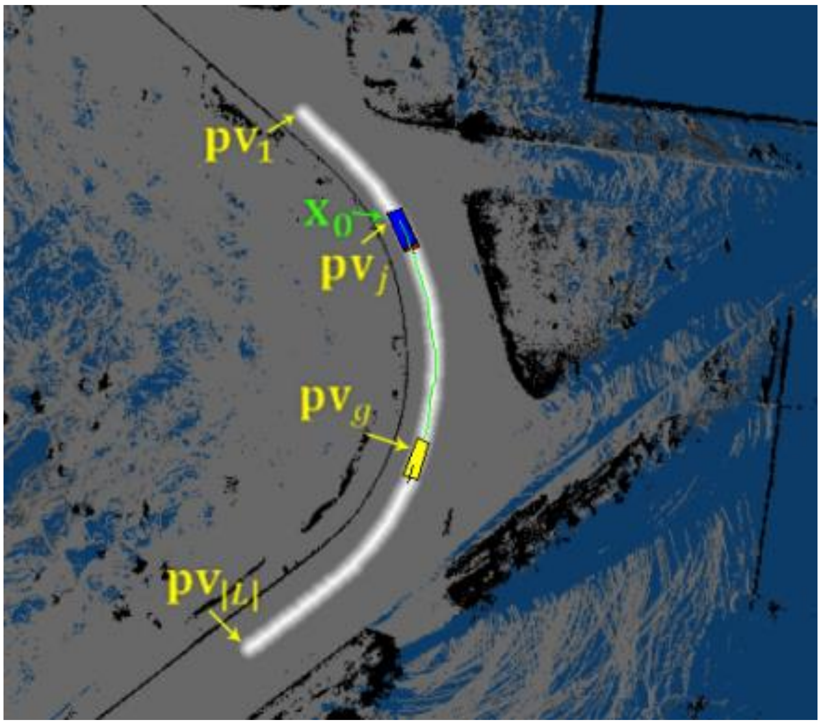


Figure 18: Variables associated with D^L, which summarizes the distances between each point in L and its nearest point in T.

The conjugate gradient algorithm requires derivatives of the cost function, $f(.)$, with respect to the optimizing variables, tt, k_1, k_2 and k_3. In our model-predictive planner, we compute these derivatives numerically using finite differences. Unfortunately, the minimization problem includes parameters (e.g. the weights w_1 to w_5) that need to be tuned empirically. Fortunately, we have a IARA`s simulator that reproduces accurately its behavior in speeds up to 9 m/s, because it is based on a neural network [DES16]. Using this simulator in specific scenarios in our simulation environment, we can tune these parameters, so that IARA can perform properly in autonomous mode.

4.4 IARA'S Model-Predictive Planner Algorithm

The listing of Algorithm 1 presents our model-predictive motion planning algorithm. At a rate of 20 Hz, compute_motion_plan() receives $\mathbf{x}_o$, $\mathbf{pv}_g$, L and map from IARA's modules as input, and returns T as output.

Algorithm 1: compute_motion_plan

```
Input: $\mathbf{x}_o, \mathbf{pv}_g, L, map$
Output: $T$
1:  $\mathbf{td} \leftarrow$ get_td($\mathbf{x}_o, \mathbf{pv}_g$)
2:  $\mathbf{dtd} \leftarrow$ get_dtd($\mathbf{td}$)
3:  $\mathbf{tcp}^{seed} \leftarrow TLT\left[\mathbf{dtd}^{(\lambda_g^d)}\right]\left[\mathbf{dtd}^{(\phi_g^d)}\right]\left[\mathbf{dtd}^{(\theta_g^d)}\right]\left[\mathbf{dtd}^{(\varphi_o^d)}\right]\left[\mathbf{dtd}^{(v_o^d)}\right]$
4:  $\mathbf{tcp}^{optimized} \leftarrow \arg\min_{\mathbf{tcp}} f(M(\mathbf{x}_o, \mathbf{pv}_g, \mathbf{tcp}^{seed}), L, map)$
5:  if valid($\mathbf{tcp}^{optimized}$) then
6:      $a \leftarrow (\mathbf{pv}_g{}^{(v_g)} - \mathbf{x}_o{}^{(v_o)})/\mathbf{tcp}^{optimized^{(tt)}}$
7:      $T \leftarrow$ get_$T$_by_simulation($\mathbf{x}_o, a, \mathbf{tcp}^{optimized}$)
8:  else
9:      $T \leftarrow \emptyset$
10: return ($T$)
```

In line 1, the Algorithm 1 uses the function get_td() to compute $\mathbf{td}$ as a function of $\mathbf{x}_o$ and $\mathbf{pv}_g$ (Section 4.5). In line 2, it uses the function get_dtd() to compute $\mathbf{dtd}$ from $\mathbf{td}$ (Section 4.5). In line 3, it gets $\mathbf{tcp}^{seed}$ from TLT and, in line 4, it runs the conjugate gradient algorithm to find a $\mathbf{tcp}^{optimized}$ that takes the car from $\mathbf{x}_o$ to $\mathbf{pv}_g$. In line 6, the acceleration a is computed. If the optimization succeeds, in line 7, the function get_T_by_simulation() (trivial implementation; listing not shown) simulates the car, according to equations (6) to (11), starting from $\mathbf{x}_o$ (time $t = 0$) and until time tt of $\mathbf{tcp}^{optimized}$ using a. At the end of this simulation, the function get_T_by_simulation() returns a trajectory T that ends as close as possible to $\mathbf{pv}_g$ and with poses: (i) as close as possible to L and (ii) as far as possible from the obstacles in the map. If the optimization fails, in line 9 an empty trajectory is returned in the cycle. If compute_motion_plan() fails to find trajectories for many consecutive cycles, IARA is stopped. Note that IARA does not move much in a single cycle and a previous trajectory might still be suitable after several cycles. IARA's obstacle avoider module takes care of it in such cycles (or in any situation that may lead to a collision) [GUI16].

4.5 Trajectory Look-up Table – *TLT*

Initial guesses for **tcp**, $\mathbf{tcp}^{seed}$, are pre-computed with varying $\mathbf{td} = (\lambda_g, \phi_g, \theta_g, v_o, \varphi_o)$, where λ_g and ϕ_g are the relative differences between $\mathbf{x}_o$ and $\mathbf{pv}_g$ in polar coordinates; θ_g is the goal orientation; v_o is the initial velocity; and φ_o is the initial steering angle (Figure 19). A $\mathbf{tcp}^{seed}$ computed from $\mathbf{td} = (\lambda_g, \phi_g, \theta_g, v_o, \varphi_o)$ is stored in the TLT cell indexed by $\mathbf{dtd} = (\lambda_g^d, \phi_g^d, \theta_g^d, v_o^d, \varphi_o^d)$, i.e., $TLT[\lambda_g^d][\phi_g^d][\theta_g^d][v_o^d][\varphi_o^d] = \mathbf{tcp}^{seed}$. The elements of **td** are derived from $\mathbf{x}_o$ and $\mathbf{pv}_g$ according to the following equations:

$$\lambda_g = \|(x_g - x_o, y_g - y_o)\|, \tag{19}$$

$$\phi_g = atan2((y_g - y_o)/(x_g - x_o)), \tag{20}$$

$$\theta_g = \mathbf{pv}_g{}^{(\theta)}, \tag{21}$$

$$v_o = \mathbf{x}_o{}^{(v)} \ and \tag{22}$$

$$\varphi_o = \mathbf{x}_o{}^{(\varphi)}, \tag{23}$$

where, in the equations (21), (22) and (23) above, $\mathbf{tuple}^{(a)}$ is the element a of a $\mathbf{tuple} = (a, b, c, \dots)$.

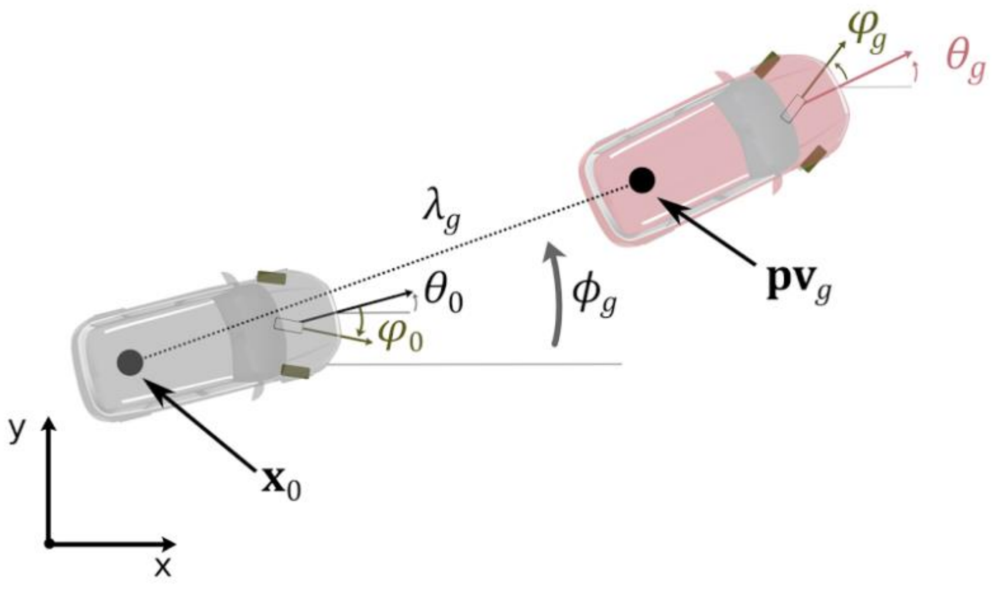


Figure 19: Elements of a **td** (λ_g, ϕ_g, θ_g, v_o and φ_o), which are derived from $\mathbf{x}_o$ and $\mathbf{pv}_g$

