A Robust Motion Planning Approach for Autonomous Driving in Urban Areas

by

Gaston A. Fiore

B.S., Aerospace Engineering with Information Technology
Massachusetts Institute of Technology, 2006

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
[June 2008]
May 2008

Author .....
Department of Aeronautics and Astronautics
May 23, 2008

Certified by .....
Jonathan P. How
Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
Emilio Frazzoli
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Prof. David L. Darmofal
Associate Department Head
Chair, Committee on Graduate Students

A Robust Motion Planning Approach for Autonomous Driving in Urban Areas

by

Gaston A. Fiore

Submitted to the Department of Aeronautics and Astronautics
on May 23, 2008, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

This thesis presents an improved sampling-based motion planning algorithm, Robust RRT, that is designed specifically for large robotic vehicles and uncertain, dynamic environments. Five main extensions have been made to the original RRT algorithm to improve performance in this type of applications. The closed-loop system is used for state propagation, enabling easy handling of complex, nonlinear, and unstable dynamics. The environment structure is exploited during the sampling process, increasing the probability that a given sample will be reachable. Efficient heuristics are employed in the expansion of the tree and a risk penalty is incorporated to capture uncertainty in the environment and keep the vehicle a safe distance away from hazards. The safety of the vehicle is guaranteed with the assumption of no unexpected changes in the environment, which is achieved by requiring that every trajectory sent for execution ends in a state with the vehicle stopped. Finally, risk evaluation follows a lazy evaluation strategy, allowing the algorithm to spend most of the computation time in the expansion step. The effectiveness of the Robust RRT algorithm for planning in an urban environment is demonstrated through numerous simulated scenarios and real data corresponding to its implementation in MIT's robotic vehicle that competed in the DARPA Urban Challenge.

Thesis Supervisor: Jonathan P. How
Title: Professor of Aeronautics and Astronautics

Thesis Supervisor: Emilio Frazzoli
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgments

For the kids of today, who will probably see robotic cars as commonplace when they reach my age.

Thanks to *Jon and Emilio* for their advice and insights, as well as to *John* for his constructive feedback.

Thanks to my loving family, *Esteban, Maria Luisa, and Romina*, for their moral support. Thanks also to *my friends*, who from time to time provided me with joyful respites in a taxing project.

Thanks to the whole MIT DGC Team, in particular *Yoshi, Justin, David, Ed, Albert, and Sertac.* I have learned a lot from you and I greatly enjoyed the time we spent together in creating Talos.

Go as far as you can see; when you get there, you'll see farther.

Thomas Carlyle

With all humility, I think "Whatsoever thy hand findeth to do, do it with thy might" infinitely more important than the vain attempt to love one's neighbor as oneself. If you want to hit a bird on the wing, you must have all your will in a focus; you must not be thinking about yourself, and, equally, you must not be thinking about your neighbor; you must be living in your eye on that bird. Every achievement is a bird on the wing.

Oliver Wendell Holmes

Contents

1 Introduction **17**

1.1 What Is an Autonomous Automobile? 17

1.2 Why Develop Autonomous Automobiles? 18

1.3 The DARPA Urban Challenge 20

1.3.1 National Qualification Event 22

1.3.2 Final Event 23

1.4 Thesis Structure 27

2 Motion Planning **29**

2.1 Basic Ingredients of Planning 29

2.2 The Configuration Space 30

2.3 Algorithmic Considerations 31

2.3.1 Algorithm Completeness 31

2.4 The Motion Planning Problem 32

2.4.1 Obstacle Region for a Rigid Body 32

2.4.2 Basic Motion Planning 34

3 Existing Approaches for Motion Planning **35**

3.1 Combinatorial Motion Planning 35

3.2 Sampling-based Motion Planning 37

3.3 Approaches Used in the DARPA Grand Challenge 39

3.3.1 Offline Smoothing plus Online Search of 2D Space of Maneuvers 41

3.3.2 Manual Preprocessing plus Online Conformal Search 42

3.3.3 Receding Horizon Control 43

3.4 Conclusion on Available Alternatives 44

4 Robust RRT Algorithm **47**

4.1 Motion Planning Framework 47

4.1.1 System Dynamics 47
4.1.2 Urban Driving Problem Formulation 49
4.2 Algorithm Overview 51
4.2.1 Original RRT Algorithm 51
4.2.2 Robust RRT Algorithm 52
4.3 Use of Closed-loop System 56
4.3.1 Repropagation 58
4.4 Exploitation of Environment Structure in Sampling 59
4.4.1 Sampling the Space 60
4.4.2 Physical Environment Structure as a Bias 61
4.4.3 Logical Environment Structure as a Bias 63
4.4.4 Batch Sampling 65
4.5 Risk Evaluation with Low Computational Overhead 69
4.5.1 Configuration Space Cost Map 69
4.5.2 Risk Evaluation Strategy 71
4.5.3 Lazy Evaluation 72
4.6 Efficient Expansion of the Tree 74
4.6.1 Evaluation of the Distance to Samples 74
4.6.2 Expanding the Tree 76
4.6.3 Estimates of the Cost-To-Go 78
4.7 Construction of a Safe and Rich Tree 79
4.7.1 Improving the Richness of the Tree 79
4.7.2 Safety as an Invariant Property 81
4.7.3 Safe Maximum Speed and Lateral Acceleration 82

5 Robust RRT Evaluation **85**
5.1 Simulation Data 86
5.1.1 Simulated Scenarios 86
Scenario 1: Full Parking Lot 88
Scenario 2: Blocked Road 91
Scenario 3: Rectangular Track with Obstacles 93
Scenario 4: Dense Obstacle Field 95
Scenario 4.1: Dense Obstacle Field, 0° Initial Orientation . . . 97
Scenario 4.2: Dense Obstacle Field, 90° Initial Orientation . . 101
Scenario 4.3: Dense Obstacle Field, 270° Initial Orientation . 105
Scenario 4.4: Dense Obstacle Field, Different Initial Location 109

Scenario 4.5: Dense Obstacle Field, Different Hazard Weights 113
Scenario 5: Narrow Passage 117
Scenario 6: Dead End 122
5.2 DARPA Urban Challenge Data 130
5.2.1 National Qualification Event 130
Parking 130
Road Blockage 131
Lane Following 131
5.2.2 Urban Challenge Event 136
Interaction with Virginia Tech's Odin 136
Encounter of a Gate Blockage 139
First Interaction with CarOLO's Caroline 142
Second Interaction with Caroline 146
Interaction with Cornell's Skynet 151

6 Conclusions and Future Work **157**

References **160**

List of Figures

1-1 LUX is capable of autonomous driving yet it looks like a regular production VW Passat. 18
1-2 MIT's Talos driving in DARPAtown. 21
1-3 Austin Robot Technology's Marvin the Land Robot waiting to merge into traffic at one of the two T intersections in Area A. 23
1-4 Team Autonomous Solutions' Ted driving along an S-shaped road in Area B. 24
1-5 Team Jefferson's Tommy Jr. at a busy intersection in Area C. 25
1-6 Talos driving along a two-way road during the Final Event. 26
1-7 Talos system architecture with motion planner module highlighted. . 28

2-1 The motion planning problem is to find a path from q_I to q_G in $\mathcal{C}_{free}$. The entire region represents the configuration space $\mathcal{C} = \mathcal{C}_{free} \cup \mathcal{C}_{obs}$. . 33

3-1 The sampling-based planning philosophy uses collision detection as a "black box". In this way, it separates motion planning from the particular geometric and kinematic models. C-space sampling and discrete searching are performed. 37
3-2 Stanley, Stanford's robot that won the 2005 DARPA Grand Challenge. 40

4-1 Vehicle trajectories are generated by using the vehicle's dynamical model. The generated trajectories are then evaluated for physical feasibility. 53
4-2 Closed-loop prediction. 56
4-3 Repropagation from the current states. 59
4-4 Example of the 2D Gaussian sampling. 61
4-5 Location of samples when the car is located in a lane. 62
4-6 Location of samples when the car is located in an intersection. 63
4-7 Location of samples when the car is located in a parking lot. 64

4-8 Sampling biases during a U-turn maneuver. 66
4-9 Snapshot of the cost map. 70

5-1 Planning and control system architecture. 86
5-2 Robust RRT algorithm planning in a full parking lot. Figures show the evolution of the tree as Talos drives towards the parking lot exit. 89
5-3 Path length distribution while planning in a full parking lot. 90
5-4 Robust RRT algorithm planning a U-turn maneuver. 92
5-5 Talos went around a rectangular track four times, passing stationary vehicles and performing a U-turn. 94
5-6 The obstacle avoidance scenario used to evaluate the length of paths for different initial conditions. 95
5-7 Planning through a dense obstacle field with an initial condition corresponding to far left and 0°. 98
5-8 Planning through a dense obstacle field with an initial condition corresponding to far left and 0°. 99
5-9 Distribution of path lengths for the initial conditions corresponding to far left and 0°. 100
5-10 Planning through a dense obstacle field with an initial condition corresponding to far left and 90°. 102
5-11 Planning through a dense obstacle field with an initial condition corresponding to far left and 90°. 103
5-12 Distribution of path lengths for the initial conditions corresponding to far left and 90°. 104
5-13 Planning through a dense obstacle field with an initial condition corresponding to far left and 270°. 106
5-14 Planning through a dense obstacle field with an initial condition corresponding to far left and 270°. 107
5-15 Distribution of path lengths for the initial conditions corresponding to far left and 270°. 108
5-16 Planning through a dense obstacle field with an initial condition corresponding to far top and facing the goal. 110
5-17 Planning through a dense obstacle field with an initial condition corresponding to far top and facing the goal. 111
5-18 Distribution of path lengths for the initial conditions corresponding to far top and facing the goal. 112

5-19 Planning through a dense obstacle field with hazard weight set to $w = 0$ and an initial condition corresponding to far left and 0°. 114
5-20 Planning through a dense obstacle field with hazard weight set to $w = 0$ and an initial condition corresponding to far left and 0°. 115
5-21 Distribution of path lengths for different values of the hazard weight w and initial conditions corresponding to far left and 0°. 116
5-22 Planning in a narrow passage scenario with entrance segment set to 10 m. 118
5-23 Planning in a narrow passage scenario with entrance segment set to 15 m. 119
5-24 Planning in a narrow passage scenario with entrance segment set to 20 m. 120
5-25 Success rate of the algorithm for a narrow passage with varying length of the entrance segment into the passsage. 121
5-26 Planning in a dead end scenario with dead end width set to 50 m. . . 123
5-27 Planning in a dead end scenario with dead end width set to 50 m. . . 124
5-28 Planning in a dead end scenario with dead end width set to 75 m. . . 125
5-29 Planning in a dead end scenario with dead end width set to 75 m. . . 126
5-30 Planning in a dead end scenario with dead end width set to 100 m. . 127
5-31 Planning in a dead end scenario with dead end width set to 100 m. . 128
5-32 Success rate of the algorithm for the dead end scenario for widths of 50 m, 75 m, and 100 m. 129
5-33 Talos parking inside the dirt zone during the third run of the Area B test. 132
5-34 Talos performing a U-turn at the second blockage during the third run of the Area B test. 133
5-35 Talos performing a U-turn at the second blockage during the third run of the Area B test. 134
5-36 Motion planner planning along a lane. 135
5-37 Talos interacts with Virginia Tech's Odin at an intersection. 137
5-38 Talos successfully avoids Odin. 138
5-39 Talos successfully brakes in front of an unexpected gate in the middle of the road. 140
5-40 Talos successfully brakes in front of an unexpected gate in the middle of the road. 141
5-41 First interaction with CarOLO's Caroline. 143

5-42 First interaction with CarOLO's Caroline 144
5-43 Reaction of the motion planner to Caroline's cutting across showing views from the front-center camera in Talos' roof. 145
5-44 Talos approaches Caroline in a zone. 148
5-45 Caroline chokes Talos against a fence inside the zone. 149
5-46 Caroline and Talos make contact in a minor collision. 150
5-47 Talos tries to pass Skynet's chase car as it was moving slowly. 153
5-48 Having passed the chase car, Talos embarks to also pass a stopped Skynet when it starts moving. 154
5-49 Skynet is regarded as a static obstacle and Talos tries to merge back but both cars collide. 155

List of Tables

1.1 UCE timing summary. 27

4.1 Land Rover LR3 characteristical constants. 49

Chapter 1

Introduction

1.1 What Is an Autonomous Automobile?

An autonomous automobile is an automobile (or car) that drives entirely on its own with no human driver and no remote control. Various sensors are used to form a representation of the surrounding environment and various positioning systems determine the precise location of the car in that environment. This information is then used by the artificial intelligence module of the car to create a series of efficient commands that drive the car safely through a course that reaches the desired destination.

Some developers of autonomous cars are also starting to focus on minimizing the number of sensors used and concealing them in strategic places of the car. The objective sought is to create a robotic car with a shape that is practically unmodified from that of the original production car used as a platform. Beyond the aesthetics, the aerodynamic performance of the vehicle can be seriously compromised by numerous protuberances breaking the streamline of the original vehicle. As an example of the progress that has already been made in this direction consider LUX, shown in Figure 1-1, Team LUX's entry for the DARPA Urban Challenge competition [56]. This car can drive autonomously, but it looks like a regular production Volkswagen Passat.

Figure 1-1: LUX is capable of autonomous driving yet it looks like a regular production VW Passat.

1.2 Why Develop Autonomous Automobiles?

First, autonomous automobiles have the potential to significantly decrease the number of road accidents and therefore increase passenger and pedestrian safety. Second, autonomous automobiles, when coupled with an automated routing system, can reduce traffic congestion particularly on highways. Third, some of the technologies can be implemented in today's manned cars in the form of driver assistance systems to aid the driver in preventing imminent collisions. Fourth, the technology can also be easily adapted to other vehicles with major opportunities for positive impact in three main areas: farming, mining, and construction [29, 60].

The vast majority of road accidents are due to human error, with less than 10% caused by vehicle defects [71]. Every year, about 1.2 million people die in traffic accidents worldwide [11]. In the United States, for 2005, there were nearly 6,420,000 car accidents, 2.9 million human injuries, and 42,636 fatalities. The financial cost of these crashes totalled more than $230 billion dollars [11, 24]. In the European Union during 2000, road accidents killed over 40,000 people and injured more than 1.7 million. The direct measurable cost of road accidents is on the order of €45 billion. Indirect costs are three to four times higher. The annual figure is estimated at €160 billion, equivalent to 2% of the EU's GNP [71].

Autonomous cars have certain capabilities that human drivers lack, such as being

able to "see" in the dark and through dense fog. Autonomous cars could also bring other benefits to society in addition to increased safety. They could transport anyone who is not able to drive, such as geriatric grandparents, schoolchildren, or intoxicated teenagers. They could turn commutes into productive time, because all of the vehicles' passengers could now sit back and catch up on work or relax while the robot drives toward the desired destination.

Autonomous cars could follow closely behind other cars in highways without compromising safety and thus increase the number of vehicles per unit area. In order to make this possible, cars need to be able to realize the intentions of the car in front with minimal delay. Developing cars that automatically follow each other at preset small distances will make better use of highway space and reduce traffic congestion. Most luxury car manufacturers have already developed primitive versions of such technologies in the form of adaptive cruise control, which uses radar and laser sensors to maintain a minimum distance from leading cars.

Autonomous cars use a variety of radar, laser scanners, and cameras to detect the edges of the road and other cars. Those technologies are already applied by companies like Cognex and Mobileye to make trucks safer. Lane departure warning systems detect highway lane lines and set off an alarm when a driver begins to weave sleepily over lanes. Sensor manufacturers including Germany's SICK and IBEO, along with car makers like GM, are working on similar sensors for consumer cars to detect pedestrians or other vehicles and automatically stop or slow down the vehicle before a collision can occur.

Repetitive off-road tasks like farming, lacking some of the complexities inherent to urban driving, have already benefitted from automation. Tractor driving, for example, does not involve recognizing different road shapes and widths, nor does it involve avoiding other cars and obstacles. As a result, autonomous tractor-driving technologies have existed for years. Trimble sells a device called EZ-Steer [72] that attaches to a tractor's steering wheel and automatically guides its path to within eight inches.

Caterpillar's bulldozers use some of the world's most advanced applications of

robotic car's technology. One feature, called AccuGrade [32], outfits a bulldozer with GPS and fits laser sensors onto its blade. Using a 3D model of the construction surface, the tractor gains a sense of where dirt should be added or removed. The driver simply moves the vehicle back and forth, and the bulldozer's blade automatically guides itself up and down to create a perfectly flat surface.

Although autonomous farming and construction vehicles seem to be sufficiently mature to be fielded in real applications today, autonomous cars are still far from being commercialized mainly because of the much harder problem at hand. An urban environment is dynamic and uncertain, and accounting for moving obstacles where their intentions are not well known is an extremely hard problem. The need for autonomous vehicles, however, is considerable, particularly from the military. To catalyze research and development of autonomous urban vehicle technology, the Defense Advanced Research Projects Agency (DARPA) created the DARPA Urban Challenge (DUC) competition. MIT, enticed by the challenge posed by the autonomous urban driving problem, formed a team of faculty and students and entered the competition. The DUC is presented in some detail next.

1.3 The DARPA Urban Challenge

The DARPA Urban Challenge was an autonomous vehicle research and development program organized by the Defense Advanced Research Projects Agency (DARPA), which is the central research and development organization for the Department of Defense (DoD). The goal of the program was to develop technologies that will keep warfighters safe off the battlefield and allow human resources to be used more effectively. In the National Defense Authorization Act for Fiscal Year 2001, Public Law 106-398, Congress mandated in Section 220 that "It shall be a goal of the Armed Forces to achieve the fielding of unmanned, remotely controlled technology such that... by 2015, one-third of the operational ground combat vehicles are unmanned." DARPA conducts the Urban Challenge program in support of this Congressional mandate [18]. Hopefully the previous section laid clearly that the application of this

Figure 1-2: MIT's Talos driving in DARPAtown.

technology to civilian vehicles will have similar pervasive repercussions, particularly at improving safety on the roads.

The program was an outgrowth of two previous DARPA Grand Challenge autonomous vehicle competitions. The first Grand Challenge event was held in March 2004 and featured a 142-mile desert course. Fifteen autonomous ground vehicles attempted the course with none finishing. In the 2005 Grand Challenge, four autonomous vehicles successfully completed a 132-mile desert route under the required 10-hour limit, and DARPA awarded a $2 million prize to "Stanley" from Stanford University.

The Urban Challenge Final Event (UCE) was held on November 3, 2007, at the former George Air Force Base in Victorville, California. The competition required teams to build an autonomous vehicle to drive through a simulated suburbia called "DARPAtown". The robots had to be capable of driving in traffic, perform complex maneuvers such as merging, passing, parking, and negotiating intersections, while

executing a military supply mission. The course was approximately 60 miles long (100 km) and the robots had to traverse it in less than 6 hours (therefore having to achieve an average speed of 10 mph). This event was truly ground-breaking as it constituted the first time that autonomous vehicles interacted with both manned and unmanned vehicle traffic in an urban environment [18, 30, 31, 57].

Teams from around the world were whittled down through a series of qualifying steps, beginning with technical papers and videos, and then advancing to actual vehicle testing at team sites. Of the 89 teams to initially apply, 35 teams were invited to the National Qualification Event (NQE), a rigorous eight-day vehicle testing period. The NQE was co-located with the UCE in Victorville. DARPA offered $2 million for the fastest vehicle to complete the given mission, and $1 million and $500,000 for second and third place.

1.3.1 National Qualification Event

The NQE for the Urban Challenge was divided into three separate test areas, each with its own distinct characteristics and set of challenges. These test areas were simply called Area A, Area B, and Area C [18].

Test Area A featured a tight, circulating course with two T-shaped intersections and tested robots on their ability to safely merge into and out of two-way traffic with very fluid flow. The final score on the test was computed based on the number of laps around the circuit that the robot completed as well as how safe the robot behavior was while merging into traffic. Amazingly, in eight days of testing, only one traffic vehicle was actually struck by a robotic vehicle.

Test area B featured a 2.8-mile meandering course that tested robots mainly on their ability to stay within a lane while driving and on their ability to park in a tight parking spot. One section, affectionately termed "The Gauntlet", required the robots to delicately maneuver through a series of parallel parked cars and road obstacles. The parking test required the robots to find an assigned parking spot between adjacent parked cars, then safely pull into and back out of the spot before proceeding on its mission.

Figure 1-3: Austin Robot Technology's Marvin the Land Robot waiting to merge into traffic at one of the two T intersections in Area A.

Test Area C was divided into two different halves. The course consisted of a series of four-way stop intersections and in the second half various road blocks were added along the course. For the first half, each intersection presented its own arrangement of traffic and the robots were tested on their ability to negotiate these intersections. Robots had to recognize the other vehicles, determine the order of precedence, and then safely proceed through the intersection when it was their turn. For the second half of the Area C test, various road blocks were emplaced and the robots were tested on their ability to recognize the road block, execute a U-turn, and dynamically replan a new route to complete their mission.

1.3.2 Final Event

After tallying all of the NQE scores, DARPA announced on November 1, 2007, that 11 teams would be competing in the UCE. And so at 7 am on November 3, one by one, all

Figure 1-4: Team Autonomous Solutions' Ted driving along an S-shaped road in Area B.

11 finalist robots were released from their starting chutes, followed by a chase vehicle equipped with an emergency stop control. Thirty manned traffic vehicles were also released onto the course to increase traffic density. This fleet of Ford Tauruses were retrofitted with safety cages, race seats, fire systems, radios and tracking systems, and were driven by professional drivers. In all, over 50 vehicles, both manned and unmanned, were driving through DARPAtown simultaneously during the final event.

The course for the final event was communicated to the teams in the form of two files, analogous to a map and a specific mission. The technical names for these two files were Route Network Definition File (RNDF) and Mission Definition File (MDF), respectively. The RNDF consisted mainly of a list of waypoints in GPS coordinates describing a specific course, while the MDF consisted mainly of an ordered list of checkpoints that the vehicle had to visit along with speed limits for each section. Upon announcing the finalist selections on November 1, teams were given the RNDF of the final course. However, each team did not receive their MDF until five minutes

Figure 1-5: Team Jefferson's Tommy Jr. at a busy intersection in Area C.

before they launched on race day. With this approach, the teams had no a priori knowledge of their missions, creating a truly autonomous driving test.

After strong starts by all the finalists, by mid-morning almost half of the robots had been removed from the race for a variety of reasons. Terramax went awry in a parking lot and was stopped moments before crashing against the old commissary building. Team UCF pulled off the road and into a carport and was therefore removed from the race by officials. Despite these problems, six teams emerged as strong contenders as they executed their missions, with CMU and Stanford apparently taking the lead. At 1:43 pm, Stanford's "Junior" crossed the finish line first with a run-time of just over four hours. A minute later CMU's "Boss" crossed the finish line. Eventually, six robots would cross the finish line and complete the race.

The final event was not just a timed race, however, and robots were also being judged on their ability to follow California driving rules. DARPA officials analyzed collected data throughout the night, examining each team's infractions and elapsed

Figure 1-6: Talos driving along a two-way road during the Final Event.

run times. At the awards ceremony the next morning, DARPA announced the winning order. Boss from Carnegie-Mellon University took first place and the $2 million, Junior from Stanford University took second place and $1 million, and Odin from Virginia Tech took third place and $500,000. After a substantial time correction to account for an error in the specification of the MDF, Talos, MIT's autonomous Land Rover LR3, placed fourth (fifth to cross the finish line). The precise timings for each of the six finishers and showing MIT's fourth place were officially released by DARPA and are reproduced in Table 1.1.

Note that the Urban Challenge was the first time that MIT participated in a DARPA Grand Challenge. MIT was among the eleven teams selected to receive $1 million from DARPA for technology development. When the Urban Challenge kicked off, DARPA announced an opportunity for teams to receive funding in amounts up to $1 million to develop their autonomous vehicle. Sixty-five proposals were reviewed and evaluated, and 11 recipients were announced, among them being MIT.

Table 1.1: UCE timing summary.

Team	Wall Clock	E-Stop	Adjusted
Tartan Racing	5:17:22	3:52:09	4:10:20
Stanford Racing	5:36:54	4:04:16	4:29:28
Victor Tango	5:46:15	4:09:22	4:36:38
MIT	7:24:02	5:38:21	5:33:50
Ben Franklin Racing	6:41:30	5:11:48	5:41:32
Cornell	7:21:48	5:55:32	6:23:34

1.4 Thesis Structure

Developing a robotic vehicle that could complete the DUC was a major systems engineering effort. It required the development and integration of state-of-the-art technologies in perception, planning, and control [35]. The overall system architecture for MIT's vehicle, a modified Land Rover LR3 named Talos, appears in Figure 1-7. It comprises 10 different subsystems for which a report can be found in [52]. The primary focus of this thesis is on the algorithmic and computational issues of the planning problem associated with the Urban Challenge. The design and implementation aspects of the motion planner used in Talos will be thoroughly described.

This thesis presents a new incremental sampling and searching motion planning algorithm based on Rapidly-exploring Random Trees (RRTs), called Robust RRT, which is designed specifically for motion planning under differential constraints in a cluttered environment. Robust RRT enables the on-line use of RRTs on robotic vehicles with complex, unstable dynamics and significant drift, while preserving safety in the face of uncertainty and limited sensing. This algorithm was implemented as the motion-planning system for Talos. This thesis has the following structure:

- Chapter 2 presents the fundamentals of motion planning and gives an overview of the relevant theoretical background.
- Chapter 3 reviews existing motion planning approaches and discusses their applicability for planning in an urban environment.

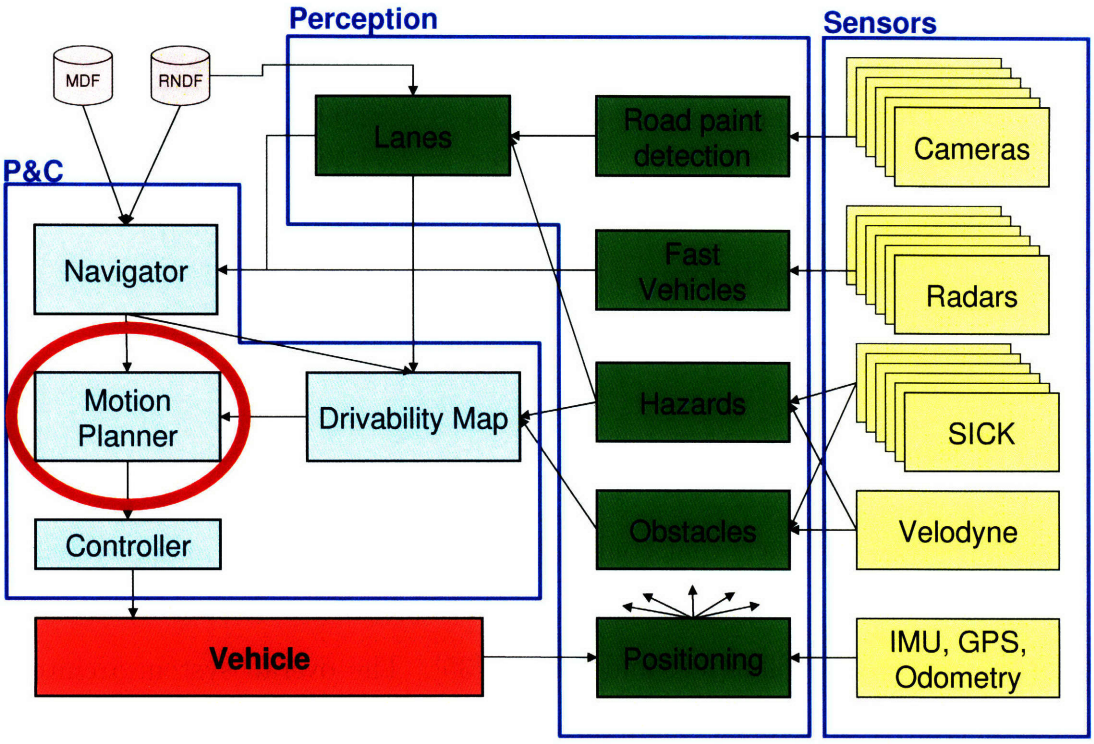


Figure 1-7: Talos system architecture with motion planner module highlighted.

- Chapter 4 presents our main contribution, Robust RRT, which is a new sampling-based motion planning algorithm specifically designed for planning in dynamic, uncertain, and cluttered environments.

- Chapter 5 evaluates the performance of the Robust RRT algorithm on both real and simulated data sets. The real data set corresponds to MIT's autonomous car as it drives during the DARPA Urban Challenge.

- Chapter 6 concludes with a summary of the algorithm presented and outlines further extensions that could be implemented in the future.

Chapter 2

Motion Planning

In robotics, *motion planning* addresses the development of algorithms that convert high-level specifications of tasks for a machine into low-level descriptions of how the machine should move to accomplish those tasks. The motion planning problem is to find a path from one configuration to another in a *continuous state space.* This thesis tackles the motion planning problem from a control theory perspective. In control theory and therefore in this thesis, the motion planning problem is solved by constructing inputs to a (generally nonlinear) dynamical system, which drive the system from an initial state to a specified goal state. This chapter elaborates mathematically the motion planning problem and introduces the technical background needed to understand the material presented in subsequent chapters. The material in this chapter is based on that presented in [48].

2.1 Basic Ingredients of Planning

There are several ingredients that arise throughout virtually all of the topics covered as part of planning [48]. These ingredients are presented in this section in the context of motion planning for urban driving.

State Each distinct situation for the world that could arise is called a *state* and the set of all possible states is called a *state space.* The state space of interest is a

continuous space. The state represents the position $P = (x, y)$, orientation θ, and speed v of the vehicle.

Time The planning problem involves a sequence of decisions that must be applied over *time*. The problem is driving a car as quickly as possible through an obstacle course and therefore time is explicitly modeled.

Actions The plan generates *actions*, as commonly called in artificial intelligence, or *inputs*, as commonly called in control theory, that manipulate the state. We specify how the state changes when actions are applied by an ordinary differential equation. Explicit reference to time is avoided by directly specifying a path through a continuous state space. Such paths are obtained by integrating the differential equations.

Initial and goal states The planning problem involves starting in some *initial state* and trying to arrive at a specified *goal state*. The actions are selected in a way that aims to achieve this goal.

A criterion The *criterion* encodes the desired outcome of a plan in terms of the state and actions that are executed. There are generally two types of planning concerns based on the type of criterion:

Feasibility Find a plan that causes arrival at the goal state, regardless of its efficiency.

Optimality Find a feasible plan that optimizes performance in some carefully specified manner, in addition to arriving at the goal state.

A plan In general, a *plan* imposes a specific strategy or behavior on a decision maker. The plan specifies a sequence of actions for the vehicle to execute.

2.2 The Configuration Space

For the purposes of planning it is important to define the state space. The state space for motion planning is a set of possible transformations that could be applied to the

robot. In the context of planning, the state space is referred to as the *configuration space* and its name is often shortened to *C-space* [48]. Having presented this concept, we simply restrict ourselves to presenting two topological definitions that will later be used in the thesis. The concepts to be presented are *topological graphs* and the *swath of a graph.*

Definition 2.2.1 (Topological graph)**.** Let $\mathcal{X}$ be a topological space. A *topological graph* is a graph for which every vertex corresponds to a point in $\mathcal{X}$ and every edge corresponds to a continuous, injective function $\tau : [0, 1] \rightarrow \mathcal{X}$.

The image of τ connects the points in $\mathcal{X}$ that correspond to the vertices of the edge. The images of different edge functions are not allowed to intersect except at the vertices.

Definition 2.2.2 (Swath of a graph)**.** Let $\mathcal{X}$ be a topological space, $\mathcal{G}(V, E)$ a topological graph. The *swath* S of the graph is defined as

$$S = \bigcup_{e \in E} e([0, 1]), \tag{2.1}$$

where $e([0, 1]) \subset \mathcal{X}$ is the image of the path e.

The swath indicates the set of all points reached by the graph.

2.3 Algorithmic Considerations

2.3.1 Algorithm Completeness

The notion of completeness determines whether an algorithm guarantees finding a solution to a problem instance if that solution exists. There are three different levels of completeness: *completeness*, *resolution completeness*, and *probabilistic completeness* [48]. An algorithm is considered *complete* if, for any problem instance, it terminates in finite time, and either finds a solution, or correctly reports that no solution exists. It is assumed that the problem instance lies within the space of problems for

which the algorithm is designed. An algorithm is *resolution complete* if the algorithm is complete for certain values of a scalar parameter describing the algorithm behavior (e.g., if the resolution is high enough). These values may depend on the instance of the problem. An algorithm is *probabilistically complete* if the probability that it finds an existing solution converges to one as the number of iterations increases.

2.4 The Motion Planning Problem

This section describes the environment with obstacles and defines the basic motion planning problem in that environment.

2.4.1 Obstacle Region for a Rigid Body

Suppose that the two-dimensional world $\mathcal{W} = \mathbb{R}^2$ contains an obstacle region $\mathcal{O} \subset \mathcal{W}$ and that a rigid robot $\mathcal{A} \subset \mathcal{W}$ is defined. Assume that both $\mathcal{A}$ and $\mathcal{O}$ are expressed as semi-algebraic models. Let $\mathcal{C}$ denote the configuration space and $q \in \mathcal{C}$ a particular configuration of $\mathcal{A}$, where $q = (x_t, y_t, \theta)$ [48].

The *obstacle region* $\mathcal{C}_{obs} \subseteq \mathcal{C}$ is defined as the set of all configurations q at which the transformed robot $\mathcal{A}(q)$ intersects the obstacle region $\mathcal{O}$,

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}. \tag{2.2}$$

Since $\mathcal{O}$ and $\mathcal{A}(q)$ are closed sets in $\mathcal{W}$, the obstacle region is a closed set in $\mathcal{C}$.

The configurations outside $\mathcal{C}_{obs}$ define the *free space* $\mathcal{C}_{free}$,

$$\mathcal{C}_{free} = \mathcal{C} - \mathcal{C}_{obs}. \tag{2.3}$$

Since $\mathcal{C}$ is a topological space and $\mathcal{C}_{obs}$ is a closed set, $\mathcal{C}_{free}$ must be an open set. This implies that the robot can come arbitrarily close to the obstacles while remaining in $\mathcal{C}_{free}$. If, however, the robot "touches" an obstacle, then that configuration results in

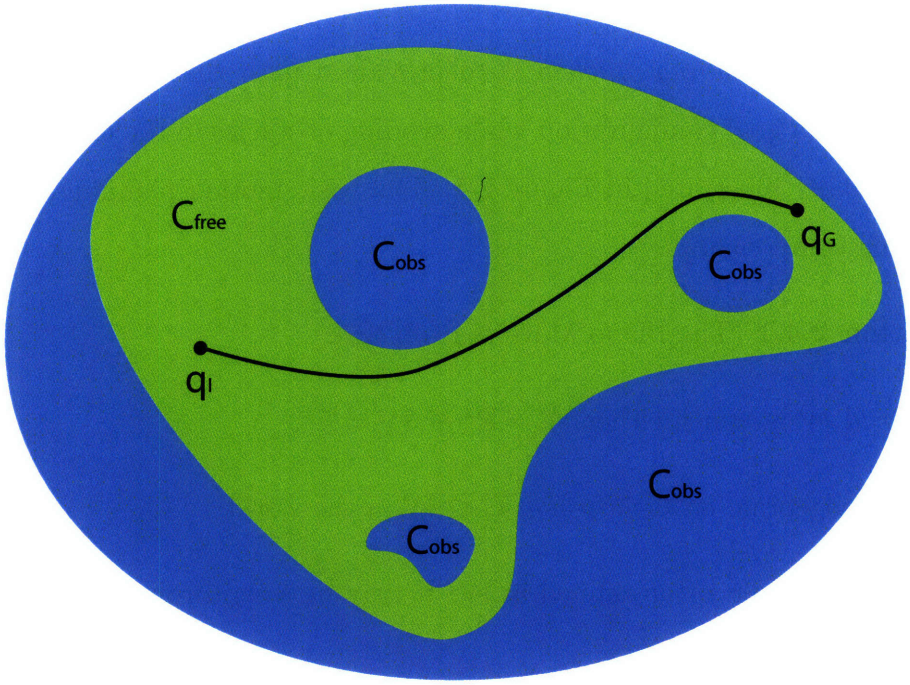


Figure 2-1: The motion planning problem is to find a path from q_I to q_G in $\mathcal{C}_{free}$. The entire region represents the configuration space $\mathcal{C} = \mathcal{C}_{free} \cup \mathcal{C}_{obs}$.

a collision. Mathematically, even if only the boundaries of $\mathcal{A}$ and $\mathcal{C}$ intersect,

$$\text{int}(\mathcal{O}) \cap \text{int}(\mathcal{A}(q)) = \emptyset \tag{2.4}$$

$$\mathcal{O} \cap \mathcal{A}(q) \neq \emptyset, \tag{2.5}$$

then $q \in \mathcal{C}_{obs}$ (in the equation above, int means the interior of the set).