The discrete indexes, λ_g^d, ϕ_g^d, θ_g^d, v_o^d and φ_o^d, are defined as:

$$\lambda_g^d = g(\lambda_g, cr^{\lambda_g}, sf^{\lambda_g}, zi^{\lambda_g}), \tag{24}$$

$$\phi_g^d = h(\phi_g, cd^{\phi_g}, zi^{\phi_g}), \tag{25}$$

$$\theta_g^d = g(\theta_g, cr^{\theta_g}, sf^{\theta_g}, zi^{\theta_g}), \tag{26}$$

$$v_o^d = g(v_o, cr^{v_o}, sf^{v_o}, zi^{v_o}) \text{ and} \tag{27}$$

$$\varphi_o^d = g(\varphi_o, cr^{\varphi_o}, sf^{\varphi_o}, zi^{\varphi_o}), \tag{28}$$

where

$$g(\iota, cr, sf, zi) = \left[log_{cr}\frac{\iota + sf}{sf}\right] + zi \tag{29}$$

for $\iota = \lambda_g$ (or θ_g, v_o and φ_o), $cr = cr^{\lambda_g}$ (or cr^{θ_g}, cr^{v_o} and cr^{φ_o}), $sf = sf^{\lambda_g}$ (or sf^{θ_g}, sf^{v_o} and sf^{φ_o}) and $zi = zi^{\lambda_g}$ (or zi^{θ_g}, zi^{v_o} and zi^{φ_o});

$$h(\iota, cd, zi) = \left[\frac{\iota}{cd}\right] + zi \tag{30}$$

for $\iota = \phi_g$, $cd = cd^{\phi_g}$ and $zi = zi^{\phi_g}$; and, in the equations (29) and (30), $[e]$ is equal to the nearest integer to e.

The functions $g(.)$ and $h(.)$, and their parameters for each element of $\mathbf{dtd}$, were chosen in order to compute a $\mathbf{dtd}$ suitable for accessing a TLT that would contain the appropriate $\mathbf{tcp}^{seed}$ for building trajectories with properties in the range of operation of IARA. More specifically, the functions $g(.)$ and $h(.)$ were chosen in order to apply finer-grained discretization of the elements of $\mathbf{dtd}$ for smaller relative differences between $\mathbf{x}_o$ and $\mathbf{pv}_g$ and coarser-grained discretization for larger ones. Table I presents the values we have chosen for the parameters of $g(.)$ and $h(.)$ of each element of $\mathbf{dtd}$, and the bounds, n, of $g(.)$ and $h(.)$ for each element. We also have functions $g^{-1}(.)$ and $h^{-1}(.)$ that compute the inverse of $g(.)$ and $h(.)$, i.e., map each member of $\mathbf{dtd}$ to an element of $\mathbf{td}$ (trivial deduction; not shown).

Table I: $\boldsymbol{g}(.)$ AND $\boldsymbol{h}(.)$ PARAMETERS AND BOUNDS

dtd	**$g(.)$ and $h(.)$ parameters**				**bounds (n)**	
	cr	sf	zi	cd	**min**	**Max**
λ_g^d	1.8	2.3	-1	-	0	15
ϕ_g^d	-	-	7	0.139	0	15
θ_g^d	1.3	0.174	7	-	0	15
φ_o^d	1.394	0.052	7	-	0	15
v_o^d	1.381	1.3	0	-	0	8

The listing of Algorithm 2 presents the algorithm designed to generate the TLT. In line 1, Algorithm 2 initializes TLT, L and map as empty (when building a $\mathbf{tcp}^{seed}$ one does not need to consider a lane or map). In lines 2, 3 and 4, Algorithm 2 cycles through all possible values of λ_g^d, v_o^d and φ_o^d (i.e. within the bounds of Table I), while, in lines 5 and 6, it samples k_2 and k_3 throughout the range of possible IARA's steering angle values (at small intervals). The value min^{φ} is the minimum steering angle and max^{φ} is the maximum steering angle. Throughout lines 7 to 14, Algorithm 2 use all these values to compute a $\mathbf{tcp}^{\text{seed}}$, which is saved into TLT in line 16.

Algorithm 2: fill_trajectory_lookup_table

```
Input: void
Output: TLT
1:  $TLT \leftarrow \emptyset, L \leftarrow \emptyset, map \leftarrow \emptyset$
2:  for $\lambda_g^d = 0$ to $n^{\lambda_g} - 1$
3:    for $v_o^d = 0$ to $n^{v_o} - 1$
4:      for $\varphi_o^d = 0$ to $n^{\varphi_o} - 1$
5:        for $k_2 = min^{\varphi}$ to $max^{\varphi}$
6:          for $k_3 = min^{\varphi}$ to $max^{\varphi}$
7:            $\mathbf{tcp}^{sample}, a \leftarrow$ get_tcp_sample_and_acc($\lambda_g^d, v_o^d, k_2, k_3$)
8:            $\mathbf{x}_o = (0,0,0, g^{-1}(v_o^d), g^{-1}(\varphi_o^d), 0)$
9:            $\mathbf{td}, v_g \leftarrow$ get_td_and_vg_by_simulation($\mathbf{x}_0, a, \mathbf{tcp}^{sample}$)
10:           $\mathbf{dtd} \leftarrow$ get_dtd($\mathbf{td}$)
11:           if valid($\mathbf{dtd}$) then
12:             $\lambda = g^{-1}(\mathbf{dtd}^{(\lambda_g^d)}), \phi = h^{-1}(\mathbf{dtd}^{(\phi_g^d)}), \theta = g^{-1}(\mathbf{dtd}^{(\theta_g^d)})$
13:             $\mathbf{pv}_g = (\lambda \cos\phi, \lambda \sin\phi, \theta, v_g)$
14:             $\mathbf{tcp}^{seed} \leftarrow \arg\min_{\mathbf{tcp}} f(M(\mathbf{x}_o, \mathbf{pv}_g, \mathbf{tcp}^{sample}), L, map)$
15:             if valid($\mathbf{tcp}^{seed}$) then
16:               $TLT[\mathbf{dtd}^{(\lambda_g^d)}][\mathbf{dtd}^{(\phi_g^d)}][\mathbf{dtd}^{(\theta_g^d)}][\mathbf{dtd}^{(\varphi_o^d)}][\mathbf{dtd}^{(v_o^d)}] \leftarrow \mathbf{tcp}^{seed}$
17: return ($TLT$)
```

The function get_tcp_sample_and_acc(), called in line 7 of Algorithm 2, is presented in Algorithm 3. This function computes a $\mathbf{tcp}^{\mathrm{sample}}$ and an acceleration, a. The elements k_2 and k_3 of $\mathbf{tcp}^{\mathrm{sample}}$ come directly from the inputs of get_tcp_sample_and_acc(), while tt of $\mathbf{tcp}^{\mathrm{sample}}$ is computed in lines 3 to 9 of Algorithm 3. Note that k_1 is not filled in $\mathbf{tcp}^{\mathrm{seed}}$, since this part of a **tcp** is only computed during run time – k_1 is used for adding maneuverability to avoid obstacles and to keep IARA precisely in the road during the execution of compute_motion_plan(), Algorithm 1. Finally, the acceleration a is computed in line 10 of Algorithm 3 using the equation of motion, $\lambda = v_o \times t + a \times t^2/2$, solved for the acceleration ($a = (\lambda - v_o \times t)/(t^2/2)$).

Algorithm 3: get_tcp_sample_and_acc

```
Input: $\lambda_g^d, v_o^d, k_2, k_3$
Output: $\mathbf{tcp}^{sample}, a$
1:  $\mathbf{tcp}^{sample(k_2)} = k_2$
2:  $\mathbf{tcp}^{sample(k_3)} = k_3$
3:  $\lambda = g^{-1}(\lambda_g^d)$
4:  if $\lambda > 7$ then
5:    $\mathbf{tcp}^{sample(tt)} = 5s$
6:  else if $\lambda > 3.5$ then
7:    $\mathbf{tcp}^{sample(tt)} = 2.5s$
8:  else
9:    $\mathbf{tcp}^{sample(tt)} = 2s$
10: $a = (\lambda - g^{-1}(v_o^d) \times tt)/(tt^2/2)$
11: return ($\mathbf{tcp}^{sample}, a$)
```

Algorithm 2 initializes the car at the origin in line 8, but with velocity $g^{-1}(v_o^d)$ and steering angle $g^{-1}(\varphi_o^d)$. In line 9, using the function get_td_and_vg_by_simulation() (trivial implementation; listing not shown), it simulates the car, according to equations (6) to (11), from $\mathbf{x}_o$ (time $t = 0$) until time tt of $\mathbf{tcp}^{sample}$ using acceleration a. At the end of this simulation, get_td_and_vg_by_simulation() computes a **td** using the last state of the simulation as goal. The function get_td_and_vg_by_simulation() then returns this **td** as well as the velocity of the last state of the simulation as v_g.

In line 10, Algorithm 2 uses the function get_dtd() (trivial implementation; listing not shown) to compute a **dtd** from a **td**, according to equations (24) to (28). This **dtd** might be invalid (one of its elements might be out of the bounds shown in Table I) and, in such cases, it is discarded. If this $\mathbf{dtd}$ is valid, in lines 12 and 13, Algorithm 2 builds a $\mathbf{pv}_g$ and, in line 14, runs the conjugate gradient algorithm to find a $\mathbf{tcp}$ that takes the car from $\mathbf{x}_o$(time t = 0) to $\mathbf{pv}_g$ (time $t = tt$). If the optimization succeeds, in line 16 the computed $\mathbf{tcp}^{seed}$ is saved in TLT.

Many combinations of $\lambda_g^d, \phi_g^d, \theta_g^d, \varphi_o^d, v_o^d$ can never occur in real life due to the physical restrictions of IARA. For example, it cannot change from an angle θ_o to a largely different angle θ_g in a short λ_g. Therefore, many TLT cells are empty after Algorithm 2. To fill as much cells as possible, after Algorithm 2, we run a function that goes back to each empty cell of TLT and use nearby occupied cells as seeds for computing the $\mathbf{tcp}$ of these empty cells (listing not shown). This process is repeated until the function cannot fill extra cells with a valid $\mathbf{tcp}$. Using this process, we were able to fill 57.63% of TLT. The experimental evaluation has shown that this is enough for proper real time operation.