The idea of getting arbitrarily close to obstacles allows for a clean formulation of the motion planning problem, even though this idea might be irrelevant for practical applications. It is important to note that because $\mathcal{C}_{free}$ is open it is impossible to formulate some optimization problems such as finding the shortest path, and the closure $\text{cl}(\mathcal{C}_{free})$ should be used instead [48, Section 7.7].

2.4.2 Basic Motion Planning

The *basic motion planning problem* is to provide a path that goes from an initial configuration to a goal configuration while avoding obstacles in the environment [48]. It is conceptually illustrated in Figure 2-1. The basic motion planning problem has the following components:

Formulation 2.4.1 (The Piano Mover's Problem).

1. A *world* $\mathcal{W}$ in which either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. A semi-algebraic *obstacle region* $\mathcal{O} \subset \mathcal{W}$ in the world.
3. A semi-algebraic *robot* defined in $\mathcal{W}$. It may be a rigid robot $\mathcal{A}$ or a collection of m links $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_m$.
4. The *configuration space* $\mathcal{C}$ determined by specifying the set of all possible transformations that may be applied to the robot. From this, $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ are derived.
5. An *initial configuration* $q_I \in \mathcal{C}_{free}$.
6. A *goal configuration* $q_G \in \mathcal{C}_{free}$.

The initial and goal configurations together are often called a *query pair* or simply *query* and designated as (q_I, q_G). The motion planning problem can then be defined mathematically as follows:

Definition 2.4.2 (Planning in continuous state spaces). A planning algorithm must compute a continuous *path* $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ such that $\tau(0) = q_I$ and $\tau(1) = q_G$.

The main difficulty of the problem is that it is neither straightforward nor efficient to construct an explicit boundary or solid representation of either $\mathcal{C}_{free}$ or $\mathcal{C}_{obs}$. It was shown by Reif [59] that this problem is PSPACE-hard, which implies NP-hard. As a result, algorithms that solve the problem efficiently but not optimally might be sought in order to render its solution tractable.

Chapter 3

Existing Approaches for Motion Planning

There are many algorithms available for solving the motion planning problem in continuous state spaces defined in Definition 2.4.2. All of these approaches, however, belong to either of two main categories: *combinatorial motion planning* and *sampling-based motion planning* [46, 48]. This chapter introduces combinatorial and sampling-based motion planning, presents the main algorithms available within each category, and mentions some of the advantages, disadvantages, and shortcomings of these algorithms for solving the urban motion planning problem to be formally stated in Problem 4.1.1.

3.1 Combinatorial Motion Planning

Combinatorial approaches to motion planning find paths through the continuous C-space without resorting to approximations [48]. Due to this property, combinatorial approaches are alternatively referred to as *exact* algorithms. To solve queries virtually all combinatorial motion planning approaches construct a *roadmap*. A roadmap provides a discrete representation of the continuous motion planning problem without losing any of the original connectivity information required to solve it. A query (q_I, q_G) is solved by connecting each query point to the roadmap and then performing

a discrete graph search on $\mathcal{G}$. Some of the algorithms first construct a cell decomposition of $\mathcal{C}_{free}$ from which the roadmap is consequently derived, whereas other methods directly construct a roadmap without the consideration of cells.

Let $\mathcal{G}$ be a topological graph that maps into $\mathcal{C}_{free}$ and let $S \subset \mathcal{C}_{free}$ bet the swath of the graph. Then a roadmap can be defined as follows [48].

Definition 3.1.1 (Roadmap)**.** A graph $\mathcal{G}$ is called a roadmap if it satisfies the following two important properties:

Accessibility From any $q \in \mathcal{C}_{free}$ it is simple and efficient to compute a path $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ such that $\tau(0) = q$ and $\tau(1) = s$, in which s may be any point in S, the swath of the graph. Usually, s is the closest point to q, assuming $\mathcal{C}$ is a metric space.

Connectivity-preserving Using the first condition, it is always possible to connect some q_I and q_G to some s_1 and s_2, respectively, in S. The second condition requires that if there exists a path $\tau : [0,1] \rightarrow S$ such that $\tau(0) = q_I$ and $\tau(1) = q_G$ then there also exists a path $\tau' : [0,1] \rightarrow S$ such that $\tau'(0) = s_1$ and $\tau'(1) = s_2$. Thus, solutions are not missed because $\mathcal{G}$ fails to capture the connectivity of $\mathcal{C}_{free}$.

By satisfying these properties a roadmap ensures that the algorithm is complete. The first condition ensures that any query can be connected to $\mathcal{G}$ and the second condition ensures that the search always succeeds if a solution exists. Notice that connectivity-preserving ensures that complete algorithms are developed.

When studying combinatorial motion planning algorithms, it is important to carefully consider the definition of the input. The specification of possible inputs defines a set of problem instances on which the algorithm will operate. If the instances have low dimensionality and convexity, then the combinatorial algorithm may provide an elegant and practical solution. If the set of instances is too broad, then a requirement of both completeness and practical solutions may be unreasoble. Many general formulations of general motion planning problems are PSPACE-hard (which implies

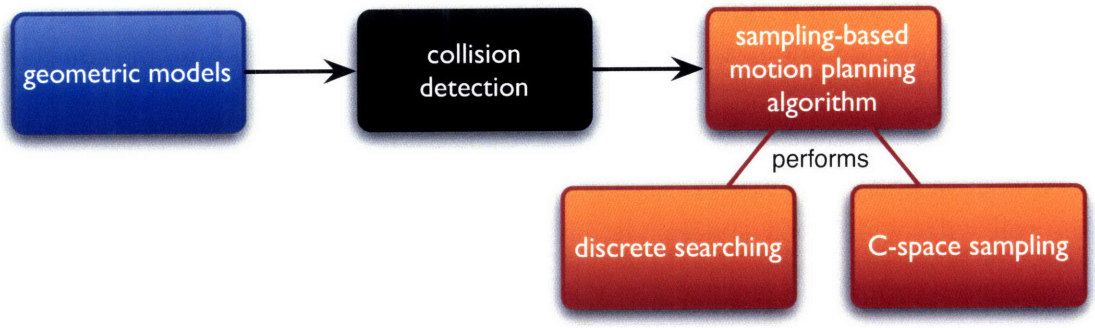


Figure 3-1: The sampling-based planning philosophy uses collision detection as a "black box". In this way, it separates motion planning from the particular geometric and kinematic models. C-space sampling and discrete searching are performed.

NP-hard), and therefore such a hope appears unattainable. Representing the environment in an appropriate way is one of the maincomplications in combinatorial motion planning. Nevertheless, there exist general, complete motion planning algorithms.

Early papers appear in [62]. The classical motion planning textbook of Latombe [45] covers most of the methods in combinatorial planning. An excellent reference for material on computational geometry is the book [19].

3.2 Sampling-based Motion Planning

Sampling-based approaches to motion planning conduct a search that probes the continuous C-space with a sampling scheme in order to avoid the explicit construction of $\mathcal{C}_{obs}$. The idea is outlined in Figure 3-1 [48]. Probing of the C-space is enabled by a collision detection module, which the motion planning algorithm considers as a "black box". This enables the development of planning algorithms that are independent of the particular geometric models. The collision detection module handles concerns such as whether the models are semi-algebraic sets, 3D triangles, nonconvex polyhedra, and so on.

This general philosophy has been very successful in recent years for solving problems that involve thousands and even millions of geometric primitives in such diverse areas as robotics, manufacturing, and biological applications. Such problems would

be practically impossible to solve using techniques that implicitly represent $\mathcal{C}_{obs}$. The incremental sampling and searching framework features two main planning methods, the Probabilistic Roadmap and the Rapidly-exploring Random Tree.

The *Probabilistic RoadMap* (PRM) offers several possible routes and therefore deals well with wide open spaces. It circumvents the computational complexity of deterministic, complete algorithms. The PRM approach was first introduced as a fast and efficient algorithm for geometric, multiple-query motion planning [41]. The original PRM planner is based on an off-line preprocessing phase and an on-line query phase. The preprocessing phase is aimed at constructing a graph of feasible paths in the entire configuration space (the roadmap), which would make future queries easy to solve. The on-line query phase selects a suitable path from the already computed roadmap, together with the computation of two paths to connect starting and ending points to the closest nodes of the roadmap. The PRM algorithm is probabilistically complete. Moreover, performance bounds have been derived as a function of the expansiveness of the environment, meaning the growth rate of the set of points that can be connected to the roadmap with respect to the number of nodes. These bounds prove that the probability of correct termination approaches one exponentially fast in the number of nodes.

For many motion planning applications, such as when dealing with a dynamic and rapidly changing environment, building a roadmap a priori may not be feasible. Another sampling-based motion-planning approach, called *Rapidly-exploring Random Tree (RRT)* addresses this shortcoming. RRTs were introduced in [47, 50]. Many variants of RRTs have been developed and used in several applications [6, 9, 10, 12, 13, 15, 20, 40, 54, 55, 65, 66, 69, 73, 74]. In particular, in autonomous vehicles [25, 42, 53], humanoid robots [38, 39], and computer animation [28]. The RRT approach builds a tree of feasible trajectories on-line by extending branches towards randomly generated goal points, for a subsequent query phase. Although in the PRM approach the idea was to explore the configuration space exhaustively in the preprocessing phase, the RRT algorithm tends to achieve fast and efficient single-query planning by exploring the environment as little as possible. Although most of the applications of RRT

to date have been for ordinary motion planning, they were originally developed for planning under differential constraints and are therefore suitable for solving the urban driving problem.

3.3 Approaches Used in the DARPA Grand Challenge

The DARPA Grand Challenge (DGC) was the predecessor of the DARPA Urban Challenge and its goal was to develop autonomous vehicles capable of traversing unrehearsed off-road terrain [17]. The DGC was arguably a simpler and easier version of the DUC. Nonetheless, they share similarities in many aspects. Consequently, a study of the approaches that some of the teams used to tackle the motion planning problem during the DGC might give some insight on how the DUC planning problem could be tackled.

The first DGC competition carried a prize of $1 million and took place on March 13, 2004. It required the robots to navigate a 142-mile long course through the Mojave desert in no more than 10 hours. 107 teams registered and 15 raced, but none of the participating robots navigated more than 5% of the entire course. The competition was repeated on October 8, 2005, with an increased prize of $2 million. This time 195 teams registered, 23 raced, and 5 finished, with Stanford University's robot "Stanley" finishing the race in 6 hrs, 53 mins, and 58 s and winning the DGC. Figure 3-2 shows Stanley just before traversing the "Beer Bottle" pass.

Teams were required to build an autonomous ground vehicle capable of traversing a desert course up to 175 miles long in less than 10 hours. The first robot to complete the course in under the time limit would win the competition and earn the $2 million prize. Both the 2004 and 2005 races were held in the Mojave desert in the southwest United States. The 2004 course started in Barstow, CA, approximately 100 miles northeast of Los Angeles and finished in Primm, NV, approximately 30 miles southwest of Las Vegas. The 2005 course both started and finished in Primm, NV. The course terrain

Figure 3-2: Stanley, Stanford's robot that won the 2005 DARPA Grand Challenge.

varied from high-quality graded dirt roads to winding rocky mountain passes with only a small fraction of each course featuring paved roads. Absolutely no manual intervention was allowed during the race.

The specific race course was kept concealed from all teams until 2 hours before the race. At this time, each team was given a description of the course on CD-ROM in a DARPA-defined Route Definition Data Format (RDDF). The RDDF was a list of 2,935 waypoints in longitude and latitude coordinates defining the course, each also associated with a corridor width and a speed limit. The width of the race corridor generally tracked the width of the road and varied between 3 and 30 m. The speed limits varied between 5 and 50 mph. The RDDF defined the approximate route that robots would take and therefore no global path planning was required. The robots all competed on the same course at the same time, starting one after another at 5-minute intervals. However, when a faster robot was going to overtake a slower robot, the slower robot would be stopped and passing would occur as if the slower robot were a static obstacle. As a result, the race was primarily a test of high-speed road

finding, and static obstacle detection and avoidance in desert terrain.

The focus in this section is on the DGC 2005 because it is assumed that every team improved or at least did not downgrade the capability of the motion planning algorithms used during the DGC 2004. In the next subsections, the planning approaches for Stanford University, Carnegie Mellon Universiy (CMU), and the California Institute of Technology (Caltech) teams are briefly presented. These three teams were chosen over others due to a combination of performance obtained during the DGC, uniqueness of their approach, and prestige of the researchers involved in the teams. For more information on the approaches of these and the other teams that participated in the DGC 2005, please refer to [33, 34].

3.3.1 Offline Smoothing plus Online Search of 2D Space of Maneuvers

The motion planner for Stanley, Stanford's robot that won the 2005 DGC, was based on the search of a 2D space of maneuvers that spanned the course corridor [68]. Stanley's motion planner software consisted of two parts: the *path smoother*, which generated the base trajectory before the race, and the *online path planner*, which was responsible for determining the actual trajectory of the vehicle during the race. The motion planner was formulated in a perpendicular distance or "lateral offset" to a fixed base trajectory coordinate system. The base trajectory that defines the lateral offset was simply a smoothed version of the skeleton derived from the official race corridor data. Varying the lateral offset generated paths to the left or right of the base trajectory that could be chosen to avoid obstacles along the course. A description of the path smoother is not relevant for comparing different motion planning approaches. Therefore, here it is simply mentioned that the path smoother computed the base trajectory before the race in a four-stage procedure involving optimization and cubic spline interpolation. Stanley's online path planner is described next.

The online path planner was implemented as a search algorithm that minimized a linear combination of continuous cost functions subject to a fixed vehicle model.

The cost functions penalized running over obstacles, leaving the course corridor, and the lateral offset from the current trajectory to the sensed center of the road surface. The soft constraints induced a ranking of admissible trajectories and the best such trajectory was chosen for tracking. At every time step, the planner considered trajectories drawn from a 2D space of maneuvers. The first dimension described the amount of lateral offset to be added to the current trajectory, allowing the vehicle to move left and right while still staying essentially parallel to the base trajectory. The second dimension described the rate at which the vehicle would attempt to change to this lateral offset and controlled the urgency of obstacle avoidance. This change in lateral offset spanned the spectrum of maneuvers appropriate for high-speed obstacle avoidance. There were trajectories that featured fast changes for avoiding head-on obstacles and trajectories with slow changes for smoothly tracking the road center. All candidate paths were run through the vehicle model to ensure they obeyed the kinematic and dynamic vehicle constraints. The motion planner process was executed at 10 Hz.

3.3.2 Manual Preprocessing plus Online Conformal Search

CMU presented two robots in the DGC, Sandstorm and H1ghlander, both of which used the same motion planner based on conformal search [70]. A detailed preplanned path that was computed manually by improving on the official race corridor data served as an accurate baseline. The motion planner consisted of two modules, named *geometric planner* and *speed planner*, that adjusted the preplanned path based on an evaluation of the terrain generated by the perception algorithms. The geometric planner adjusted the path to avoid obstacles and minimized the cost of traversability of the terrain that the robot was driving over. The *speed planner* operated on the output of the geometric planner and pre-emptively slowed the robot if any sharp turns resulted as a consequence of avoiding obstacles in the course. The speed planner was responsible for ensuring that driving speeds were safe. Steering of the vehicle was accomplished by using a pure-pursuit path tracking algorithm.

The geometric planning algorithm was a deterministic heuristic-based algorithm

based on conformal search. A search was constructed relative to the preplanned path that conformed to the shape of the path and constrained the motion of the vehicle. The spacing of the graph along the path was varied to increase stability as the speed increased. The graph was regenerated each cycle and searched using A* to produce an optimal path given the most recent sensor data. The nodes comprising the solution were then connected by straight-line segments. In order to remove the sharp turns that the output path tended to have, a greedy smoothing operator was applied to the path. The smoothed path was only accepted if it had a cost approximately equal to the non-smooth path. In most cases the search operated faster than 20 Hz on the navigation computers but occasionally the search space was too complicated for the search to be computed within a reasonable amount of time. To prevent lockup, the search timed out after a 20th of a second returning the best path found at that time.

3.3.3 Receding Horizon Control

A numerical optimization method lied at the core of the planning system for Alice, the Caltech's robot for the DGC [16]. The motion planner was based on a Receding Horizon Control (RHC) framework. A spline based on the skeleton derived from the official race corridor data was used as the global plan. The RHC solver that produced the higher-quality plan performed its computation to reach a point on the lower-quality path a distance set to the range of the sensors of the vehicle.

The optimization problem consisted in trying to find trajectories that are fastest while keeping the steering and acceleration control effort low. The objective function used was

$$J = S_f \int_0^1 \frac{1}{v(s)} ds + k_1 \|\dot{\phi}(s)\|^2 + k_2 \|a(s)\|^2,$$

where s is the length along the trajectory, S_f is the total length of the trajectory, v is the scalar speed, θ is the yaw (measured from north to east), ϕ is the steering angle, a is the vehicle acceleration, and k_1 and k_2 are tunable parameters representing the weight of the steering rate and accleration terms. A discrete map that represented a spatially dependent speed limit constructed from processing geometrical and terrain

data of the course corridor was used for obstacle avoidance. This meant that areas outside of the course corridor and those that lie inside obstacles were not infeasible, but bear a very low speed limit and are consequently suboptimal. The seeding algorithm used for Alice's planner consisted of a coarse spatial path selector, planning several times beyond Alice's stopping distance. Optimizer convergence speed was a potential issue, but on Alice's 2.2 GHz Opteron, an average rate of 4.28 plans/s was achieved during the race.

3.4 Conclusion on Available Alternatives

Any motion planning algorithm to be used for an autononomous vehicle destined to the DUC needs to be able to handle the car's dynamics easily and be able to plan in an urban environment. Before deciding on which motion planning approach might be better suited for this application, we proceed to list the main challenges present in urban driving. These challenges are:

1. the vehicle dynamics are unstable and complex and suffer from substantial drift,
2. the environment is time-varying and uncertain and the sensing capabilities of the vehicle are limited,
3. the rules of the road impose additional temporal and logical constraints on the vehicle's behavior.

Based on the challenges above, none of the motion planning algorithms used in the 2005 DGC are applicable for an urban environment. In the DGC, the environment was static and rules of the road did not exist. Consequently, a new planning algorithm needed to be developed for the DUC in such a way that it was capable of handling all of the three challenges mentioned.