5 EXPERIMENTAL METHODOLOGY

In this chapter, we present the experimental methodology. In Section 5.1, we describe the IARA's Hardware sensors and actuators. In Section 5.2, we describe the framework and modules whose compose the IARA's Software tools. In Section 5.3, we describe the environment where we run the experiments to evaluate the MPMP. In Section 5.4, we describe the methodology of the experiments. In Section 5.5, we describe the metrics applied to evaluate the MPMP performance.

5.1 IARA's Hardware

We developed the hardware and software of IARA (Figure 1). IARA's hardware is based on a Ford Escape Hybrid, which was adapted by Torc Robotics [TOR17] to enable electronic actuation of the steering wheel, throttle, and brake; reading the car odometry; and powering several high-performance sensors and computers. The IARA's hardware includes one Velodyne HDL 32-E Light Detection and Ranging (LiDAR); one SICK LD-MRS 3D LiDAR; one Trimble Dual Antenna RTK GPS; one Xsens Mti IMU; four Point Grey Bumblebee stereo cameras, one ZED 2K stereo camera (Figure 21); one Switch Gigabit CISCO Catalyst; and a Dell Precision R5500 computer (Figure 20 and Figure 22).

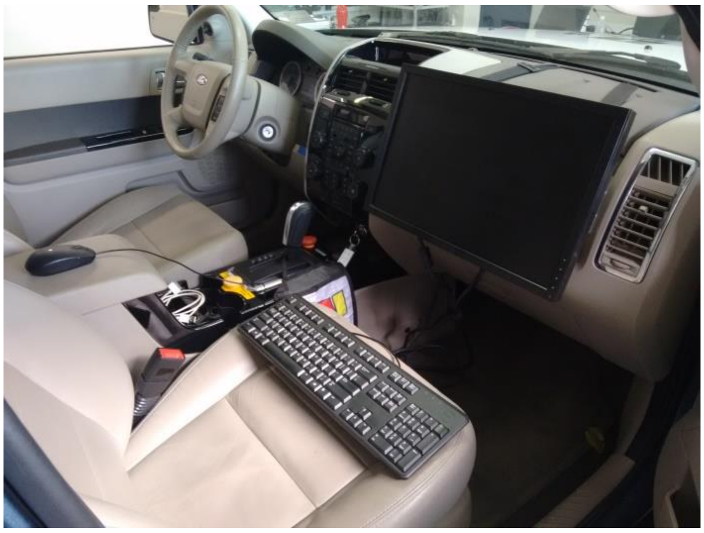


Figure 20: Interface to operate IARA's system.

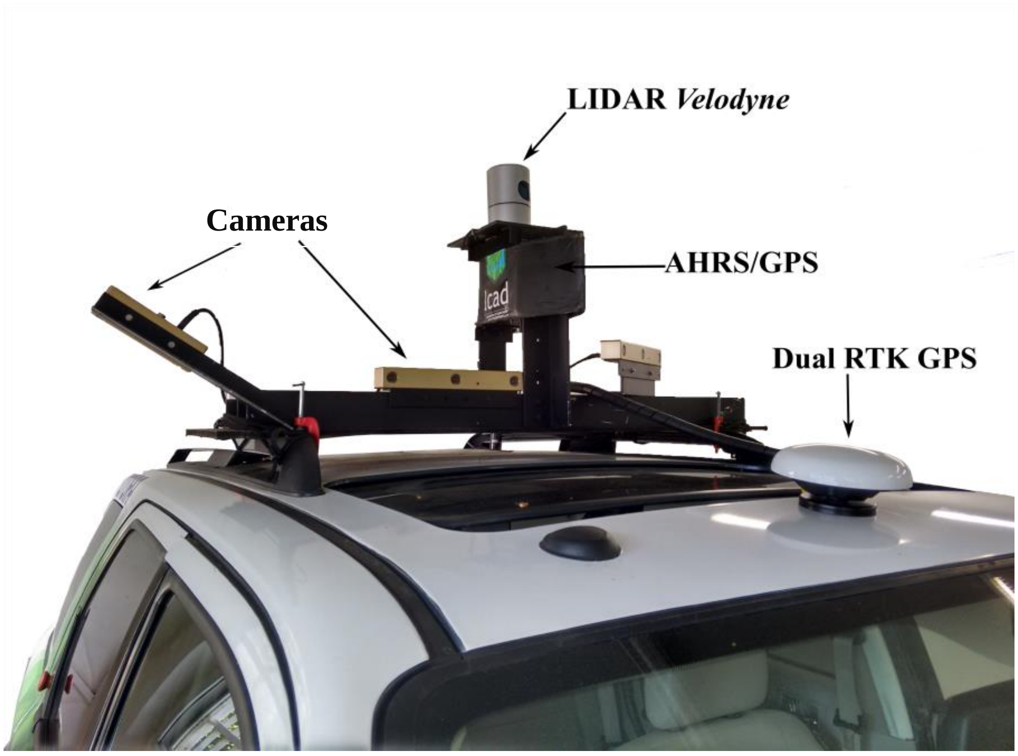


Figure 21: IARA's sensors

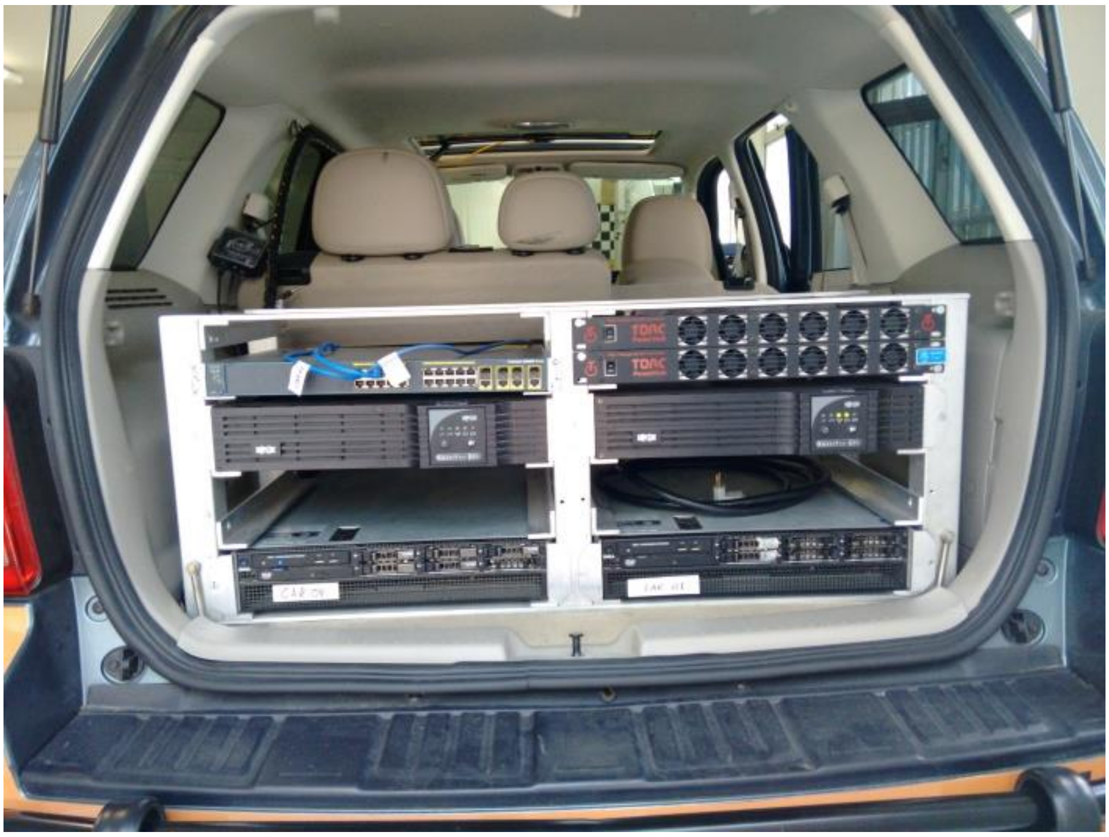


Figure 22: Computers resources.

5.2 IARA’s Software

The IARA’s software is composed of seven main modules: localizer [VER15] [VER16], mapper [MUT16], behavior selector, path planner, motion planner [RAD14] [CAR17] (the focus of this work), obstacle avoider [GUI16] and controller [GUI17]. The localizer module is responsible for estimating the state of the robot relative to the origin of the map using the LiDAR data and the offline map. The mapper module is responsible for creating an online map using the LiDAR, the occupancy grid map algorithm, and the IARA localization. The behavior selector module is responsible for defining the local goal states and velocities that IARA’s has to follow in order to reach its goal; this module considers the information about the environment, namely the map, traffic lights, traffic rules and the global plan. The path planner module is responsible for building a global path from IARA’s initial state to the desired end-pose. The motion planner module, presented in this work, is responsible for computing the local motion planning, which drives IARA’s from the current state to the next goal selected by the behavior selector module. The obstacle avoider module is responsible for verifying the current trajectory generated by the motion planner and reducing the velocities or stopping the car in case of possible collisions; this module is a safeguard that works two times faster than the motion planner. Finally, the controller module is responsible for converting the control commands of the trajectories into acceleration, brake and steering efforts [GUI17]. Together, these modules allow IARA’s autonomous operation on urban roads.

Additional modules are: health monitor, logger, traffic lights state detector, simulator [DES16], among others. These modules were implemented in Carnegie Mellon Robot Navigation (CARMEN) Toolkit. This toolkit was extended by the LCAD for the IARA’s project.

Figure 23 shows a block diagram of all the IARA’s software modules, including the main ones cited above.