After a careful analysis of the challenges involved in urban driving, we chose to base the motion planning algorithm dsetined to planning in urban environments on a sampling-based approach. More specifically, the planning algorithm to be described in

Chapter 4 is based on the Rapidly-exploring Random Tree (RRT) concept presented earlier. The main reasons for this choice were:

1. sampling-based algorithms are applicable to very general and possibly nonlinear, unstable, and complex dynamical models,

2. sampling-based algorithms plan incrementally, which facilitates their implementation in real-time, on-line applications, while still retaining certain completeness guarantees,

3. sampling-based algorithms do not require an explicit enumeration of constraints, but instead allow trajectory-wise checking of possibly very complex constraints.

In spite of their generality, the application of incremental sampling-based motion planning algorithms to robotic vehicles with complex and unstable dynamics, such as the full-size Landrover LR3 used by MIT in the DUC, is far from straightforward. For example, the unstable nature of the vehicle dynamics requires the addition of a path-tracking control loop whose performance is generally hard to characterize. Moreover, the momentum of the vehicle at speed must be taken into account, making it impossible to ensure collision avoidance by point-wise constraint checks. In fact, to the best of the author's knowledge, RRTs have never been used in on-line planning systems for robotic vehicles with the above characteristics, but have been restricted either to simulation, or to kinematic, essentially driftless robots (i.e., the robot can be stopped instantaneously by setting the control input to zero).

Chapter 4

Robust RRT Algorithm

This chapter presents the Robust RRT algorithm, an efficient and reliable sampling-based motion-planning algorithm based on RRTs. This algorithm enables the online use of RRTs on robotic vehicles with complex and possibly unstable dynamics and significant drift, while still preserving safety in the face of uncertainty and limited sensing capabilities. The sections that follow describe in detail the main features of the algorithm.

4.1 Motion Planning Framework

Before presenting formally the Robust RRT algorithm, we give an overview of the dynamical system used as the model and provide a formal formulation of the motion planning problem that the algorithm will solve. It is important to note, however, that the Robust RRT algorithm can handle any complex and unstable dynamics and has great flexibility to operate in any cluttered, dynamic, and uncertain environment.

4.1.1 System Dynamics

This section gives an overview of the interface between the motion planner and the controllers as implemented in Talos. As a model for the car dynamics, a nonlinear

bicycle model was used, as presented below [44].

$$\dot{x} = v\cos(\theta) \tag{4.1a}$$

$$\dot{y} = v\sin(\theta) \tag{4.1b}$$

$$\dot{\theta} = \frac{v}{L}\tan(\delta)\cdot G_{ss}(v) \tag{4.1c}$$

$$\dot{\delta} = \frac{1}{T_d}(\delta_c - \delta) \tag{4.1d}$$

$$\dot{v} = a \tag{4.1e}$$

$$\dot{a} = \frac{1}{T_a}(a_c - a) \tag{4.1f}$$

$$a_{\min} \leq a \leq a_{\max} \tag{4.1g}$$

$$\|\delta\| \leq \delta_{\max} \tag{4.1h}$$

$$\|\dot{\delta}\| \leq \dot{\delta}_{\max} \tag{4.1i}$$

The position x and y are defined at the center of the vehicle's rear axle, v and a are the speed and acceleration also at the rear axle, θ represents the direction of the car and is the angle between the x-axis of the body frame and the x-axis of the local frame, δ is the steering angle, and L is the wheelbase. The inputs to this system are the steering angle command δ_c and the longitudinal acceleration command a_c. These inputs go through a first-order lag with time constants T_d and T_a for the steering and the acceleration, respectively. The maximum steering angle is given by $\delta_{\max}$. Because of the particular actuator used in Talos, the steering rate is also limited, and the maximum slew rate is given by $\dot{\delta}_{\max}$. The vehicle has a maximum deceleration $a_{\min}$ and maximum acceleration $a_{\max}$. Values for these constants for the Land Rover LR3 used in the DUC appear in Table 4.1.

The function $G_{ss}(v)$ in Equation 4.1c models the effect of the side slip. It is computed as a steady-state gain of the resulting yaw rate $\dot{\theta}$ when the derivative of the side-slip angle and the derivative of the yaw rate are both zero [1], and is given

Table 4.1: Land Rover LR3 characteristical constants.

Constant	Value
L	2.89 m
T_d	0.05 s
T_a	0.3 s
$a_{\max}$	6.0 m/s^2
$a_{\min}$	-1.8 m/s^2
$\delta_{\max}$	0.544 rad
$\dot{\delta}_{\max}$	0.329 rad/s

by

$$G_{ss}(v) = \frac{1}{1 + (v/v_{CH})^2}$$

The parameter v_{CH} is called characteristic velocity [1, 27] and can be determined experimentally; the value used for Talos was 20 m/s. This side slip model has two main advantages. First, it does not increase the order of the system and therefore it does not increase the complexity of performing the model propagation; in other words, the model behaves the same as the corresponding kinematic model at low speeds. Second, this side slip model is very simple in that it requires only one tunable parameter. Moreover, the model is similar to the nonlinear single track model [1] for the urban driving conditions where extreme maneuvers are avoided and only speeds up to 35 mph are considered.

4.1.2 Urban Driving Problem Formulation

An urban environment adds several additional complexities to the motion planning problem. In particular, in addition to the dynamic and physical constraints typical of robot motion planning with obstacles, an urban environment also features rules of the road, which represent logical constraints on the vehicle's motion. Furthermore, an urban environment typically includes other moving vehicles (and pedestrians), which represent dynamic obstacles moving according to their own control laws and goals,

and so there can be a large amount of uncertainty in their motion. Being a *cluttered, dynamic, uncertain environment with logical rules*, the number of constraints needed to fully represent the urban environment might turn out to be excessive. This important characteristic was taken into account in the decision to solve the urban driving problem by using a sampling-based motion planning approach.

Let $H = [t_0, t_f] \subset \mathbb{R}$ be the planning horizon, and $\boldsymbol{u} : H \to U,\ t \mapsto \boldsymbol{u}(t)$ be a control input signal defined on such horizon; for simplicity, let $U = \mathbb{R}^m$, where m is the number of independent control inputs. Let $\mathcal{U}$ be the set of all allowable control inputs signals, defined, e.g., taking into account control input saturations and rate saturation constraints. A trajectory for the system, under the control input $\boldsymbol{u} \in \mathcal{U}$, can then be represented as a function $\boldsymbol{x} : H \to X,\ t \mapsto \boldsymbol{x}(t)$ satisfying $d\boldsymbol{x}(t)/d(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t))$, where $X = \mathbb{R}^n$ is the n-dimensional state space and $f : X \times U \to X$ is a function describing the (nonlinear) dynamics of the vehicle. Let $\mathcal{X}$ be the set of all trajectories, for all control inputs in $\mathcal{U}$. The environmental constraints on the vehicle, such as avoiding static and dynamic obstacles, as well as the rules of the road, limit the set of allowable vehicle trajectories further. Let $\mathcal{X}_{\text{free}} \subseteq \mathcal{X}$ be the set of all allowable trajectories under such constraints. The urban driving problem can then be defined as follows:

Problem 4.1.1 (Planning in an urban environment.)**.** Given

- A planning horizon $H = [t_0, t_f]$,
- the current vehicle state $\boldsymbol{x}_0 \in X$,
- a specification for a set of allowable trajectories $\mathcal{X}_{\text{free}}$,
- a goal set $X_{\text{goal}} \subset X$,

compute a control input $\boldsymbol{u} \in \mathcal{U}$ such that the resulting trajectory $\boldsymbol{x}$ has the properties:

- $\boldsymbol{x} \in \mathcal{X}_{\text{free}}$,
- $\boldsymbol{x}(t_0) = x_0$,

- $\boldsymbol{x}(t_f) \in X_{\text{goal}}$.

The urban environment is dynamic and uncertain, and therefore the motion plan must be generated online. Furthermore, to quickly react to sudden changes in the environment, the planning time must be as short as 0.1 second. This thesis used a car as the vehicle, but the approach discussed is easily applied to many other types of vehicles.

4.2 Algorithm Overview

Robust RRT is a sampling-based motion planning algorithm based on the Rapidly-exploring Random Tree (RRT) concept [49, 51]. The RRT algorithm generates a tree of dynamically feasible vehicle trajectories by sampling numerous configurations randomly. In order to generate an efficient path in a dynamic and uncertain environment, however, the Robust RRT algorithm builds upon five main extensions that have been made to the RRT approach presented in [25]. These five main extensions are:

1. Use the closed-loop system for state propagation,
2. Exploit the environment structure in sampling,
3. Use efficient heuristics in the expansion of the tree,
4. Consider safety as an invariant property,
5. Risk evaluation following a lazy evaluation strategy.

Each of these five main extensions will be described in detail in their own sections throughout this chapter. Here we limit ourselves to providing a brief overview of the main features of the algorithm.

4.2.1 Original RRT Algorithm

Let $\mathcal{C}$ denote a metric space, $\mathcal{C}_{free} \subseteq \mathcal{C}$ be that part of the space that is free from obstacles, α an infinite sequence of samples in $\mathcal{C}$ drawn from a probability distribution,

and $\alpha(i)$ the ith sample. An RRT is then a topological graph $\mathcal{G}(V, E)$ in which any two vertices $v \in V$ are connected by exactly one edge $e \in E$ [48]. The original RRT algorithm sets a first vertex, or node, containing the initial conditions (x_I, t_I), and incrementally builds a tree of trajectories (edges) that cover the space $\mathcal{C}$. At each step the idea is to add a new trajectory and a new node to the tree. The tree grows in a way that exhibits a fractal appearance. Several main branches will first be constructed as the tree rapidly reaches the far corners of the space, and gradually more area will be filled in by smaller branches until in the limit the tree completely fills the space. In other words, the tree gradually improves the coverage of the space $\mathcal{C}$ as the iterations continue.

The RRT in the presence of obstacles is based on the determination of a sequence of samples $\alpha(i)$ that guide the vehicle to the desired configuration while avoiding obstacles. As with any other sampling-based motion planner, the obstacle region $\mathcal{C}_{obs}$ is not explicitly represented. Therefore, it must be taken into account in the construction of the tree. The procedure that expands the tree yields the nearest configuration possible to the boundary of $\mathcal{C}_{free}$, along the direction towards $\alpha(i)$. The nearest node $(x_n, t_n) \in S$ is defined to be the same (obstacles are ignored in the calculation of the distance to the sample) but, however, in the presence of obstacles the new trajectory might not reach to $\alpha(i)$. In this case, a trajectory is made from (x_n, t_n) to (x_s, t_s), the last configuration possible before hitting the obstacle. It is possible that (x_n, t_n) is already as close as possible to the boundary of $\mathcal{C}_{free}$ in the direction of $\alpha(i)$. In this case, no new node is added for that iteration.

4.2.2 Robust RRT Algorithm

The Robust RRT algorithm assumes the existence of both a high-level route planner that is able to provide a short-term goal and a low-level controller that can track a path and a speed command. The task of the algorithm is then to provide a path and a speed command to the controller that guides the vehicle from its current location to the short-term goal while avoiding obstacles and staying within lane boundaries. To account for the dynamic nature of urban driving, the algorithm must be able to

quickly react to changes in the environment. The information about the environment as perceived by the sensors has inherent noise, and therefore it also must be robust to uncertainties in sensing.

The main steps involved in the Robust RRT algorithm are shown in Algorithm 1. Similar to the original RRT algorithm, Robust RRT samples the space (line 5), selects the best node to connect from (line 6), expands the tree (line 7), and evaluates physical feasibility (line 9), in this order. The tree expansion continues until a time limit is reached (line 23), and at that instant the best reference path is sent to the controller for execution (line 28). As a result, the motion planner sends commands to the controller at a fixed rate. The expansion of the tree is resumed after updating the vehicle states and the environment information (line 2). Figure 4-1 shows an example of a tree generated by the Robust RRT algorithm. As can be seen, the tree consists of reference paths that constitute the input to the controller (shown in blue) and the predicted vehicle trajectories (shown in green and red).

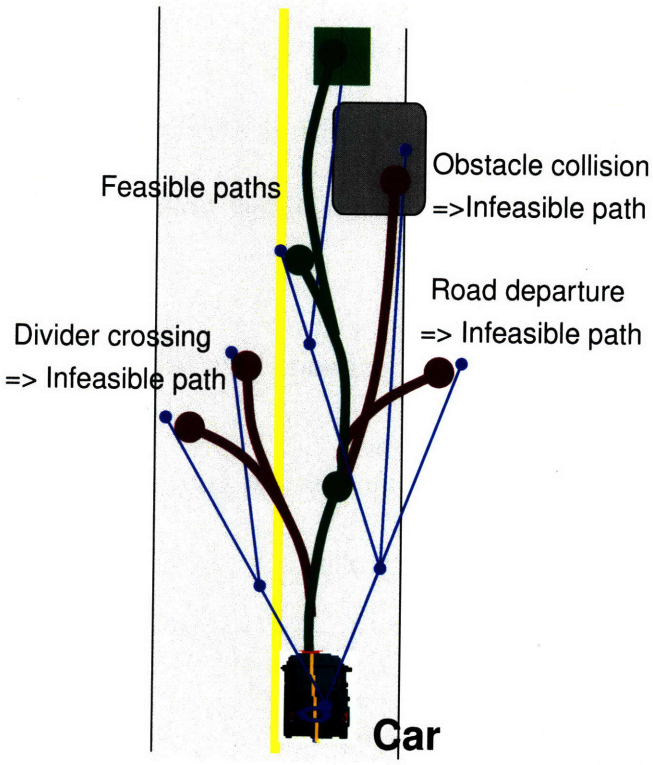


Figure 4-1: Vehicle trajectories are generated by using the vehicle's dynamical model. The generated trajectories are then evaluated for physical feasibility.

Algorithm 1 Robust RRT motion planning algorithm

```
1: repeat
2:   Receive current vehicle states and environment
3:   Propagate states until time limit
4:   repeat
5:     Take sample for input to controller
6:     Select node in tree using heuristics
7:     Propagate from selected node to sample until vehicle stops
8:     Add intermediate nodes to path
9:     if propagated path is feasible with drivability map then
10:      Add sample and intermediate nodes to tree
11:    else
12:      if all intermediate nodes are feasible then
13:        Add intermediate nodes to tree and mark them unsafe
14:      end if
15:    end if
16:    for each newly added node v do
17:      Propagate to goal
18:      if propagated path is feasible with drivability map then
19:        Add path to tree
20:        Set cost of propagated path as upper bound of cost-to-go at v
21:      end if
22:    end for
23:  until time limit is reached
24:  Choose best safe trajectory in tree and check feasibility with latest drivability
     map
25:  if best trajectory is infeasible then
26:    Remove infeasible portion from tree and goto line 24
27:  end if
28:  Send best reference path to controller
29: until Vehicle reaches goal
```

Some of the preceding steps are similar, but nonetheless with notable differences, to the main steps in the original RRT algorithm. First, at each iteration we try to connect the sample to each one of the nodes currently in the tree in turn before discarding the sample as unreachable. On the other hand, in the original RRT algorithm only the closest node is tested for reachability. The RRT criterion of testing the closest node translates into the heuristics of testing the nodes in ascending distance order. Second, the minimum-length Dubins path in the obstacle-free case is used as a measure of distance in the selection of the nodes to expand. On the contrary, the original RRT algorithm expands the node in the tree which has the shortest Euclidean distance to the sample that was picked. Third, although the sequence of samples obeys a random probability distribution, it is deterministically biased in the direction of the goal state by exploiting the environment structure. In the original RRT concept the sequence of samples is completely random, with no deterministic bias.

One fundamental difference with the original RRT algorithm is that the Robust RRT samples the space of inputs to the vehicle controller, as opposed to sampling the space of inputs to the vehicle directly. Each trajectory is obtained by running a forward simulation of the closed-loop system consisting of the vehicle dynamic model and the controller. By using a stable closed-loop system in the simulation instead of the unstable open-loop vehicle dynamic model, this approach has the advantage of being able to incorporate any nonlinear or unstable plant. Furthermore, the vehicle controller generates commands to the vehicle at high rate and therefore the motion planner can send commands to the controller at a low-rate without compromising vehicle stability and path tracking performance.

Each node of the tree has two estimates of the cost-to-go: a lower bound and an upper bound. The lower bound of the cost-to-go is set as the value of the Euclidean distance between the location of the node and the location of the goal. On the other hand, the upper bound is set as the sum of the cost of each edge that corresponds to the best node sequence from the current node to the goal. When no physically feasible trajectory to the goal is found, the upper bound of the cost to go is set to

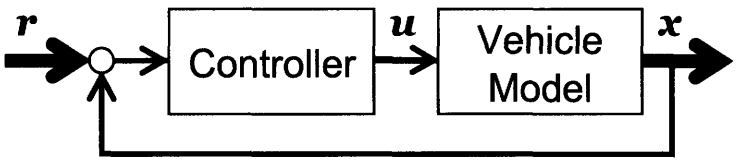


Figure 4-2: Closed-loop prediction.

infinity. When no feasible trajectory is found while the car is already moving, an emergency braking maneuver is commanded, although this is a rare event.

4.3 Use of Closed-loop System

The motion planner outputs a controller reference plan $\boldsymbol{r}$ that constitutes the input to the *stable* closed-loop system consisting of the plant and the controller [44]. This reference plan is constructed based on a quadruple (x, y, v_d, d) at every point in the path, where x and y are the Cartesian coordinates of the point in the plane, v_d is the desired speed of the vehicle at that point, and d is the driving direction of the vehicle (forward or reverse).

While any stabilizing controller can in principle be used for this task, the particular choice of controllers adopted for the DUC consists of a pure-pursuit steering controller and a Proportional-Integral (PI) speed controller [44]. The pure-pursuit controller takes as input a piecewise-linear reference path, and generates steering commands that guide the vehicle along this path. The PI speed controller takes as input a time-varying speed reference and generates gas/brake commands to track this speed.

By running forward simulation using the controller and the vehicle model, the output $\boldsymbol{x}$ of the closed-loop prediction is obtained, as shown in Figure 4-2. The feasibility of this output is checked against vehicle and environmental constraints, such as obstacle avoidance. The planner's role here is to generate a "large" signal in the form of the controller's input, and it is the controller's task to track the commanded path in a "small" signal sense.

This closed-loop approach has several advantages when compared to the standard approach that samples the input $\boldsymbol{u}$ to the vehicle [25, 48]. First, by adding a stabilizing

controller, this approach works with vehicles exhibiting unstable dynamics, such as cars and helicopters. Second, the use of a stabilizing controller reduces the prediction mismatch typically caused by the modeling error of the vehicle. Third, the forward simulation can handle any nonlinear vehicle model and/or controller, and the resulting trajectory is dynamically feasible by construction. Fourth, a single input to the closed-loop system can create a long trajectory (on the order of a few seconds) while the stabilizing controller smoothly changes the input to the vehicle. This significantly improves the efficiency of randomized approaches such as RRT, because it is difficult to generate a good sequence of vehicle inputs if the input to the vehicle is drawn randomly.

When the controller does not accurately track the reference path due to modeling errors or disturbances, the planner could change the reference path so that the vehicle achieves the original desired path. However, it introduces an additional feedback loop, potentially making the overall system unstable. When both the planner and the controller try to correct for the same error, they could be fighting with each other.

In our approach, the planner generates a large signal in the form of a reference path, but it does not do any adjustments. It is the controller's responsibility to track the path in the small signal sense. Thus, the propagation of trajectories (line 7 of Algorithm 1) starts from the *predicted vehicle states* at the node. Although the simulation of the vehicle dynamics uses the closed-loop system, this method decouples the motion planner from the low-level controller. Decoupling the planner and controller eliminates the adverse interactions that could result if the planner continually updates the input to the controller based on the latest states of the vehicle. The predicted trajectory could extend several seconds into the future, and this could be dangerous during the evaluation for collisions if the prediction error is significant. Consequently, it is critical to keep the prediction error as small as possible to ensure that the actual vehicle indeed does not collide with any obstacles along the true path it follows. How to efficiently keep the prediction error small is described next.

4.3.1 Repropagation

Errors in the model of the vehicle and/or the effect of external disturbances inexorably affect the prediction of the vehicle states, resulting in a non-zero error between the predicted path and the true path followed by the vehicle. One possible solution to ensure that no collisions occur when the true path starts deviating from the predicted path would be to delete the tree (keeping the current reference segment being executed by the controller) and grow a new tree from scratch. However, this method is very inefficient. All the computation time spent growing a rich tree would become wasted, and growing a new tree that covers the free space densely will take several cycles. Therefore, deleting the tree is very undesirable, particularly in real-time applications.

To address the errors that result in the vehicle state propagation efficiently, the Robust RRT algorithm performs a *repropagation of the vehicle states*, in which the controller inputs stored in the tree are reused to propagate anew from the current vehicle states. The repropagation process is shown in Figure 4-3. Notice that the repropagated trajectory is only used in the evaluation of collision with obstacles and not for replanning from the current vehicle states, which would result in the motion planner "fighting" the controller with detrimental consequences for performance. The main advantage of the repropagation method is its efficiency in terms of avoiding the deletion and subsequent need for a new tree. In this case, the tree is retained independently of how large or small the error in the prediction is. Using a stable low-level controller, the difference between the original prediction of the vehicle motion and the repropagation will converge to zero in the limit.

When applying the repropagation to the RRT framework, one can repropagate over the entire tree from the latest states. Although it keeps the controller input, this approach is essentially discarding all the state trajectories that are previously computed, and can be computationally expensive. A more efficient approach is to repropagate from the latest states only along the best sequence of nodes at the end of computation time limit, as shown in line 24 of Algorithm 1.

If the repropagated trajectory is collision free, the corresponding controller input

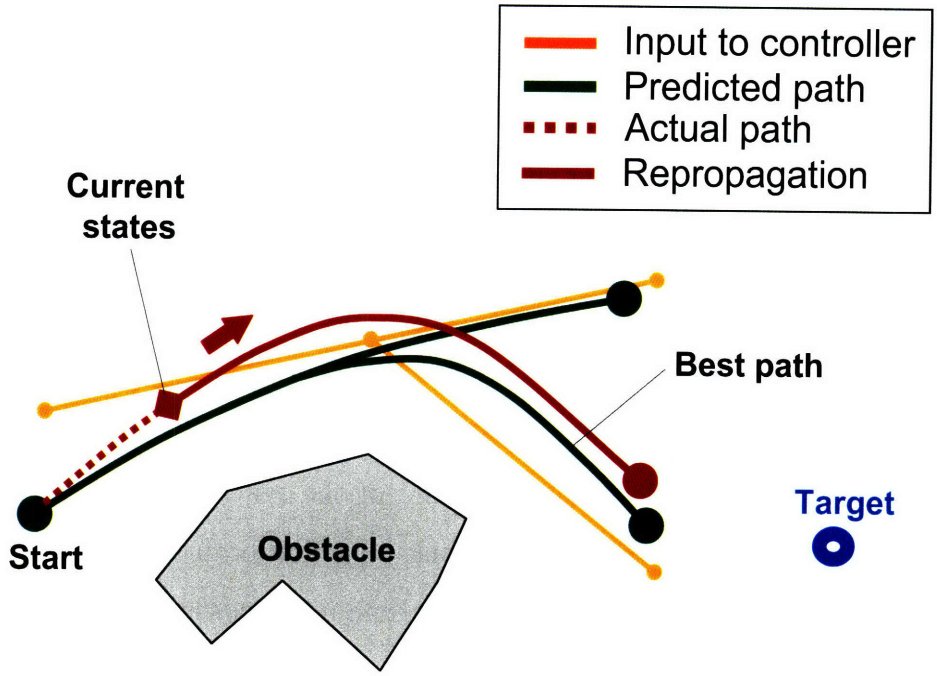


Figure 4-3: Repropagation from the current states.

is sent to the controller. Otherwise, the infeasible part of the trajectory is deleted from the tree, and the next lowest-cost trajectory is selected. This method requires very few re-evaluations for collision with obstacles while ensuring the feasibility of the plan that is sent to the controller, even if the vehicle is not accurately following the original prediction.

4.4 Exploitation of Environment Structure in Sampling

Sampling the configuration space $\mathcal{C}$ blindly without any information about the structure of the environment would result in numerous samples being located over obstacles. This is inefficient as computation time is wasted exploring the obstacle space $\mathcal{C}_{obs}$ that could well be used for sampling the obstacle-free space $\mathcal{C}_{free}$ and generating paths with a very high probability of being collision-free. As a result, methods are sought that would deterministically bias the random sampling of $\mathcal{C}$ in order to focus

as much as possible over the area of the environment that is potentially free from obstacles [54, 55, 65, 69, 74]. The Robust RRT algorithm takes advantage of the knowledge about the location of the vehicle and the structure of the environment to try to focus the sampling over the space that is potentially free from obstacles or other constraints.

4.4.1 Sampling the Space

A sample is a point $S = (s_x, s_y)$ in Cartesian coordinates and is used as part of the input to the steering controller. The sampling of the space occurs according to a Gaussian distribution with a deterministic bias based on both the physical and the logical constraints of the environment. This deterministic bias is achieved by altering the shape and orientation of the Gaussian cloud of samples,

$$\begin{bmatrix} s_x \\ s_y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + r \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \text{ with } \begin{cases} r = \sigma_r |n_r| + r_0, \\ \theta = \sigma_\theta n_\theta + \theta_0. \end{cases} \tag{4.2}$$

where (x_0, y_0) represents the origin of the Gaussian cloud measured in meters, n_r and n_θ are random variables with Gaussian distributions, σ_r is the standard deviation in the radial direction measured in meters, σ_θ is the standard deviation in the circumferential direction measured in radians, and r_0 and θ_0 are offsets with respect to the origin, measured in meters and radians, respectively. Figure 4-4 shows an example with 100 samples and where the different parameters have been set to $(x_0, y_0) = (0, 0)$, $\sigma_r = 10$, $\sigma_\theta = \pi/10$, $r_0 = 3$, and $\theta_0 = \pi/4$. The region enclosed by the solid lines corresponds to one standard deviation in both the radial and circumferential directions. The value of these parameters change according to where the vehicle is located, such as a lane, an intersection, or a parking lot, as well as in conformity with the rules of the road such as whether passing is allowed, U-turns are allowed, etc.

Biasing the sampling significantly increases the probability of generating feasible trajectories because in this way the sampling tends to be confined to the obstacle-free space $\mathcal{C}_{free} \subseteq \mathcal{C}$ and abides by the rules of the road. This increases the likelihood

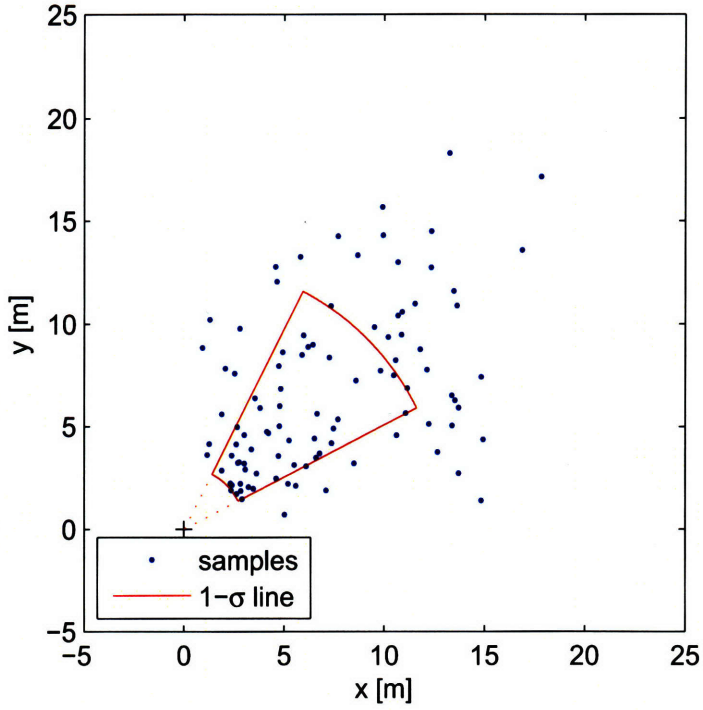


Figure 4-4: Example of the 2D Gaussian sampling.

of generating collision-free trajectories that do not violate any traffic regulations and are therefore feasible. The sampling procedure is therefore both flexible and efficient. It can handle the several different scenarios that occur in an urban environment.

4.4.2 Physical Environment Structure as a Bias

On a lane, the Gaussian distribution is tuned such that the sampling occurs along a long and narrow region whose longitudinal axis tries to follow the estimate of the lane center. To maximize the availability of samples along the lane, a random, uniformly-distributed sampling along a line perpendicular to the lane at every point in the estimate of the lane center and with initial width equal to the estimated width of the lane is also used. The width of the sampling increases slightly with distance from the vehicle to account for uncertainty in the true location of the lane as the sensing range becomes longer. Figure 4-5 shows an example of the region where the samples would tend to be located when the vehicle is driving along a lane. The radial standard deviation value σ_r for the Gaussian distribution used when sampling along

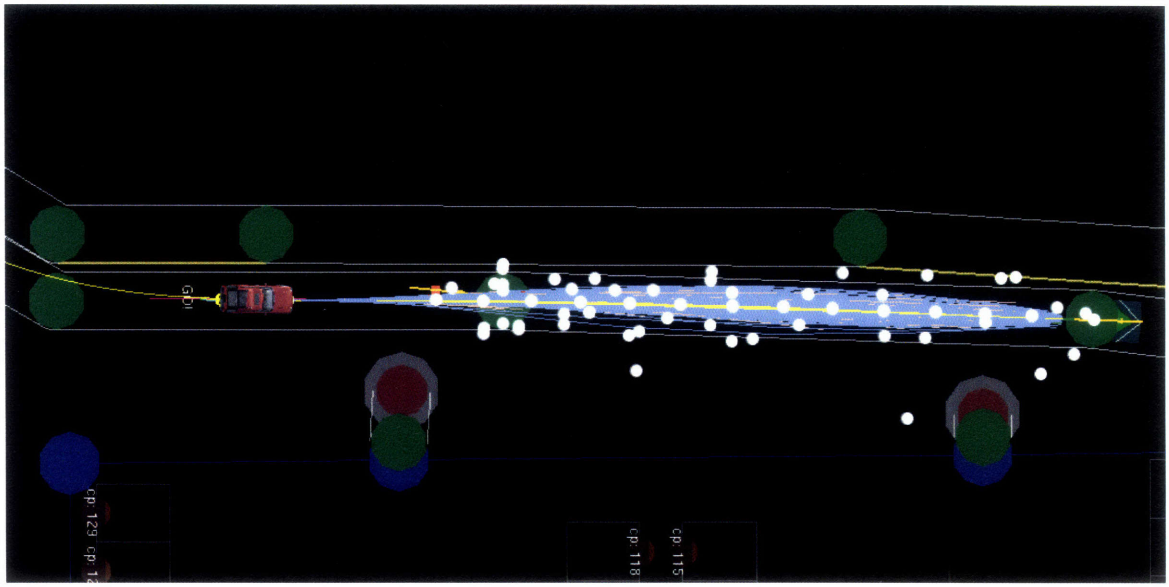


Figure 4-5: Location of samples when the car is located in a lane.

the lane is set to the distance up to the goal but it never extends beyond 50 m. The circumferential standard deviation value σ_θ is set to $\sigma_\theta = \pi/18$ to confine the samples along the lane corridor.

At an intersection, the Gaussian distribution is tuned in a way that the sampling occurs in a wide and relatively short region that covers the open space inside the boundary that delimits a typical intersection. Figure 4-6 shows an example of the region where the samples would tend to be located while the vehicle is traversing an intersection. The value of the radial standard deviation σ_r for sampling in intersections is set to the value of the distance to the goal. The value of the circumferential standard deviation σ_θ is set to $\sigma_\theta = 2\pi/5$.

In parking lots, sampling of the space is performed both around the vehicle and around the parking spot where the vehicle should park. Around the car, the Gaussian distribution is tuned so that the sampling occurs over a wide and long region mostly in the direction that the vehicle is moving, while around the parking spot the sampling is performed according to a uniform distribution along a line that has the same direction as that of the goal inside the parking spot. While the broad sampling around the vehicle aims to facilitate planning in an obstacle maze – as a full parking lot might well resemble – the sampling along a line leading to the parking spot aims to refine

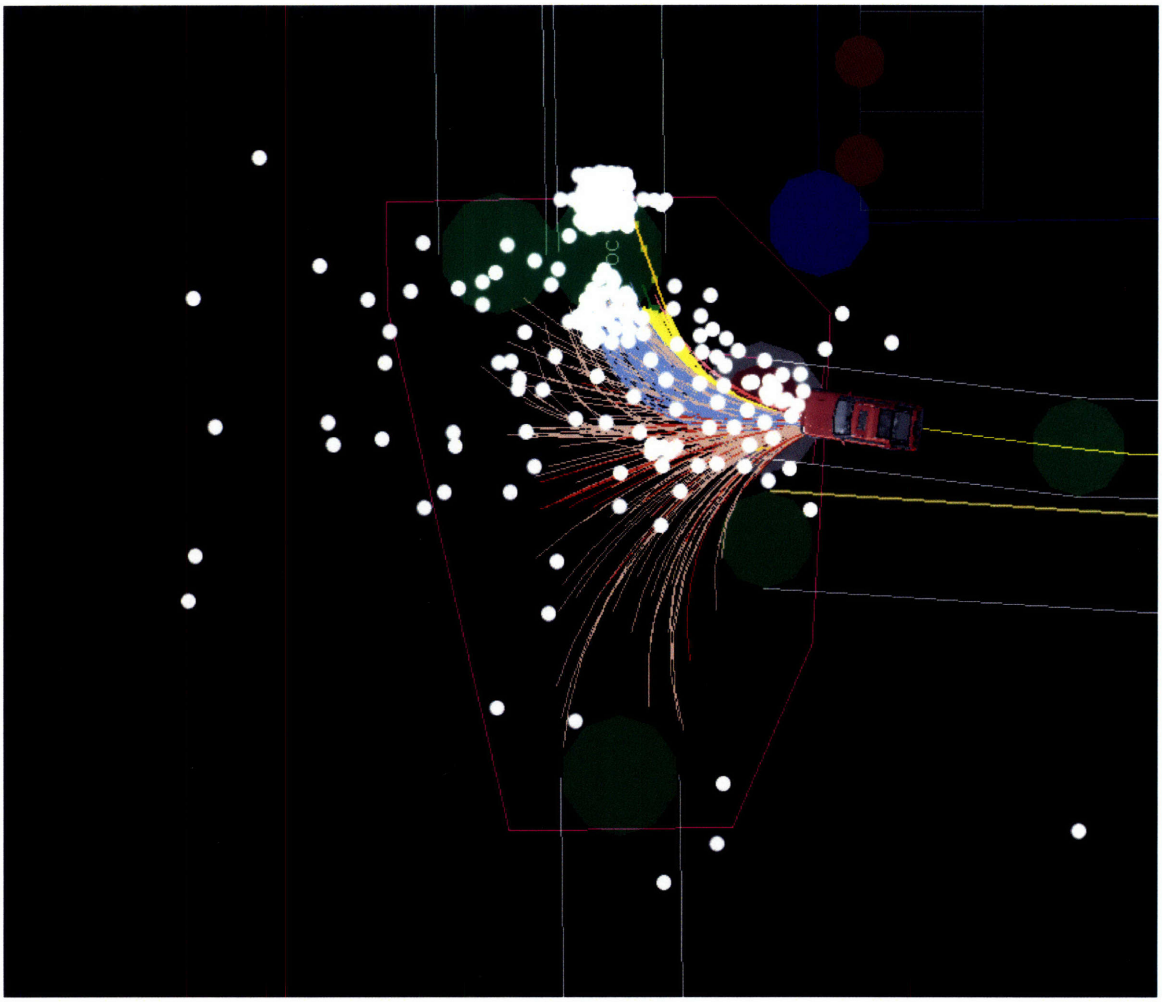


Figure 4-6: Location of samples when the car is located in an intersection.

the trajectories that make the vehicle precisely maneuver into its confined parking place. Figure 4-7 shows an example of where the samples would tend to be located if the vehicle needs to park. The value of the radial and circumferential standard deviations used for the Gaussian distribution when sampling in zones is $\sigma_r = 50$ and $\sigma_\theta = \pi$.

4.4.3 Logical Environment Structure as a Bias

When the vehicle needs to pass another vehicle or some other obstacle on its lane, the sampling is skewed toward the left in order to focus the search for feasible trajectories along the opposite lane of travel. Once a feasible trajectory is found that arrives back to the original lane of travel, this skew is eliminated.