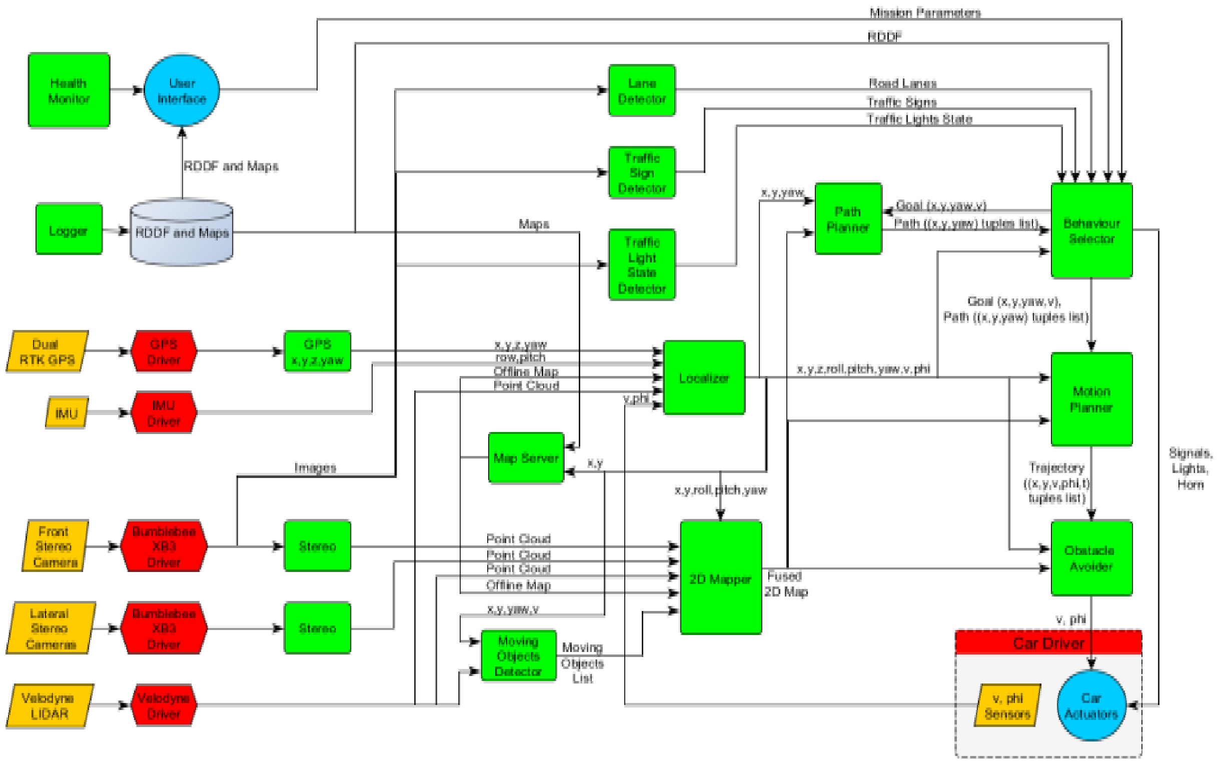


Figure 23: Block diagram of all IARA's software modules.

5.2.1 CARMEN Toolkit

There are some frameworks for robots software development. Regarding open source ones, we can cite CARMEN (Carnegie Mellon Robot Navigation Toolkit) (http://carmen.sourceforge.net/) and ROS (Robot Operating System) (http://www.ros.org/).

CARMEN is a modular collection of software for mobile robot control. CARMEN supports several robot hardware platforms, sensors and provides basic navigation primitives. This Toolkit aims to standardize the mobile robot programming as well as avoid rework, once it is open source, and its programs (modules) can be shared and used in different robot platforms [MON03].

Each module handles a subtask in a robot system and communicates with each other. To communicate between modules, CARMEN uses the Inter-Process Communication protocol (IPC), wherewith each module can publish messages and/or subscribe to receive published messages. The sending of a new message happens through a publication sent by a Publisher to the Central server. Message reception is asynchronous, that is, the Subscribers provide a callback function associated with each signed message. Then the Central server sends a copy of the message to all Subscribers and every time a new message is received the respective callback function is executed. The Central server is also responsible for storing the systems' information, such as the message's name, network routes and network traffic log messages [SIM17]. IPC supports messages containing complex data structures. Figure 24 shows an illustration of the inter-process model.

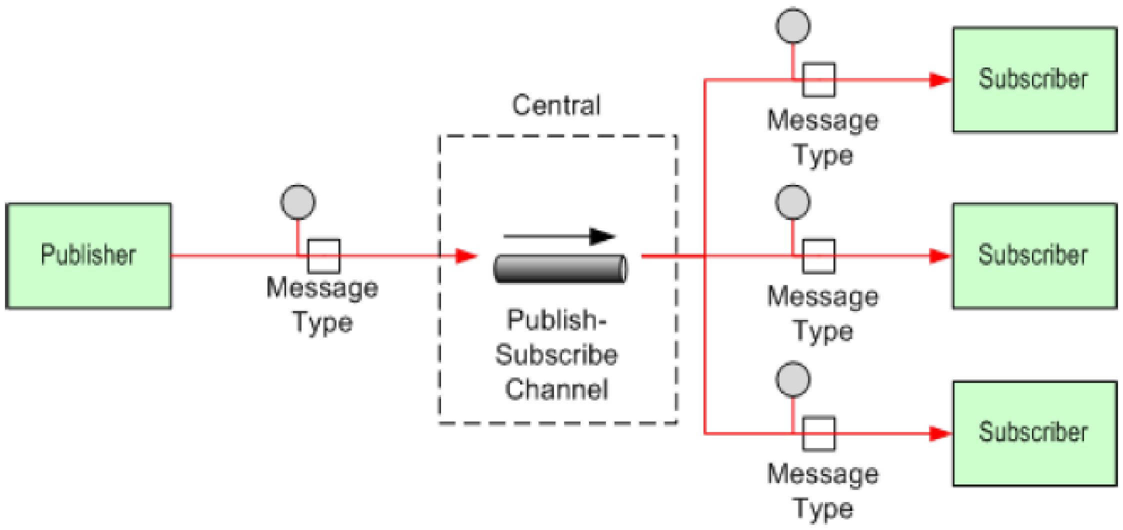


Figure 24: Illustration of the Publish-Subscribe inter-process model used in CARMEN with IPC [HOH03].

ROS is a collection of tools, libraries, and conventions for creating robots, which supports a variety of robotic platforms. ROS is a multi-lingual framework that supports various programming languages, such as C++, Python, Octave and LISP. In this way, programs (nodes) implemented in different languages can communicate with each other through a message layer. ROS also uses a publish/subscribe scheme for inter-process communication as CARMEN, but with peer-to-peer communication, that is, the messages data do not route through a Central server; instead, a *name server* stores the routes and the nodes communicate directly with each other. The transmission around a network does not use a single transport protocol; so, nodes negotiate a connection with its appropriated protocol [ROS17] [QUI09].

Even though ROS is also a robust framework, when the IARA's project began, ROS was under a lot of change. Therefore, it was unstable at the time for the project's purpose. CARMEN was simpler and, at the time, seemed to be enough as a platform for the development of the LCAD's autonomous car, what has proven to be true with the successful development of IARA. Currently, the LCAD has a trained team of developers using CARMEN. LCAD CARMEN version advanced from the initial version and outcome initial limitations, allowing now the development of anything developed currently on ROS for non-articulated robots. In this way, the modules described in Section 5.2, including the MPMP, were developed using the CARMEN toolkit.

5.3 Test Environment

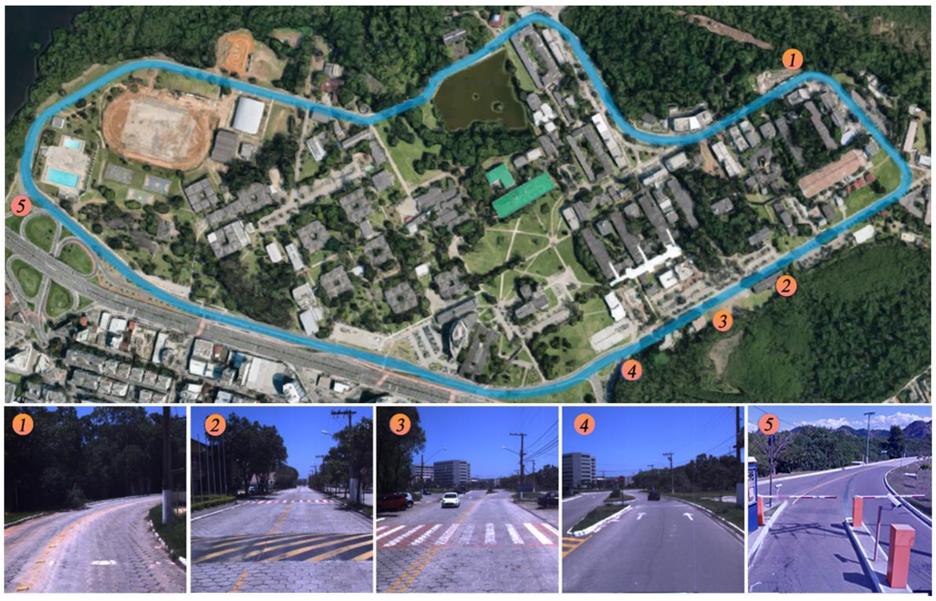


Figure 25: Pictures of the university testing environment. The images 1 to 5 show some of the relevant parts of the course.

IARA's test environment is the *Universidade Federal do Espírito Santo* (www.ufes.br) main campus beltway (UFES beltway). Figure 25 shows parts of the UFES beltway. It is a very challenging course with 3.7 km since it is frequently busy, with many cars, motorcycles, and buses traveling around, parked cars on both sides of the road, as well as pedestrians crossing (Figure 25(2) and (3)). The road pavement includes segments with cobbles (Figure

25(1)) and asphalt (Figure 25(4)). Furthermore, it has six speed bumps, sharp and wide curves, varying track widths and two gate barriers. Therefore, the motion planner must be able to deal with all these hazards in order to drive the car smoothly and safely.