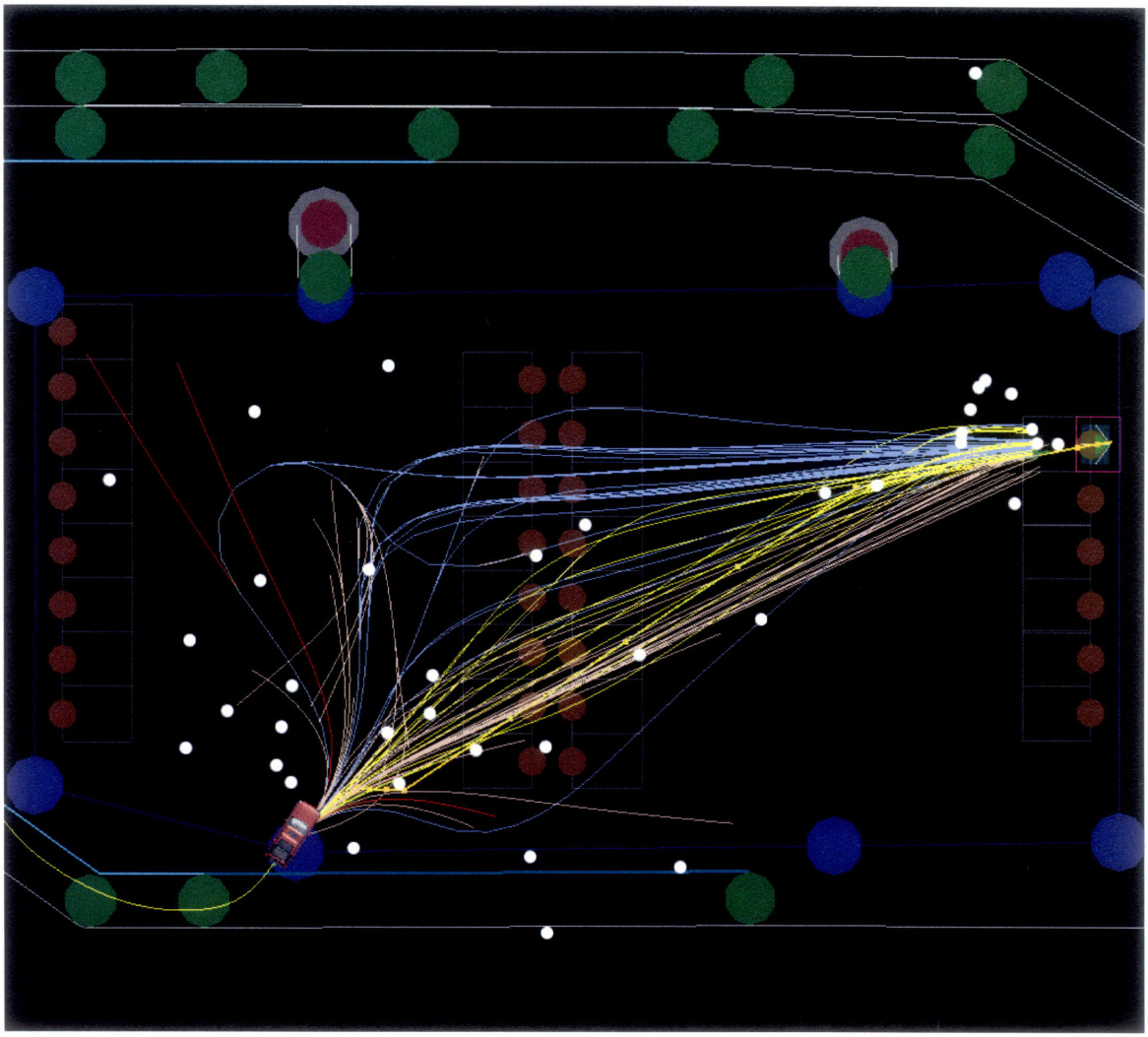


Figure 4-7: Location of samples when the car is located in a parking lot.

When the specific circumstances for a U-turn arise, the planner will search for a maneuver preferably consisting of a three-point turn that could be used to complete the U-turn. In order to facilitate the emergence of a three-point turn, the random, normally-distributed sampling of forward and reverse configurations is confined to specific regions of the space that correspond to plausible regions that the vehicle would have to traverse in case of executing a three-point turn. Therefore, sampling with a forward direction of travel is confined to the front-left and rear-left of the original position of the vehicle before executing a three-point turn, while sampling for traveling in reverse is confined to the front-right and back of the vehicle.

The sampling strategy used for U-turns is shown in Figure 4-8. The location of

the different regions correspond to the different phases that constitute a three-point turn. During a three-point turn, the vehicle first travels forward and to its left, it then reverses aiming the original lane of travel, and finally the vehicle then moves forward towards the opposite lane of travel, now heading in the correct direction of the lane. The parameter values used for each of the three sample sets are:

$$\begin{array}{llll} \sigma_r = 5, & \sigma_\theta = \pi/10, & r_0 = 3, & \theta_0 = 4\pi/9 \quad \text{first cloud} \\ \sigma_r = 5, & \sigma_\theta = \pi/5, & r_0 = 3, & \theta_0 = -\pi/6 \quad \text{second cloud} \\ \sigma_r = 10, & \sigma_\theta = \pi/4, & r_0 = 3, & \theta_0 = 5\pi/6 \quad \text{third cloud} \end{array} \tag{4.3}$$

In the parameters above, the angular offsets σ_θ are measured from the x-axis of the body frame for the car in its original configuration before initiating the U-turn, i.e., with respect to the longitudinal axis of the car with its original orientation before the U-turn. The origin (x_0, y_0) of the Gaussian clouds is at the initial location of the car before initiating the U-turn. Finally, note that an additional cloud of reverse samples is also generated behind the vehicle in case it stopped very close to the road blockage, therefore requiring a reverse maneuver before any U-turn is possible.

4.4.4 Batch Sampling

For the Urban Challenge, the car had to arrive at a given location with a certain direction (heading). Consequently, the goal has a specific direction that, within some threshold, any arriving path needs to satisfy in order for that path to be considered as having reached the goal. To facilitate the satisfaction of this direction constraint, the Robust RRT algorithm features a method of sampling called *batch sampling*. Batch sampling refers to the sampling of multiple points simultaneously, which are then joined to form an edge. In this way, the points forming the edge can be arranged conveniently so that edge features some specific shape. Once the edge has been formed, its first sample is then connected to a node in the tree, resulting in a single edge analogous to any other edge in the tree.

Only sampling in zones, intersections, and U-turns feature batch sampling. Sam-

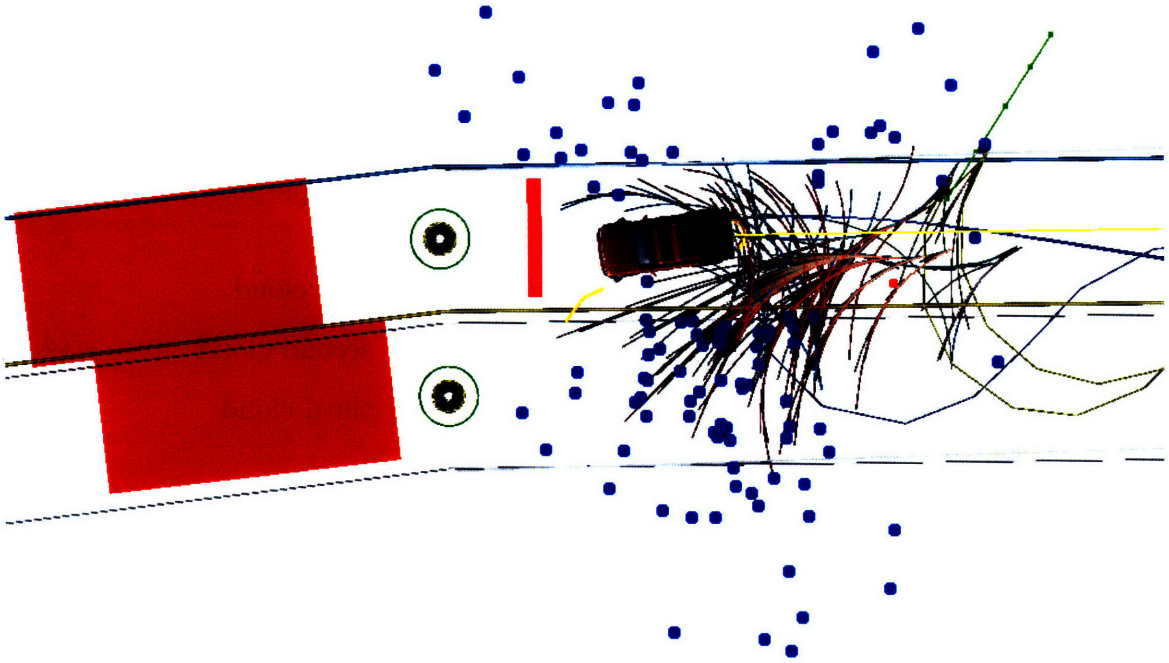


Figure 4-8: Sampling biases during a U-turn maneuver.

pling in lanes does not need batch sampling to improve performance since the free space effectively narrows down the plausible set of paths to those having a direction similar to that of the goal. Batch sampling is usually set to occur less frequently than single sampling because, although it is beneficial for satisfying the direction constraint of the goal, the whole path determined by the samples needs to be feasible. This path is usually longer than the path resulting from a single sample, which means that there is a higher probability that edges formed by batch sampling turn out to be infeasible and are therefore discarded.

In zones, each batch of samples is comprised of four samples. The first sample is drawn out of a Gaussian distribution covering a wide and long region in the direction that the vehicle is moving, similar to the single sampling method, but with a radial standard deviation value of $\sigma_r = 20$ in this case. This lower value is justified since this sample will be the first in a group of four that will gradually guide the vehicle towards the goal, and therefore there is no need for the first sample to be located

excessively far away from the vehicle. The second sample is drawn out of a uniform distribution with a conic shape centered on the parking spot and with vertex 5 m away from the goal, which as a result leads directly to the center of the parking spot. The purpose of this conic sampling leading to the parking spot is to refine the trajectories to those that make the vehicle precisely maneuver into the confined parking place. The third sample is located at the vertex of the cone just described, and, together with the fourth sample, which is located over the goal location, forms the last segment of the path determined by the batch and has the objective of finishing to lead the car straight into the spot.

In intersections, each batch of samples is composed of three samples. Similarly to the single sampling case, the first sample is drawn out of a Gaussian distribution covering a wide and short region, where the standard deviation values are now set to $\sigma_r = 10$ and $\sigma_\theta = \pi/9$. This region would cover most of the space inside the boundary that delimits a typical intersection. The second sample follows a uniform distribution with a conic shape centered at the beginning of the new lane after the intersection. This cone leads to the center of the new lane and has as purpose to ensure that all paths resulting from the batch sampling meet the goal direction constraint. As a result, for the implementation of the Robust RRT algorithm used in the DUC, the angle of the vertex located at the center of the new lane was set to 60° to match the direction error threshold of the goal, which was ±30°. The third and last sample is drawn out of a uniform distribution and is confined to a square centered on the lane and positioned such that one of its edges perpendicular to the lane pases over the goal and the opposite edge is closer to the intersection. This square can be easily see in Figure 4-6. It has a side length of 2 m so it can cover most of the width of a regular lane.

The purpose of randomizing the location of the last sample is to account for uncertainty in the sensed environment. For example, suppose that the road that the car will be turning to has a wall directly over one of its sides. If the wall is perceived as closer to the road than what it really is, there is a high probability that it might be perceived as partially blocking part of the road. If the last sample in the batch were

located exactly over the goal, it might be the case that all of the resulting paths are infeasible because the side of the car touches the perceived wall. As a result, the car will never leave the original lane to go through the intersection as no feasible path is reaching the goal and driving and stopping over an intersection is purposefully not allowed as it is not safe. On the other hand, by randomizing the location of the last sample there will eventually be a feasible path that reaches the goal but closer to the side of the road opposite to the wall. Having a feasible path, the car will proceed normally through the intersection and then keep driving along the new lane.

For U-turns, each batch of samples is composed of four samples. Similarly to the case of single sampling, the location of the different samples corresponds to the different phases that form a three-point turn, with the addition of a last fourth sample to help guide the car forward along the opposite lane. The parameter values used for each of the four sample clouds are:

$$\begin{array}{llll} \sigma_r = 8, & \sigma_\theta = \pi/10, & r_0 = 3, & \theta_0 = 4\pi/9 \quad \text{first cloud} \\ \sigma_r = 10, & \sigma_\theta = \pi/10, & r_0 = 5, & \theta_0 = -\pi/4 \quad \text{second cloud} \\ \sigma_r = 12, & \sigma_\theta = \pi/10, & r_0 = 7, & \theta_0 = 5\pi/6 \quad \text{third cloud} \\ \sigma_r = 10, & \sigma_\theta = \pi/10, & r_0 = 5, & \theta_0 = \pi \quad \text{fourth cloud} \end{array} \tag{4.4}$$

In the parameters above, the angular offsets σ_θ are measured from the x-axis of the body frame, i.e., with respect to the longitudinal axis of the car, for the car in its original orientation. The origin (x_0, y_0) of the first Gaussian cloud resides at the location of the vehicle before initiating the U-turn maneuver, whereas the origins of the other three clouds reside at the location of the sample from the previous cloud, respectively. Sampling of the regions corresponding to each cloud occurs in sequence, i.e., first a sample corresponding to the first cloud is obtained, then this sample is used as a reference for determining the origin of the cloud for the second sample, etc. This sequential procedure increases the probability of obtaining a nice three-point turn, as the location of each individual sample is with respect to each other and not with respect to a global location. Therefore, if one of the samples is far from the mean and thus poorly located in the space, the probability that the next sample will

be inconveniently located with respect to this sample and contribute even more to a poor maneuver is independent of the poor location of the previous sample.

U-turn batch sampling was restricted to only 50 trials in the implementation used in Talos. This is because any portion that is infeasible would result in discarding the whole maneuver. Since U-turns happen near obstacles, this probability might be considerable. Single sampling was used the rest of the time.

4.5 Risk Evaluation with Low Computational Overhead

4.5.1 Configuration Space Cost Map

The evaluation of whether a trajectory collides with obstacles or violates any rule of the road is done through a configuration space *cost map*, also called *drivability map*, in the local frame of reference. This cost map is implemented using a grid-based lookup table with a resolution of 20 cm since this value gave enough accuracy while keeping the table size manageable. As a result, the configuration space takes discrete values and is only approximate, but this limitation is taken into account by ensuring that the representation of the obstacles is never smaller than their perceived size. The cost map processes static and moving obstacle data, lane tracking data, and road surface hazard data to render the environment and classify three different regions according to their drivability characteristics.

The three different regions that the gridmap displays are *infeasible*, *restricted*, and *high cost*. Infeasible regions represent obstacles and are impossible to traverse due to an imminent collision. Restricted regions are used to prevent the vehicle from violating minor traffic regulations, and therefore may be driven through but only if the vehicle can then drive out of them. High cost regions are drivable but incur a penalty on the cost-to-go for being close to obstacles or lane boundaries, thus posing a potential safety hazard. Figure 4-9 shows a snapshot of the cost map. The white arrow with a green background indicates the short-term goal location, red indicates

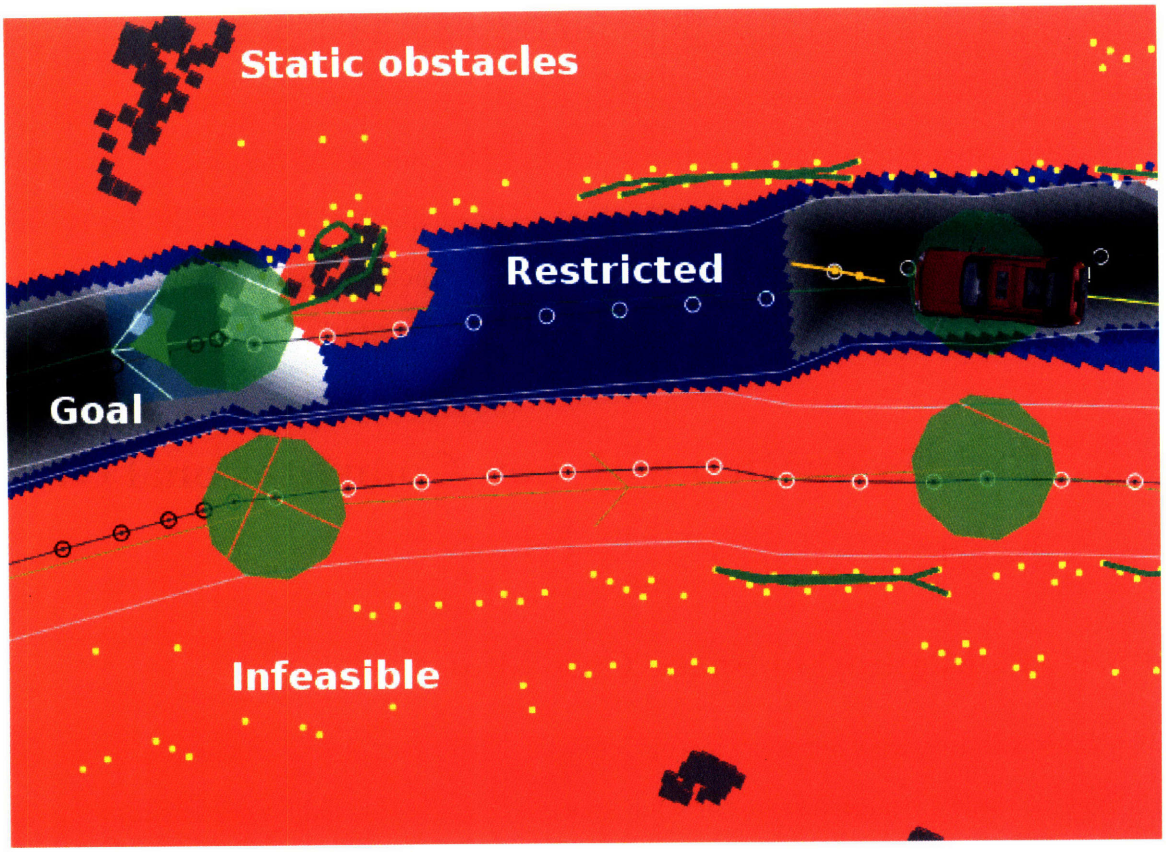


Figure 4-9: Snapshot of the cost map.

the infeasible regions that the vehicle can not reach, blue indicates the restricted regions that the vehicle may only enter if an unrestricted region further ahead can be reached, and white or gray indicates the high cost regions that the vehicle can drive over.

Restricted regions have their main application in ensuring that the car stops with the required standoff distance from obstacles while traveling along a lane. Consider the case where some obstacle is blocking the lane along which the car is traveling and therefore a passing maneuver is necessary. Thanks to the restricted region around the obstacle, the car stops the required distance away because there is no other area where the vehicle can drive to from inside the restricted region. When the opposite lane becomes free to drive, however, the car can then go over the restricted region to pass the obstacle in the lane because now the car is able to leave the restricted region. In this way, restricted regions provide a certain level of constraint without completely banning access to the region they cover. This prevents the vehicle from

potentially getting stuck due to artificial constraints that are not present in the real environment.

4.5.2 Risk Evaluation Strategy

The Robust RRT algorithm uses a *hazard penalty* value to represent the risk due to hazards in the environment such as getting close to obstacles and lane boundaries, or traversing certain curb cuts that are too faint to be declared as non-drivable. This hazard penalty value will then form part of the cost associated with a particular edge. When evaluating the feasibility of trajectories using the cost map lookup table, the penalty for getting close to hazards that is stored in the map is also obtained. A hazard penalty is associated with every point forming a given path. The hazard value of a certain edge (path) is then defined as the path integral of the penalty at each point,

$$h_{edge} = w \int_{l} \rho \, dl, \tag{4.5}$$

where l_{edge} is the length of the edge, ρ is the hazard penalty at each point in the edge, and w is a constant representing the desired weight to give to ensure that the vehicle stays far from obstacles and away from lane boundaries. The weight used for the implementation of the algorithm for the DUC was $w = 0.005$. This value gave enough repulsion from obstacles without compromising the path lengths substantially and, fundamentally, it gave a strong bias for the vehicle to stay close to lane centers, something that produced a nice behavior when turning at intersections because the car tended to avoid cutting corners. The value was determined by trial-and-error until the behavior of the car satisfied the designer. The availability of and edge hazard value results in the car normally maintaining a prudent distance from obstacles and lane boundaries, while still allowing it to get close to constraints when no other alternative is possible such as is the case of narrow roads.

The hazard penalty is represented with a byte and can therefore take values ranging from 0 to 255, where 0 means no penalty and 255 means there is a collision with an obstacle or the vehicle is over a lane boundary or touching any infeasible region.

In the case both an obstacle and a deviation from the lane center where contributing to the penalty value at a given point, the highest value among the two contributions is taken as the hazard penalty at that point. When signaling the danger of a potential collision, the penalty increases as the vehicle gets closer to obstacles. On the other hand, when penalizing the deviation from the lane center, the penalty increases as the perpendicular distance between the lane center and the vehicle increases. In this way, the vehicle will be repelled from getting close to obstacles but it will be attracted to stay close to the lane center as it drives along a lane.

In the implementation of the Robust RRT algorithm, the functions used to represent these two penalties were different. A cosine function with a long decay rate was used to model the cost around obstacles, whereas a linear function with a much shorter tail was used to penalize deviations from the center of the lane. The cosine function ensured that getting very close to obstacles was very costly, which models the very high risk that driving next to obstacles entails due to the danger of a potential collision. The long tail has the effect that, among two trajectories of similar length, the one furthest away from obstacles is preferred and chosen as best. In the case of lane following, getting close to lane boundaries is not desirable but not as dangerous since the space beyond a boundary is usually free, and therefore a linear penalty function with a short tail penalizes gradually as the vehicle deviates from the lane center.

4.5.3 Lazy Evaluation

Sampling-based motion planning algorithms typically spend most of the execution time evaluating trajectories for collision with obstacles, and methods are sought to reduce the number of collision checks [8]. In a dynamic and uncertain environment, the feasibility of each edge in the tree should be evaluated against the latest available sensory information. Because the environment in constantly changing, however, each of the edges in the tree would need to be re-evaluated for feasibility every time that something changes in the environment. A rich tree might have thousands of edges and collision evaluation would therefore require significant computation time. Any

time that is spent evaluating for collisions is time that can not be used for growing the tree.

During the early design stages of the Robust RRT algorithm, an analysis was performed to try to estimate the cost of the collision checking procedure. The planning algorithm was run on a scenario with obstacles, first with collision evaluation enabled for every trajectory, and later with collision evaluation completely disabled. The average number of samples in both cases was then measured and compared. For the case when collision evaluation was disabled, the number of samples generated was approximately 100 samples per second. On the other hand, when collision evaluation was active, the number of samples generated dropped to about 60-80 samples per second. This result corresponds to a 20-40% decrease in the number of samples per second that were generated. From this result, it was concluded that a method to reduce the number of collision checks was necessary.

To reduce the number of collision checks performed per iteration while still ensuring that any trajectory executed by the vehicle is feasible with respect to the latest sensory information a *lazy evaluation* strategy for collision evaluation was implemented. The Robust RRT algorithm re-evaluates the feasibility of a certain edge only if that edge is selected as part of the best path sequence to be sent to the controller. If the best trajectory is found infeasible, the infeasible portion of the tree is deleted and the next best sequence is selected for re-evaluation. Lazy evaluation enables the algorithm to focus mainly on growing the tree, while ensuring that the executed trajectory is still feasible according to the latest information about the environment.

One disadvantage of the lazy evaluation strategy is that the hazard penalty values stored in the tree may not be based on the latest perception data. As a result, on the rare occasions when unexpected obstacles appear in the environment, the selected trajectory might pass close to these new obstacles. It should be noted, however, that the feasibility of the trajectory is nonetheless guaranteed, as the trajectory is indeed checked for collisions and discarded if found infeasible. As long as the trajectory is feasible with the latest map, the planner will send it for execution to the controller,

and will discard it otherwise. As a result, the lazy evaluation strategy reduces the computation overhead without compromising the safety of the vehicle.

4.6 Efficient Expansion of the Tree

The main function to be performed at each step in the iteration is the expansion of the tree. The expansion is aimed at adding new nodes to the tree in such a way that the space covered by the tree increases as fast as possible.

4.6.1 Evaluation of the Distance to Samples

As a measure of distance, the Robust RRT algorithm uses the value of the minimum-length Dubins path between a node in the tree and the sample $\alpha(i)$. A full characterization of optimal Dubins paths is given in [22] and a further classification in [63]. We present some results on the analysis of the length of Dubins paths for a single vehicle from an initial configuration to a given point in the plane as developed in [23]. These results were used in the implementation of the planning algorithm.

Consider a nonholonomic vehicle i constrained to move along a path with bounded curvature, and let $1/\rho$ be the maximum curvature. Without loss of generality, we assume that the vehicle moves at constant speed $v = 1$. Based on this, let the configuration $g_i \in SE(2)$ of the vehicle be given in coordinates by $g_i = (x_i, y_i, \theta_i)$, where x_i and y_i are the projections of the vehicle's position into the local inertial reference frame, and θ_i is the orientation of the vehicle's longitudinal axis pointing forward with respect to the x-axis in the local frame. The dynamics of Dubins vehicle i are then described by the differential equations

$$\dot{x}_i = \cos(\theta_i), \tag{4.6a}$$

$$\dot{y}_i = \sin(\theta_i), \tag{4.6b}$$

$$\dot{\theta}_i = \omega_i, \quad \omega_i \in [-1/\rho, 1/\rho]. \tag{4.6c}$$

Note that for a given vehicle i and considering unit speed $v = 1$, Equations 4.6a and

4.6b are equivalent to Equations 4.1a and 4.1b, and Equation 4.6c is equivalent to Equation 4.1c for a circle of radius ρ. In other words, the Dubins vehicle model is a simplified but representative model for characterizing the vehicle motion.

Let $L_\rho(q) : \mathbb{R}^2 \to \mathbb{R}$ be the minimum length of the path satisfying Equation 4.6. This path steers a Dubins vehicle from the identity in $SE(2)$ to a point q in the plane without any constraints on the final heading, i.e., to any configuration in the set $\{q\} \times S^1 \subset SE(2)$. The characteristics of paths of minimal length with such boundary conditions have been studied in [67]. There it is proved that all such paths are a concatenation of and arc of a minimum-radius circle either in the positive or negative direction, with an arc of a minimum-radius circle in the opposite direction, or with a straight segment. The length of the circular arcs is upper bounded by $2\pi\rho$, and all subpaths are allowed to have zero length. In the Dubins formalism, such paths are called either of CC (circle, circle) or CL (circle, line) type, and include the subtypes C or L, arising when the length of either one of the two subpaths is zero.

It can be shown that the optimal paths are of type CC when q lies within the interior of one of the two circles of radius ρ that are tangent to the vehicle's direction of motion at the initial configuration, and are of type CL otherwise. In other words, having defined $\mathcal{D}_\rho^+ = \{q \in \mathbb{R}^2 : \|q - (0, \rho)\| < \rho\}$, $\mathcal{D}_\rho^- = \{q \in \mathbb{R}^2 : \|q - (0, -\rho)\| < \rho\}$, if $q \in \mathcal{D}_\rho^+ \cup \mathcal{D}_\rho^-$, the optimal path is of type CC, otherwise it is of type CL. Based on these results, the minimum length of a Dubins path can be quantified as follows [23].

Theorem 4.6.1 (Minimum-length Dubins path). *The minimum length $L_\rho(q)$ of a Dubins path steering a vehicle from $g_0 = (0, 0, 0) \in SE(2)$ to a point $q = (x, y) \in \mathbb{R}^2$ is given by:*

$$L_\rho = \begin{cases} \sqrt{d_c^2(q) - \rho^2} + \rho\left(\theta_c(q) - \arccos\dfrac{\rho}{d_c}\right) & q \notin \mathcal{D}_\rho^+ \cup \mathcal{D}_\rho^-, \\ \rho\left(2\pi - \phi(q) + \arcsin\dfrac{d_c(q)\sin(\theta_c(q))}{d_f(q)} + \arcsin\dfrac{\rho\sin(\phi(q))}{d_f(q)}\right) & \textit{otherwise,} \end{cases}$$

where $d_c(q) = \sqrt{x^2 + (|y| - \rho)^2}$ and $\theta_c(q) = \arctan(x, \rho - |y|)$ are polar coordinates of the point q with respect to the center of either $\mathcal{D}_\rho^+$ or $\mathcal{D}_\rho^-$, whichever is the closest,

$d_f(q) = \sqrt{x^2 + (|y| + \rho)^2}$ *is the distance of* q *from the other center, and*

$$\phi(q) = \arccos\left(\frac{5\rho^2 - d_f(q)^2}{4\rho^2}\right).$$

Note that in the implementation the range of the atan2 function must be set to be $[0, 2\pi)$ to give a valid distance. Theorem 4.6.1 is used in the computation of the distance between a node and the sample in the Robust RRT algorithm. This analytical calculation can be used to quickly evaluate all the nodes in the tree for a promising connection point.

4.6.2 Expanding the Tree

The easiest way to determine to which node the sample should be connected to is to follow a random ordering of the nodes in the tree. According to results found in [25], however, some performance improvements result if evaluation for potential collisions of the trajectory reaching the random sample $\alpha(i)$ is tested from tree nodes in a more efficient way. In the Robust RRT algorithm the heuristics used to add new nodes to the tree are an extension of the ideas presented in [25]. For more information on other efficient nearest-neighbor searching approaches, refer to the recent survey [36], and [2–5, 7, 14, 26, 37, 43, 58, 64, 75].

Before a feasible trajectory has been found, the tree is grown mainly according to an *exploration* heuristic, in which the emphasis of the algorithm is on adding new nodes to the tree that enlarge the space it reaches. To enhance exploration while keeping the number of collision checks to a minimum, the tree nodes are sorted in ascending order of distance to the sample $\alpha(i)$. In this way, physical feasibility is tested first from the nodes that are closer to the sample and therefore more likely to provide a collision-free trajectory. This results in a lower number of calls to the collision-checking procedure and in general reduces computation time more than the increase produced by the sorting of the nodes and evaluation of the distances. Note that in this exploration heuristics potentially every node in the tree could be tried during the expansion step. In the actual implementation of the algorithm the maximum number

of nodes to try to expand from was limited to 10 to bound the computation time.

Once a feasible trajectory has been found, the tree is instead grown primarily according to an *optimization* heuristic, in which the focus of the search shifts from exploration to the optimization of the computed trajectory. To make the new trajectories progressively approach the shortest path, the nodes are now sorted in ascending order of total cost to reach the sample $\alpha(i)$. This total cost consists of the accumulated cost up to the particular node v plus the cost-to-go from v to $\alpha(i)$,

$$c_{total} = c_{cum}(v) + L_\rho(\alpha(i)), \tag{4.7}$$

where $c_{cum}(v)$ represents the cumulative cost from the root of the tree to the particular node v. To enhance the quality of the computed solution, the tree nodes are sorted in ascending order of total cost to the sample $\alpha(i)$ and as a result physical feasibility is tested first from the nodes that provide the shortest trajectory to the sample and therefore are closest to the shortest path. This is at the cost of the computation time required to sort the nodes and evaluating the Dubins path length.

Employing one or the other expansion heuristics in a binary way based on having reached the goal is not the best strategy. Even before having a trajectory that reaches the goal and consequently when the focus should be on exploration, using the optimization heuristics to reduce wavy trajectories is beneficial. Similarly, once a feasible trajectory to the goal has been found and therefore the focus should be on refining the available solution, some exploration of the environment is beneficial for example in case an unexpected obstacle blocks the area around the feasible solution, or for having a richer tree when the goal switches to the next. Based on this reasoning, each expansion heuristic is used mostly in the particular case they were conceived for, but the other heuristic is also used with a small probability. When implementing the algorithm, the ratio of exploration vs. optimization heuristic used was 70% vs. 30% before a trajectory to the target was found, and 30% vs. 70% once it was found.

4.6.3 Estimates of the Cost-To-Go

There are two estimates of the cost-to-go at each node in the tree, a *lower bound* on the cost-to-go and an *upper bound* on the cost-to-go.

As a measure of the lower bound we use the Euclidean distance between the position of the node and the position of the goal. Then, if the position of the node is given by $V = (v_x, v_y)$ and that of the goal by $G = (g_x, g_y)$, the lower bound of the cost-to-go at the node v becomes

$$c_{lb} = \sqrt{(v_x - g_x)^2 + (v_y - g_y)^2}. \tag{4.8}$$

The Euclidean distance is a lower bound since the shortest distance between any two points (x_1, y_1), $(x_2, y_2) \in \mathbb{R}^2$ is given by a straight line.

On the other hand, we use a more complicated metric as the upper bound on the cost-to-go. Assume that a trajectory from v to the goal g exists and let j be the number of nodes along that trajectory, including g but not v. The upper bound of the cost-to-go at the node v is then given by the sum for every node j of the sum of the length and the hazard value of the edge associated with each node j. When no trajectory from v to g exists, then the upper bound on the cost-to-go for that node is set to infinity. Mathematically,

$$c_{ub} = \begin{cases} \sum_j l_j + h_j, \quad j \in (v, g], & \text{if goal reachable,} \\ \infty, & \text{if goal not reachable,} \end{cases} \tag{4.9}$$

where l_j is the length of the edge associated with node j and h_j is the hazard value given by Equation 4.5. For nodes that are considered as reaching the goal, their upper bound cost-to-go is set equal to their lower bound. Avoiding to give a zero cost to every node having reached the goal has the advantage of still being able to differentiate between those nodes exactly over the goal and those on the boundary of the threshold defining when the goal is considered reached. In this way, for paths with similar lengths, those ending right over the goal are prefered.

Each time a sample $\alpha(i)$ is added to the tree, a trajectory to the goal is generated and evaluated for collision with obstacles. If the trajectory results to be collision-free, then we have found a path that guides the vehicle from the initial condition to the final goal point along the feasible space $\mathcal{C}_{free}$, therefore solving the motion planning problem. However, we are interested in finding a very efficient trajectory satisfying the feasibility problem and not just any trajectory.

For this reason, every time a new feasible solution is found, the estimates of the upper bound on the cost-to-go of the nodes in the path from the newly added node to the root of the tree are updated. To update these costs we traverse the tree backward toward the root. At each node we compare the old upper bound on the cost-to-go with the new cost resulting from the newly found trajectory. If the new cost is lower than the old one, we update the upper bound and continue the process from the parent. Otherwise, the process stops, indicating that there exists another subtree from the parent node that presents a lower upper bound on the cost-to-go.

The upper bound on the cost-to-go is the cost used to choose the shortest trajectory that reaches the goal after a feasible trajectory has been found. Before a feasible trajectory is found, the lower bound is used, and the search for the best trajectory follows an essentially greedy approach. Among the trajectories that have terminal nodes within a small distance from the terminal node that is the closest in terms of lower bound to the goal, the trajectory with the smallest penalty due to obstacles and deviation from the lane center is chosen as the best alternative. This approach maximizes exploration while still keeping the vehicle safe from collisions.

4.7 Construction of a Safe and Rich Tree

4.7.1 Improving the Richness of the Tree

Consider the tree at some point in time. The tree is composed of two different types of nodes, *stopping nodes* and *intermediate nodes*, which are defined as follows.

Definition 4.7.1 (Stopping node). A *stopping node* is defined as any node where a

given trajectory ends and where the state corresponds to zero speed or the vehicle being completely stopped.

Definition 4.7.2 (Intermediate node)**.** An *intermediate node* is defined as any node along a given trajectory and where the state corresponds to the vehicle being in motion.

Given these definitions, we can now proceed to discuss what happens when a sample is added to the tree.

Suppose that a new sample has just been added to the tree as a node. Each node that was formerly a sample by construction brings the vehicle to a complete stop. In this way, the tree has numerous nodes that ensure that the vehicle will eventually reach a safe state where it is completely stopped, as opposed to making the car follow a path where it is not known when and if the vehicle will be able to eventually come to a complete and safe stop. Stopping nodes, however, are not considered as potential connection points for the samples. This is because it would be very inefficient if the vehicle stopped periodically along a given trajectory when it is not required to do so. As a result, we need to generate other nodes in the tree to which samples could be connected to.

The trajectory spanning between the newly added stopping node and its parent can be split into $n > 1$ segments where the breakpoints are intermediate nodes and the endpoint is the stopping node. Because the vehicle is in motion along the trajectory, the intermediate nodes are the ones made available as potential connection points to the samples during future tree expansion steps. In the implementation of the RRT Algorithm for the DUC the number n of intermediate nodes was fixed and set to $n = 4$ for every trajectory (unless it was too short, in which case the number of intermediate nodes added was less or zero).