5.4 Test Methodology

We asked an experienced Human driver to drive IARA along the UFES beltway and logged the car poses (given by our localizer module). We transformed the sequence of poses and velocities observed during Human driving into a standard IARA path (as would be produced by our path planner). Later, we commanded IARA to perform the same path in autonomous mode using the MPMP. We evaluated the differences between the Human path and the MPMP path. Furthermore, we show the behavior of MPMP with obstacles in real world situations. The results of these evaluations are reported in Chapter 6.

5.5 Metrics

We evaluated the performance of MPMP using the mean and standard deviation of the distance between each point of the path followed by MPMP to the path followed by the Human driver (Equations (31) and (32)). Beyond that, we compared the velocities of MPMP and Human driver along the path followed.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(d_i - \overline{\mathrm{X}})^2}{n}} \tag{31}$$

To evaluate the capacity of the planner to deal with obstacles in the road, we presented situations in which obstacles blocked part of the road. We then evaluated visually the ability of the planner to compute trajectories that follow the lane while maintaining a safe distance from obstacles.

6 EXPERIMENTAL RESULTS

The performance of the MPMP driving IARA in a real world environment was compared with that of a Human driver. Figure 26 shows the distance between each point of the path followed by MPMP to the path followed by the Human driver. In the graph of this figure, the *x*-axis represents the time of each pose of the Human path, while the *y*-axis represents the distance between paths. Note that we have synchronized the poses of the MPMP path with the poses of the Human path so as to measure the distance between the correct poses, since the velocities were not the same in both paths. For that, for each point of MPMP path, we found a point in the line that connects the nearest two points in the Human path. As Figure 26 shows, the absolute distance between paths is small throughout the whole paths, never exceeding 0.8 m. The average distance was 0.15 m ($\sigma = 0.14$).

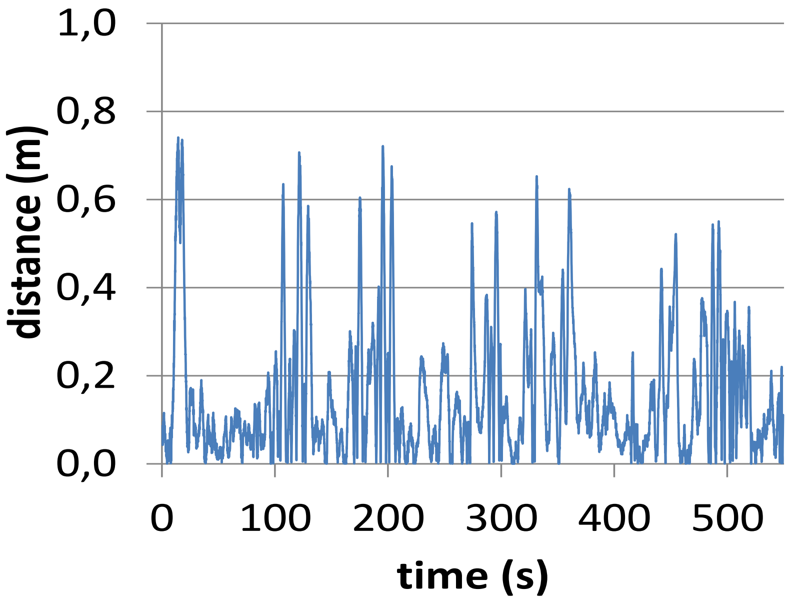


Figure 26: Absolute distance between MPMP poses and Human poses.

Figure 27 compares the velocities of MPMP and Human along the UFES beltway. In the graph of this figure, in the *x*-axis we show the time of each velocity sample, while in the *y*-axis we show the velocities (the velocity samples were synchronized with the time of the poses of Figure 26). The two curves in the graph show the MPMP velocity in red and the Human velocity in blue.

The maximum allowed speed in the UFES beltway varies, but, in most of the beltway, it is 30 km/h (8.33 m/s). We programmed MPMP to maintain 30 km/h (8.33 m/s) and asked the Human driver to follow the beltway speed limits. As the graph in Figure 27 shows, MPMP managed to maintain IARA's speed close to the programmed velocity. Its speed was reduced to less than 7.2 km/h (2 m/s) at some points due to the bumps and gate barriers present in the beltway. As can be seen in Figure 27, MPMP was more cautious while crossing bumps, gates barriers and other places that required slower speeds. Also, its speed was more stable than the Human speed throughout the course, which is known to reduce fuel consumption.

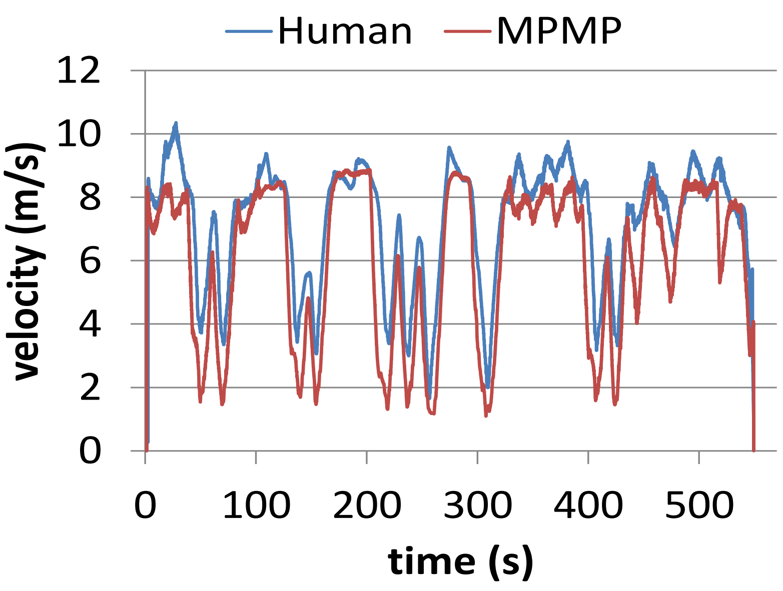


Figure 27: MPMP velocities (red) and Human velocities (blue).

Figure 28 compares the poses of Human (blue) and MPMP (red) throughout the UFES beltway. As Figure 28 shows, in the scale of the whole beltway, the poses of the two drivers are indistinguishable. The inset in Figure 28 highlights one of the regions of the beltway where the distance between the paths is around the largest. As this inset shows, both paths are smooth and the distance between them changes smoothly as well. The region is showed in Figure 29.

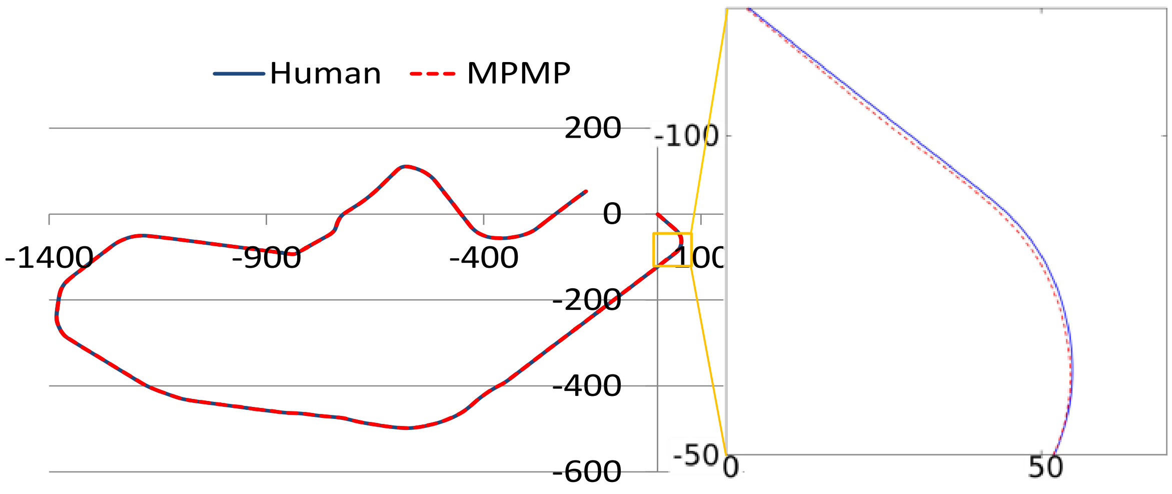


Figure 28: Poses of the two paths throughout the UFES beltway.

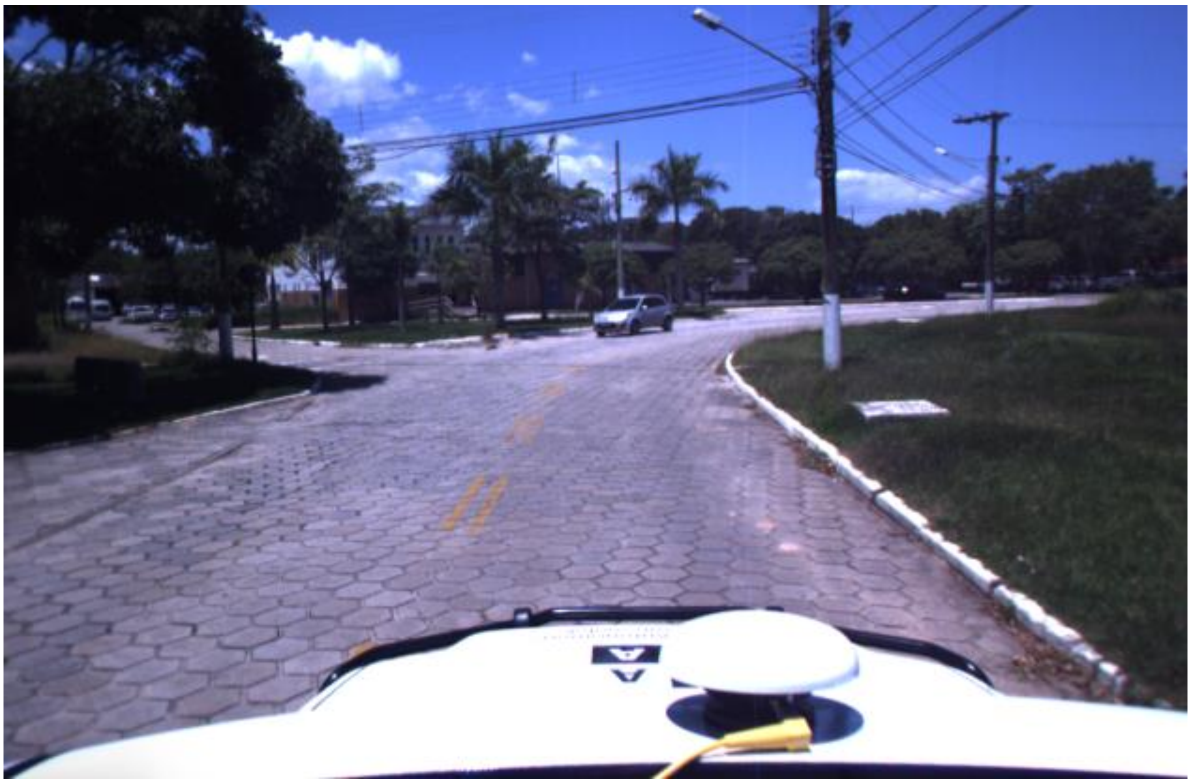


Figure 29: Region highlighted in the inset of the Figure 28.

Figure 30 shows the heading (orientation θ) of the Human path and the MPMP path in the UFES beltway of the same experiment shown in Figure 28 above. It is possible to see that the MPMP maintain its heading very close to the human heading in virtually the whole beltway.

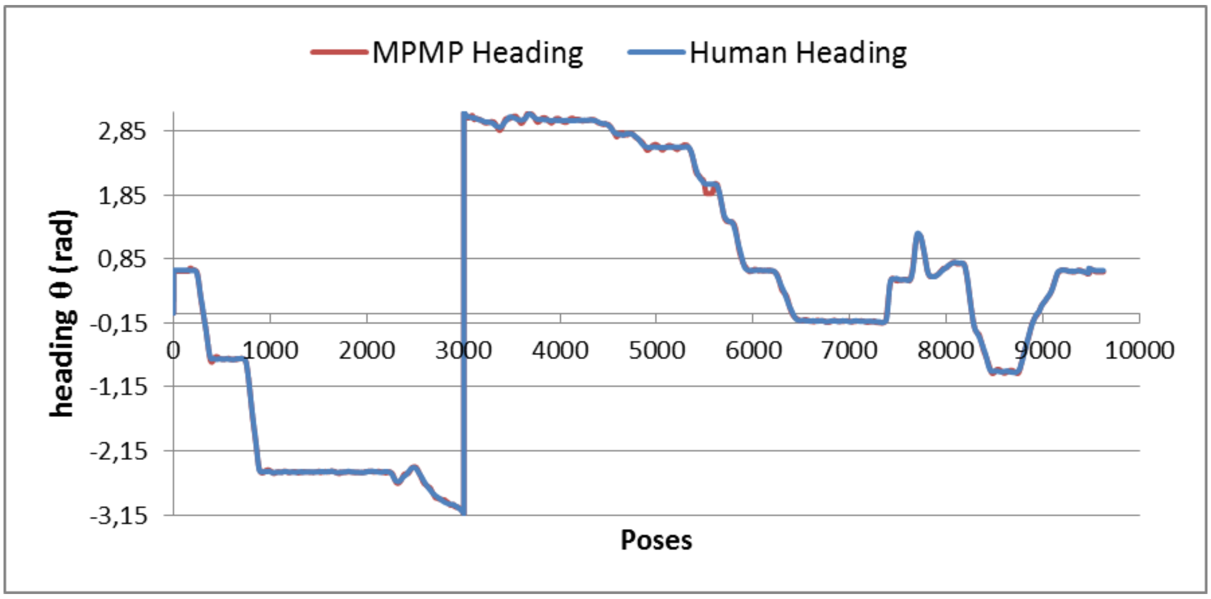
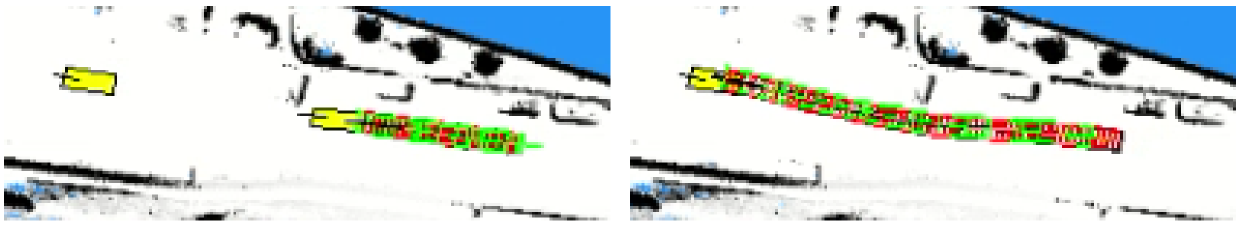


Figure 30: Headings (orientation θ) of the two paths throughout the UFES beltway.

In real traffic situations, the planner has also to deal with obstacles in the road. Figure 31 shows one such situation that happened in the UFES beltway. Several buses were parked on the side the road, blocking part of it. The MPMP had to optimize the trajectory to maintain a safe distance from obstacles while maintaining the trajectory as close as possible to the lane (Equation (13)). Please note that the Brazilian traffic regulations allow crossing school busses (or other cars) in the situation shown in Figure 31. The weights of the cost function are the same in all experiments.

Figure 31 presents the situation according to the MPMP perspective. The optimized trajectory (green/red) avoids the obstacles between the IARA (rectangle in the beginning of the trajectory, on the right of it) and the goals (yellow rectangle on the left). The MPMP tries to maintain the IARA on the lane while keeping a safe distance from the obstacles. This experiment can be seen in the video https://youtu.be/o_NU23fpZhw?t=134.



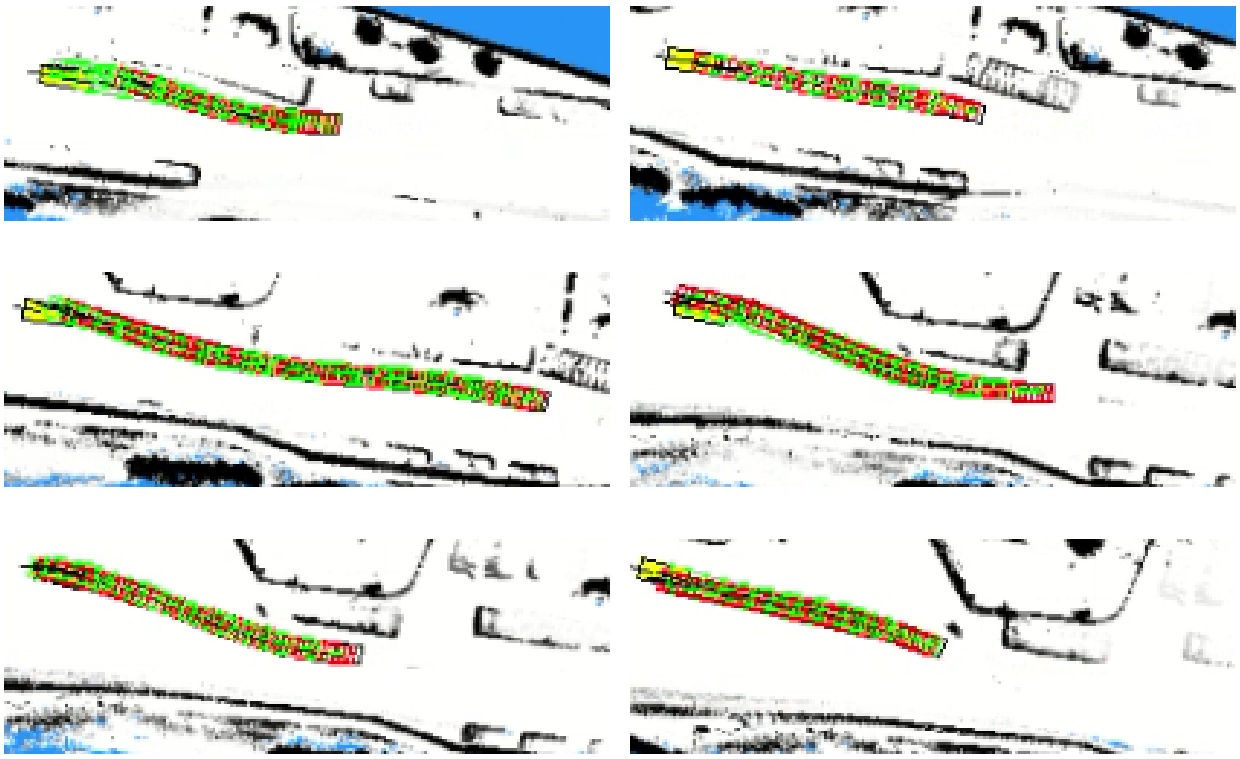
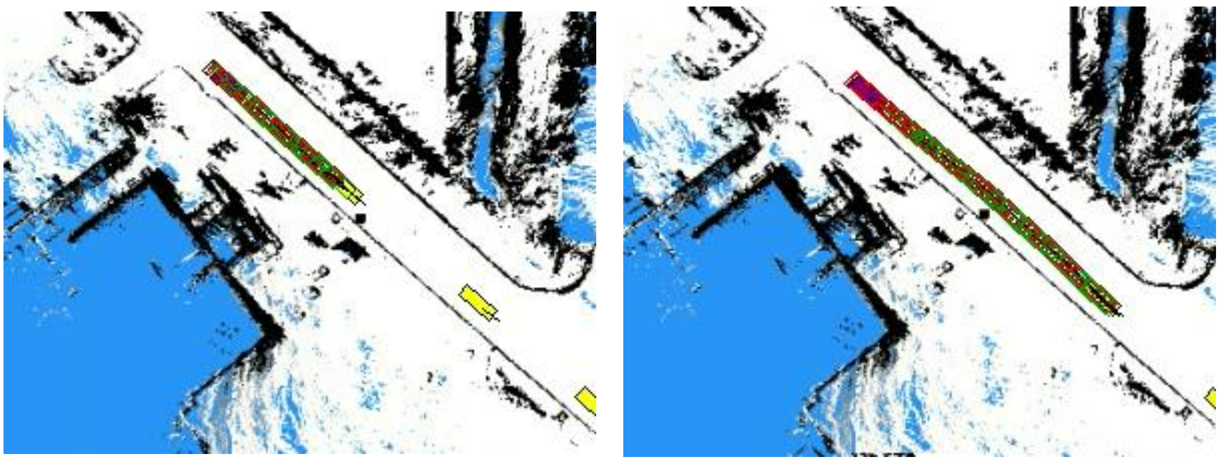


Figure 31: MPMP trajectories (in green/red) avoiding obstacles blocking the road.

Figure 32 shows another situation of obstacle avoidance. When an obstacle appears on the side of the road, the MPMP optimizes a trajectory that maintains a safe distance from the obstacle and goes back to the lane in sequence.



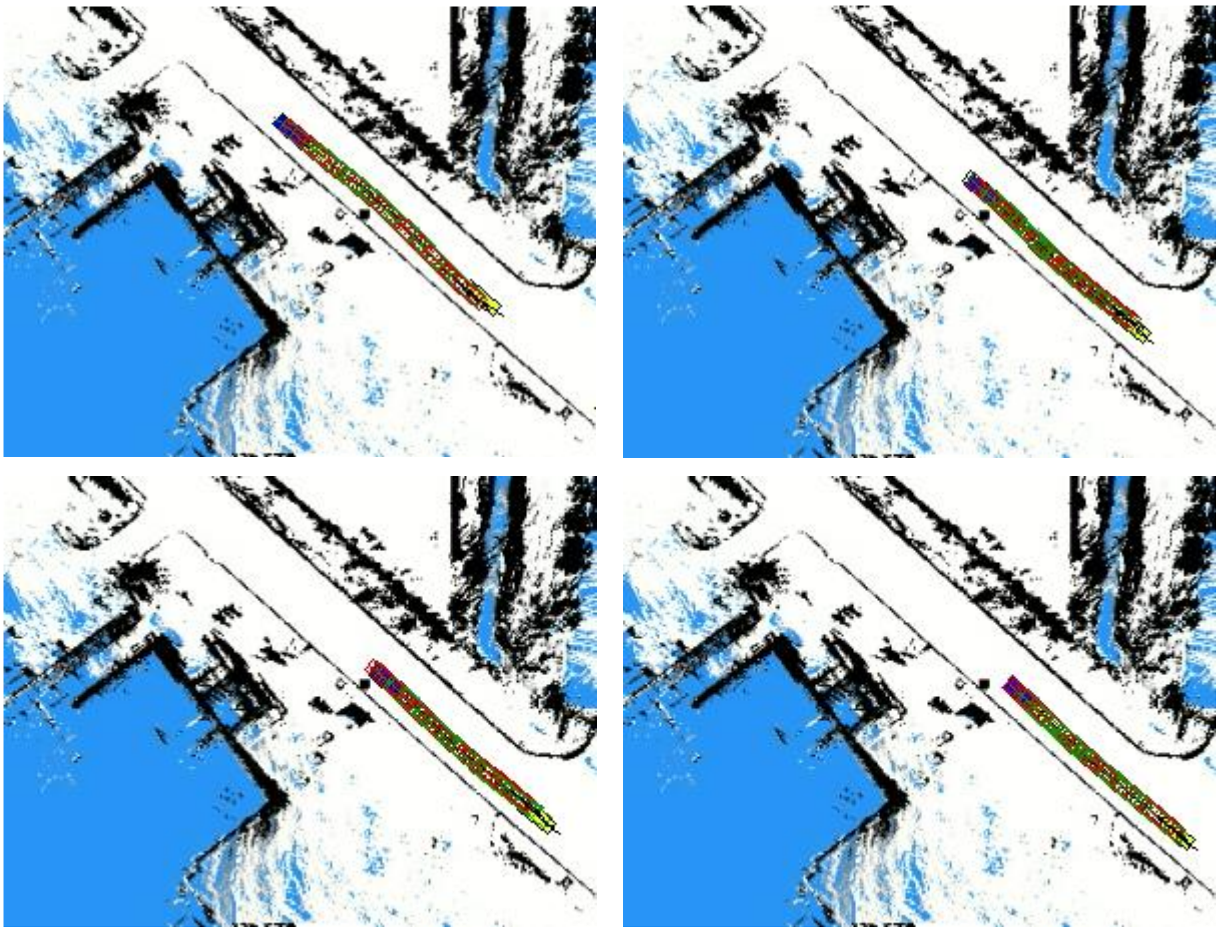


Figure 32: MPMP trajectories (in green/red) avoiding obstacle on the side of the road.

6.1 Discussion

The main contribution of this work is a motion planner for the IARA autonomous car that can operate in real-time at urban on-road scenarios.

The first experimental evaluation showed that MPMP is able to compute trajectories that precisely follow a path produced by a Human driver – the average distance between MPMP and Human paths was 0.15 m (σ = 0.14). The second and third experimental evaluations showed that MPMP is capable to maintain the car's speed close to the programmed velocity. Furthermore, MPMP speed was more stable than the Human speed throughout the course. The third experimental evaluation showed that MPMP is able to maintain the IARA on the lane, while keeping a safe distance from obstacles.

Although this work presented satisfactory results, some issues are worth discussing, which are highlighted bellow.

- The weights of the cost function (Equation (13)) are hard to tune, because it is necessary to find a balance among the optimization criteria. For example, if an obstacle is blocking part of the lane, to get around the obstacle, the weights have to be balanced in a way that the cost of getting out of the lane does not surpass the cost of avoiding the obstacle. Furthermore, the trajectory has to be kept smooth and the goal has to be achieved.

- The MPMP does not explicitly consider moving obstacles. However, moving obstacles are mapped as static obstacles in a high frequency mapping or filtered out by the high level decision maker. Thus, along with the high frequency motion planning, it is possible to avoid them.

- The collision-check in the optimization process is computationally expensive. Due to this, a different representation of moving obstacles will be needed, in order to incorporate constraints associated with them in our optimization model and model-predictive motion planning algorithm. Currently, moving obstacles can cause sudden reactions, because the MPMP does not predict their trajectory.

- A more precise Hardware Platform model is desired to achieve higher velocities. However, its computational cost needs to be the same or lower than the computational cost of the current one, in order to avoid the increase of the computational cost of the trajectory predicting phase during the optimization process.

7 CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this work, we have presented the Model-Predictive Motion Planner (MPMP) developed for the Intelligent Autonomous Robotic Automobile (IARA). MPMP is a high frequency motion planner that operates at 20 Hz and is capable of generating smooth trajectories that follow a reference path while avoiding occasional obstacles. We have tested MPMP running in IARA in a challenging course of 3.7 km and compared its performance with that of a Human in the same course (we asked IARA to follow the Human path as close as possible). Our results showed that MPMP compares well with the Human performance – its path is smooth, very close to the Human path (average distance of 0.15 m, $\sigma = 0.14$) and its speeds are more stable than that of the Human driver. Besides, MPMP trajectories obey the restrictions imposed by obstacles in the road and the platform's performance limits (speed, acceleration, rate of driving wheel turn, etc.). Currently, MPMP can safely navigate in urban environments with speeds of up to 32.4 km/h (9 m/s), while performing very close to Human drive behavior.

7.2 Future Work

In future work, we will:

- Investigate how to improve MPMP and IARA's low level control to improve its maximum speed.
- Examine how to incorporate constraints associated with moving obstacles in our trajectory optimization model and model-predictive motion planning algorithm.
- Improve the tune of the weights of the cost function.
- Investigate the impact of high speed on the actual model.

- Use a more precisely Hardware Platform Model compatible with higher speeds.
- Upgrade the IARA’s control hardware, changing part of the TORC automation.
- Develop a system for handling pedestrians.
- Build a reemission map for road detection and path extraction.
- Use offline map information to anticipate adjustment of velocities on curves and narrow streets.

8 PUBLICATIONS

Two publications were produced during the period of the master's program. The first is a direct result of the research presented here and also a partial requirement for obtaining the master degree in informatics. The second publication is related to other research in development in our laboratory.

- V. Cardoso, J. Oliveira, T. Teixeira, C. Badue, F. Mutz, T. Oliveira-Santos, L. Veronese and A. F. De Souza, "A Model-Predictive Motion Planner for the IARA Autonomous Car", 2017 IEEE International Conference on Robotics and Automation (ICRA 2017), Singapura.
- F. Mutz, V. Cardoso, T. Teixeira, L. F. R. de Jesus, M. A. Golçalves, R. Guidolini, J. Oliveira, C. Badue, A. F. De Souza, "Following the Leader using a Tracking System based on Pre-trained Deep Neural Networks", 2017 IEEE 30th International Joint Conference on Neural Networks (IJCNN 2017), Anchorage, Alaska, USA, 2017.

9 REFERENCES

[BAU15] D. G. Bautista, J. Pérez, V. Milanés and F. Nashashibi, “A Review of Motion Planning Techniques for Automated Vehicles”, IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 4, pp. 1135 – 1145, 2015.

[BUE07] M. Buehler, K. Iagnemma, S. Singh, (Eds.). “The 2005 DARPA Grand Challenge: The Great Robot Race”, Springer, 2007.

[BUE09] M. Buehler, K. Iagnemma, S. Singh, (Eds.). “The DARPA Urban Challenge: Autonomous Vehicles in City Traffic.” Springer, 2009.

[CAR17] V. Cardoso, J. Oliveira, T. Teixeira, C. Badue, F. Mutz, T. Oliveira-Santos, L. Veronese and A. F. De Souza, “A Model-Predictive Motion Planner for the IARA Autonomous Car”, 2017 IEEE International Conference on Robotics and Automation (ICRA 2017), Singapura, pp. 225-230, 2017.

[CHO05] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki and S. Thrun, “Principles of Robot Motion: Theory, Algorithms and Implementations”, Massachusetts, The MIT Press, 2005.

[CUR08] P. N. Currier, “Development of an Automotive Ground Vehicle Platform for Autonomous Urban Operations”, M.S. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, 2008.

[DEL13] D. A. De Lima and G. A. S. Pereira, “Navigation of an autonomous car using vector fields and the dynamic window approach”, Journal of Control, Automation and Electrical Systems, vol. 24(1-2), pp. 106-116, 2013.

[DES16] A. F. De Souza, J. R. C. Silva, F. Mutz, C. Badue and T. Oliveira-Santos, “Simulating Robotic Cars Using Time-Delay Neural Networks”, 2016 International Joint Conference on Neural Networks (IJCNN 2016), Vancouver, BC, pp. 1261-1268, 2016.

[DIA12] J. E. A. Dias, G. A. S. Pereira, and R. M. Palhares, “Identificação do modelo dinâmico longitudinal de um carro autônomo”, Anais do Congresso Brasileiro de Automática, pp. 461-468, 2012.

[FER05] D. Ferguson and A. Stentz, “Field D*: An Interpolation-based Path Planner and Replanner”, Robotics search: Results of the 12th International Symposium (ISRR'05), San Francisco, CA, pp. 239-253, 2005.

[FER08] D. Ferguson, T. Howard and M. Likhachev, “Motion Planning in Urban Environments”, Journal of Field Robotics, vol. 25, no. 11–12, pp. 939–960, 2008.

[FER14] L. C. Fernandes, J. R. Souza, G. Pessin, P. Y. Shinzato, D. Sales, C. Mendes, ... and Wolf, D. F., “CaRINA intelligent robotic car: Architectural design and applications”, Journal of Systems Architecture, vol. 60(4), pp. 372-392, 2014.

[GUI16] R. Guidolini, C. Badue, M. Berger and A. F. De Souza, “A Simple Yet Effective Obstacle Avoider For The IARA Autonomous Car”, 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC 2016), Rio de Janeiro, Brazil, pp. 1914-1919, 2016.

[GUI17] R. Guidolini, C. Badue, F. Mutz and A. F. De Souza, “Neural-Based Model Predictive Control for Tackling Steering Delays of Autonomous Cars”, 2017 IEEE 30th International Joint Conference on Neural Networks (IJCNN 2017), Anchorage, Alaska, USA, pp. 4324-4331, 2017.

[HOH03] G. Hohpe, B. Woolf, “Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions”, 1. ed. Addison-Wesley Professional, 2003.

[HON10] L. M. Honório, L. L. Vermaas, L. M. Gonçalves, and M. Vidigal, “Uma metodologia para aprendizado supervisionado aplicada em veículos inteligentes”, XVIII Congresso Brasileiro de Automática, pp. 1028-1035, 2010.

[HOW09] T.M. Howard, “Adaptive Model-Predictive Motion Planning for Navigation in Complex Environments”, Ph.D. Thesis, Carnegie Mellon University, 2009.

[KAT15] C. Katrakazas, M. Quddus, W.-H. Chen and L. Deka, “Real-Time Motion Planning Methods for Autonomous On-Road Driving: State-Of-The-Art and Future Research Directions”, Transportation Research Part C: Emerging Technologies, vol. 60, pp. 416-442, 2015.

[KOG06] D. Kogan and R. Murray, “Optimization-Based Navigation for the Darpa Grand Challenge”, 45th IEEE Conference on Decision and Control (CDC 2006), San Diego, USA, 2006.

[KUW09] Y. Kuwata, J. Teo, S. Member, G. Fiore, S. Karaman, E. Frazzoli, S. Member, J. P. How, “Real-Time Motion Planning with Applications to Autonomous Urban Driving”, IEEE Transactions on Control Systems Technology, vol. 17, no. 5, pp. 1105–1118, 2009.

[LIK09] M. Likhachev and D. Ferguson, “Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles”, International Journal of Robotics Research, vol. 28, no. 8, pp. 933–945, 2009.

[LYR15] L. J. Lyrio, T. Oliveira-Santos, C. Badue and A. F. De Souza, "Image-Based Mapping, Global Localization and Position Tracking Using VG-RAM Weightless Neural Networks," 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, pp. 3603-3610, 2015.

[MAG13] A. C. Magalhães, M. Prado, V. Grassi, and D. F. Wolf, “Autonomous vehicle navigation in semi-structured urban environment”, IFAC Proceedings, vol 46(10), pp. 42-47, 2013.

[MCN11] M. McNaughton, C. Urmson , J. M. Dolan and J. W. Lee, “Motion Planning for Autonomous Driving with a Conformal Spatiotemporal Lattice”, 2011 IEEE International Conference on Robotics and Automation (ICRA 2011), Shangai, China, pp. 4889-4895, 2011.

[MON03] M. Montemerlo, N. Roy and S. Thrun, “Perspectives on Standardization in Mobile Robot Programming: The Carnegie Mellon Navigation (CARMEN) Toolkit”, International Conference on Intelligent Robots and Systems (IROS), Tokyo, Japan, pp. 2436-2441, 2003.

[MUT16] F. Mutz, L. P. Veronese, T. Oliveira-Santos, E. Aguiar, F. A. Auat-Cheeín and A. F. De Souza, “Large-Scale Mapping In Complex Field Scenarios Using An Autonomous Car”, Expert Systems with Applications, vol. 46, pp. 439-462, 2016.

[QUI09] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A. Y. Ng, “ROS: an open-source Robot Operating System”. ICRA workshop on open source software, vol. 3, No. 3.2, p. 5, 2009.

[RAD14] R. R. Radaelli, C. Badue, M. A. Gonçalves, T. O. Santos and A. F. De Souza, “A Motion Planner for Car-Like Robots Based on Rapidly-Exploring Random Trees”, 4th Ibero-American Conference on Artificial Intelligence (IBERAMIA 2014), Santiago, Chile, pp. 469-480, 2014.

[ROM14] R. A. F. Romero et al., “Robótica Móvel”, 1. ed., Rio de Janeiro: LTC, 2014.

[ROS17] ROS.org, “ROS wiki” Available at http://wiki.ros.org/. Accessed on November 26, 2017.

[SAE14] SAE International, “SAE International Standard J3016”, 2014. Available at http://www.sae.org/misc/pdfs/automated_driving.pdf. Accessed on July 20, 2017.

[SHA13] R. Shanker, A. Jonas, S. Devitt, K. Huberty, S. Flannery, W. Greene, ... & J. Moore, “Autonomous cars: Self-driving the New Auto Industry Paradigm”, Morgan Stanley Blue Paper, 2013.

[SIM17] R. Simmons, “The Inter-Process Communication (IPC) System”. Available at http://www.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html. Accessed on August 22, 2017.

[THR05] S. Thrun, W. Burgard and D. Fox, “Probabilistic Robotics”, Massachusetts, The MIT Press, 2005.

[THR06] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, ... and K. Lau, “Stanley: The Robot that Won the DARPA Grand Challenge”, Journal of Field Robotics, vol. 23, no. 9, pp. 661-692, 2006.

[TOR10] TORC Technologies, “ByWire XGV™ User Manual - Hybrid Escape Drive-by-Wire Platform”, TORC Robotics, Version 1.5, Blacksburg, VA, 2010.

[TOR17] TORC Robotics. Available at http://www.torcrobotics.com. Accessed on September 15, 2017.

[VER15] L. P. Veronese, E. de Aguiar, R. C. Nascimento, J. Guivant, F. Auat Cheein, A. F. De Souza and T. Oliveira-Santos, “Re-Emission and Satellite Aerial Maps Applied to Vehicle Localization on Urban Environments”, 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2015), Hamburg, Germany, pp. 4285-4290, 2015.

[VER16] L. P. Veronese, F. A. Cheeín, T. O. Santos, F. W. Mutz, C. Badue and A. F. De Souza, “A Light-Weight Yet Accurate Localization System for Autonomous Cars in Large-Scale and Complex Environments”, 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC 2016), Rio de Janeiro, Brazil, pp. 520-525, 2016.

[WAY17] Waymo, “Waymo Safety Report”, 2017, Available at https://waymo.com/safetyreport/. Accessed on November 27, 2017.

[WHO15] WHO - World Health Organization, “WHO Global Status Report on Road Safety 2015, Geneva, 2015.

[XU12] W. Xu, J. Wei, J. M. Dolan, H. Zhao and H. Zha, “A Real-Time Motion Planner with Trajectory Optimization for Autonomous Vehicles“, 2012 IEEE International Conference on Robotics and Automation Robotics and Automation, Saint Paul, USA, pp. 2061-2067, 2012.

[ZIE14] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. Keller, E. Kaus, R. Herrtwich, C. Rabe, D. Pfeiffer, F. Lindner, F. Stein, F. Erbs, M. Enzweiler, C. Knoppel, J. Hipp, M. Haueis, M. Trepte, C. Brenk, A. Tamke, M. Ghanaat, M. Braun, A. Joos, H. Fritz, H. Mock, M. Hein and E. Zeeb, “Making Bertha Drive – An Autonomous Journey on a Historic Route”, IEEE Intelligent Transportation Systems Magazine, vol. 6, no. 2, pp. 8–20, 2014.