In the RRT approach presented in [25], if the propagated trajectory was determined to be colliding with an obstacle, then the entire trajectory was discarded. On the other hand, in the Robust RRT algorithm, if the last portion of a propagated trajectory is found infeasible because of collision with an obstacle, then the feasible

portion of the trajectory is still added to the tree. The point of the trajectory before collision with the obstacle will be added to the tree as a node and called an *unsafe node*, defined next.

Definition 4.7.3 (Unsafe node)**.** An *unsafe node* is defined as any node where a given trajectory ends but where the state of the vehicle at that node does not correspond to zero speed and consequently the vehicle is in motion.

By keeping the feasible portion of an overall infeasible trajectory, this method avoids wasting the computation time spent in sampling the space, propagating, and evaluating for collisions, and contributes to making the tree richer as well since the feasible portion of the path up to the obstacle is kept. The unsafe node might be later marked as a normal safe node if a safe trajectory ending in a stopped vehicle configuration is later connected to the node. When selecting the best trajectory to send to the controller (refer to line 24 in Algorithm **??**), the unsafe nodes are excluded. Therefore, although unsafe nodes contribute to building a rich tree that explores the space as much as possible, the safety of the vehicle is never compromised.

4.7.2 Safety as an Invariant Property

Ensuring the safety of the vehicle is a key feature of the Robust RRT algorithm, given that it was designed to plan in dynamic and uncertain environments. Before proceeding any further, we need to define a *safe state.*

Definition 4.7.4 (Safe state)**.** A given state is defined as a *safe state* if the vehicle can remain in that state for an indefinite period of time while avoiding collisions with obstacles and without violating the rules of the road.

In this definition, both static and moving obstacles are considered, where the latter are assumed to maintain their paths at the instance in time the given vehicle state was declared to be safe. In particular, safe states are such that the vehicle is stopped outside a non-restricted region as defined in Section 4.5.1 (e.g., outside of intersections). More general notions of safe states are available, for example, in [25,

61]. Having given the definition of a safe state, we can now state that of a *safe trajectory.*

Definition 4.7.5 (Safe trajectory)**.** A trajectory is said to be *safe* if it terminates in a safe state.

In the Robust RRT algorithm only predicted trajectories that are safe can be selected for execution. In other words, the ability to eventually come to a safe stop is maintained as an invariant, by requiring that any trajectory that is executed brings the vehicle to a complete stop. The existence of the stopping nodes guarantees that there is always a feasible way to come to a safe stop when the car is moving. In other words, unless there is a safe stopping node at the end of the chosen trajectory, the vehicle will not start executing it.

It should be noted that this safety guarantee applies if and only if there is no unexpected change in the surrounding environment. In the eventual case that this happens and all previously safe paths become infeasible, the motion planner will search for a new feasible path. If none is found, then the planner will command an emergency braking maneuver to the controller in order to bring the car to a stop as fast as possible.

4.7.3 Safe Maximum Speed and Lateral Acceleration

The speed limit as determined by a speed sign might be dangerous if the road presents tight curves or if its configuration cannot be properly sensed for any reason. As a result, the determination of the maximum speed at which the vehicle should travel along a given road section follows a special procedure with the objective of ensuring safety at all times. The maximum speed that the vehicle can reach at a given point along its trajectory is depends on the maximum speed v_{max} that is assigned to the space sample $\alpha(i)$ used in the propagation. This v_{max} is determined as the minimum of three quantities:

1. The speed limit from corresponding segment of the original Route Network Definition File (RNDF),

2. The speed limit that arises from a bound on lateral acceleration, and

3. The speed limit that arises from a bounded deceleration constraint.

The lateral acceleration constraint is based on an estimate of the curvature of the road and forces the vehicle to slow down appropriately in turns. To estimate this curvature easily, we consider the radius r of the unique circle that passes through the current vehicle location and the goal and is tangential at the vehicle location to the vehicle direction (heading). We bound the lateral acceleration of the vehicle to $a_c \approx 0.5\,\mathrm{m/s^2}$ and not less to give the vehicle enough agility to be able to avoid obstacles in curved segments of the course. With these values known, the maximum safe speed based on the lateral acceleration constraint that might be assigned to any sample $\alpha(i)$ is

$$v_{max} = \sqrt{a_c r}, \tag{4.10}$$

where $a_c \approx 0.5\,\mathrm{m/s^2}$ is the maximum lateral acceleration allowed and r is the radius of the circle.

The bounded longitudinal deceleration constraint is set to $6\,\mathrm{m/s^2}$. As a result of this limit, for the particular implementation of the algorithm in the DUC, the maximum safe speed was capped to 25 mph. The purpose of this limit was to enhance the vehicle safety by reducing the potential braking distance in case an unexpected obstacle or perception problem arose at high speed. At 30 mph with a deceleration of $6\,\mathrm{m/s^2}$ the braking distance is 15.0 m, whereas if the speed of the vehicle is reduced to 25 mph, that distance reduces to 10.4 m. This is a decrease greater than 30 %. Based on this result, it was decided that the corresponding increase in safety in case of poor sensing information outdid the resulting increase in travel time, and the maximum speed that the vehicle could reach during the race was set to 25 mph.

The maximum safe speed based on the lateral acceleration given to each sample does not guarantee that the vehicle will never exceed that value. The result of the calculation performed is not an upper bound, and moreover the road can certainly have tight turns between the current location of the vehicle and the goal, particularly if they are far from each other. The objective of limiting the speed at the sampling

level is to avoid having speed values that are much higher than those achievable by the vehicle, but not to guarantee safety. As a result, an extra measure is taken at the trajectory propagation level to guarantee that the lateral acceleration of the vehicle does never result in rollover.

During the simulation of the vehicle motion, if the lateral acceleration exceeds a pre-specified limit set to $a_{c_{max}} = 4\,\mathrm{m/s^2}$, the propagation is stopped and returns infeasibility. In this case, repropagation occurs using the same controller reference path but lowering the speed limit to 6% of the original value and increasing the maximum lateral acceleration limit to $a_{c_{max}} = 4.7\mathrm{m/s^2}$. The maximum lateral acceleration value is set lower during the regular propagation step in order to decrease the probability that the limit could be exceeded again during the repropagation step. If the lateral acceleration still exceeds its limit during repropagation, then that sample is discarded. In this way, the vehicle is guaranteed to never execute a trajectory that could exceed the specified maximum lateral acceleration.

Chapter 5

Robust RRT Evaluation

This chapter discusses the effectiveness of the Robust RRT algorithm based on the analysis of both simulation and real data collected during the 2007 DARPA Urban Challenge. Robust RRT was implemented as the motion planner for Talos, the robotic Land Rover LR3 that was MIT's entry in the Challenge [52].

The planning and control system architecture of Talos is presented in Figure 5-1. The motion planner computes a path to reach a goal location, which is specified by a higher-level route planner (the Navigator). This path must avoid collisions with static obstacles and other vehicles, as well as abide by the rules of the road. As with any sampling-based motion planning approach, collision detection and evaluation is performed externally to the planning algorithm (the Drivability Map). The output of the motion planner is in turn fed to a low-level controller, which interfaces directly to the vehicle. The low-level controller is responsible for the execution of the motion plan.

It is interesting to note that Talos used a single motion planner for the entire race. To the author's best knowledge, MIT was the only participating team in the DUC that used a single planner for all the different driving scenarios encountered during the race. Being able to use a single planner given the nontrivial differences in the characteristics of the varied scenarios encountered during urban driving shows the flexibility and the extensibility of the Robust RRT motion planning algorithm.

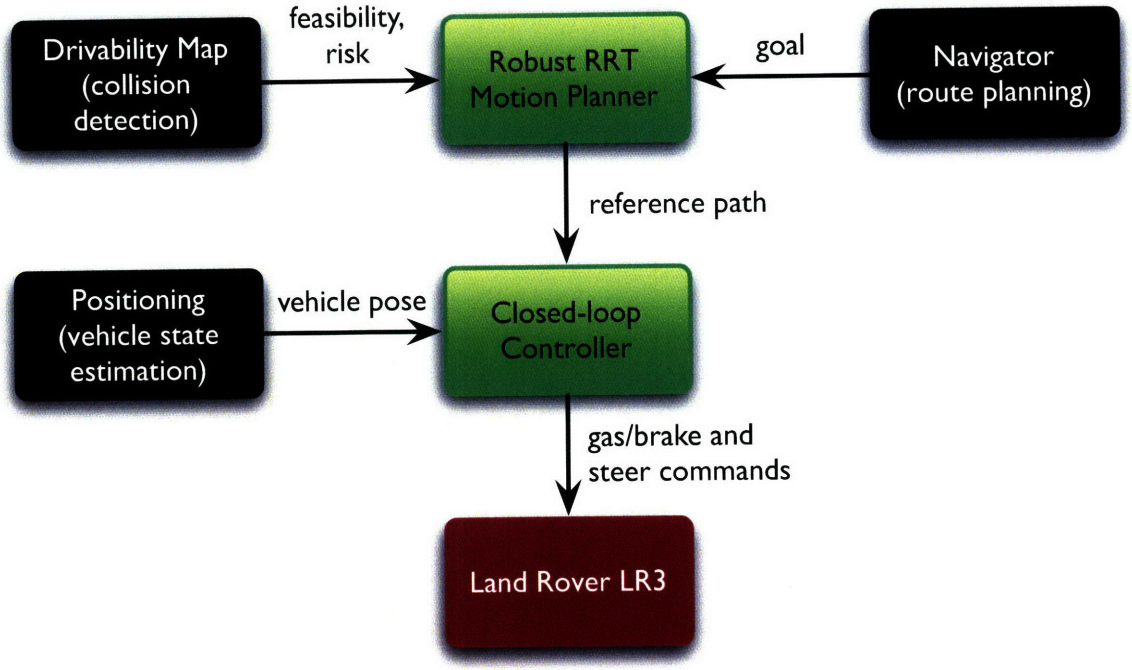


Figure 5-1: Planning and control system architecture.

5.1 Simulation Data

The robust RRT algorithm has been tested extensively in simulation early on in the program and then on the vehicle itself. This enabled the team to achieve a level of code stability before the algorithm was tested on the car, thus cutting testing time and costs as well as increasing safety both for the operators and the testing platform. After the Urban Challenge competition, we used simulation to evaluate the performance of the algorithm on more extreme cases of environmental structure than those encountered during the race. The next section presents the simulations that were performed and discusses the performance of the Robust RRT algorithm for each scenario.

5.1.1 Simulated Scenarios

There are six different test scenarios that were analyzed: full parking lot, blocked road, rectangular track with obstacles, dense obstacle field, narrow passage, and dead end. The full parking lot, blocked road, and rectangular track scenarios evaluate the

peformance of the planner in a urban setting and its ability to plan in an obstacle field, to perform U-turns, and to plan along regular roads. On the other hand, the dense obstacle field, narrow passage, and dead end scenarios deviate from those encountered in a urban setting. These scenarios have the purpose of evaluating the flexibility/performance of the algorithm (essentially unmodified from the race) in planning over non-urban environments. Any modification made to the algorithm had the goal of facilitating the analysis of the particular algorithm property being evaluated and will be described along with the description of the particular test that was conducted.

The properties of the goal are common among all the test scenarios presented in this chapter. The goal is represented by a green arrow over a dark green rectangle. The arrow indicates the direction that the car needs to have when it reaches the goal location. The goal has a zero speed, and as a result the car will come to a complete stop when reaching the goal. The car is considered to have reached the goal when the vehicle has covered at least half of the area of the rectangle determining the goal. A hard constraint of 30° is used as the tolerance for the direction of the car at the goal. In other words, any path that arrives at the goal with a direction of $\pm 30°$ with respect to the direction of the goal is considered as reaching the goal.

The visualization of the tree is also common among all test cases. A given tree might display seven different colors: orange, purple, yellow, green, light brown, red, and cyan. Orange edges correspond to the tree of reference paths for the controller, purple edges correspond to paths that reach the target, yellow edges to paths that within 10% of the cost of the best available path, green indicates the predicted trajectory of the car, light brown represents paths that are safe but that do not reach the target, red represents unsafe paths, and finally cyan indicates the paths that are to be traveled in reverse.

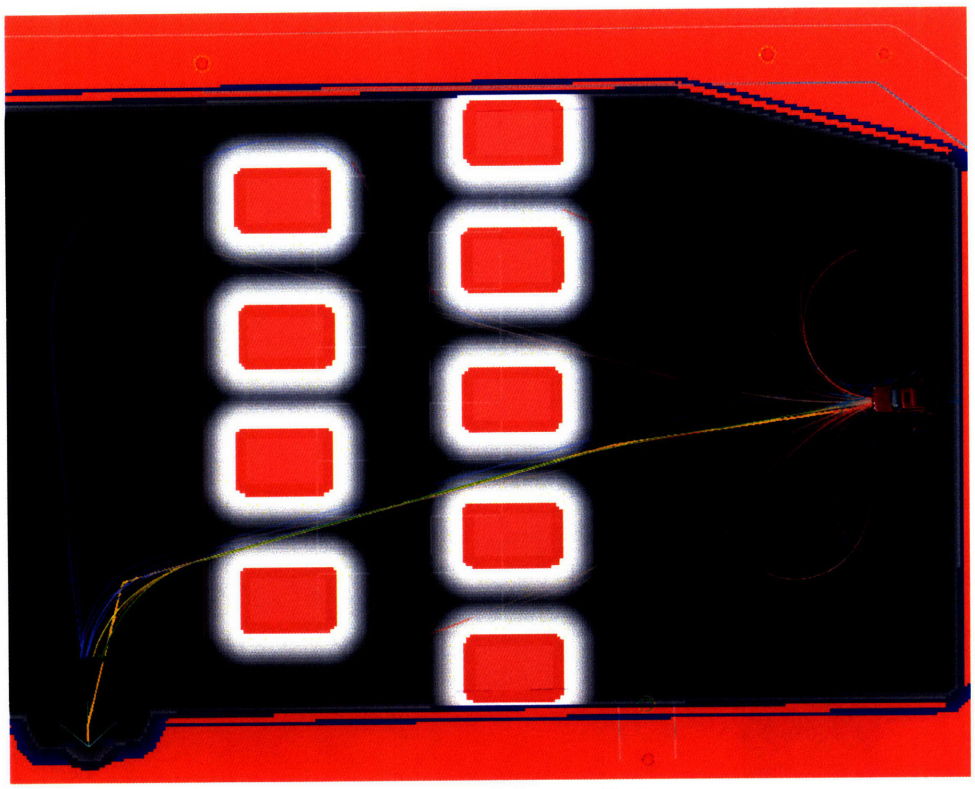
The implementation of the Robust RRT algorithm used in Talos is capable of performing advanced driving maneuvers as specified by the DUC requirements. These advanced maneuvers include navigating and parking in a potentially full parking lot and performing a U-turn maneuver in case the lane of travel were blocked. It is

important to note that the same motion planning algorithm is used even for the generation of this special set of maneuvers.

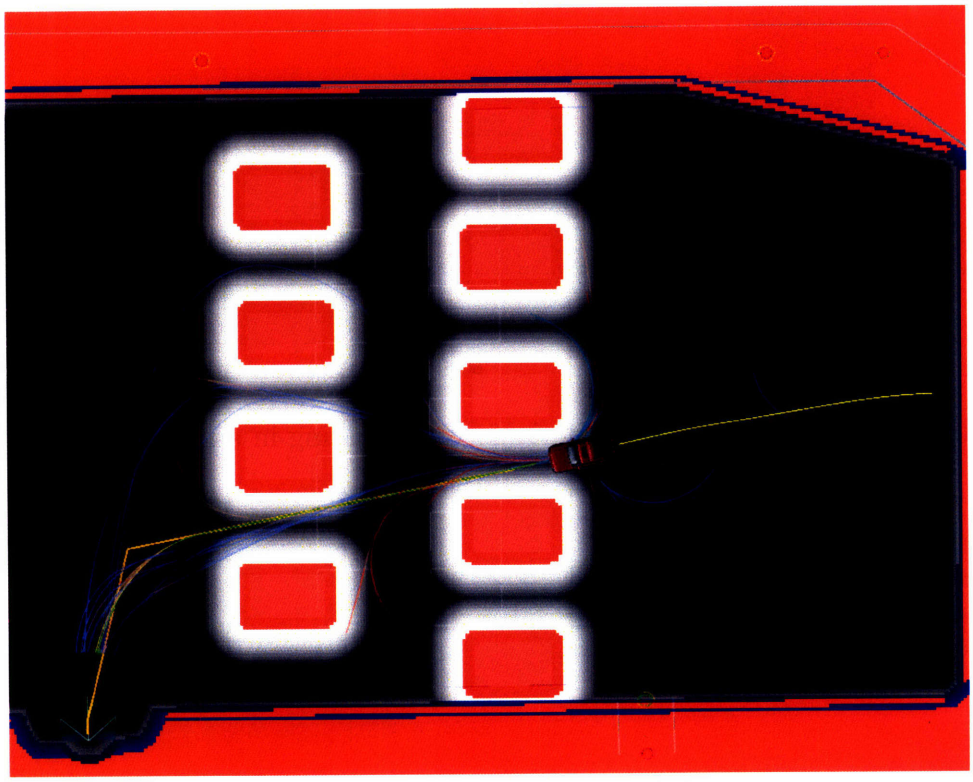
Scenario 1: Full Parking Lot

Figure 5-2 shows Talos driving through one of the parking lots of the UCE course with several obstacles simulating parked cars. Each of the obstacles is placed over a parking space and occupies it completely, and therefore measures approximately 3 m wide by 5 m long. Talos' initial condition is next to the right boundary of the zone with a direction of 180° with respect to the horizontal. At the beginning of the simulation, as shown in Figure 5-2a, several paths successfully reach the goal while still other paths explore the space around and between obstacles. As Talos progresses through the parking lot, as can be seen in Figure 5-2b, the path that the car follows is refined. There are also paths that reach the goal by going to the right of the obstacles, although these paths are not selected as best because they are longer.

Figure 5-3 shows the distribution in the length of the paths for the parking scenario. The number of trials was 20. The length of the obstacle-free minimum-length Dubins path between the car initial location and the location of the goal for this scenario is 55.2197 m. The figure shows that in 85% of the trials the path length was between 55-60 m and therefore no more than 8.7% longer than the obstacle-free path. These results indicate that the Robust RRT algorithm produces paths that are efficient in length and have small variability range.



(a) Beginning of the simulation.



(b) Approaching the first row of parked cars.

Figure 5-2: Robust RRT algorithm planning in a full parking lot. Figures show the evolution of the tree as Talos drives towards the parking lot exit.

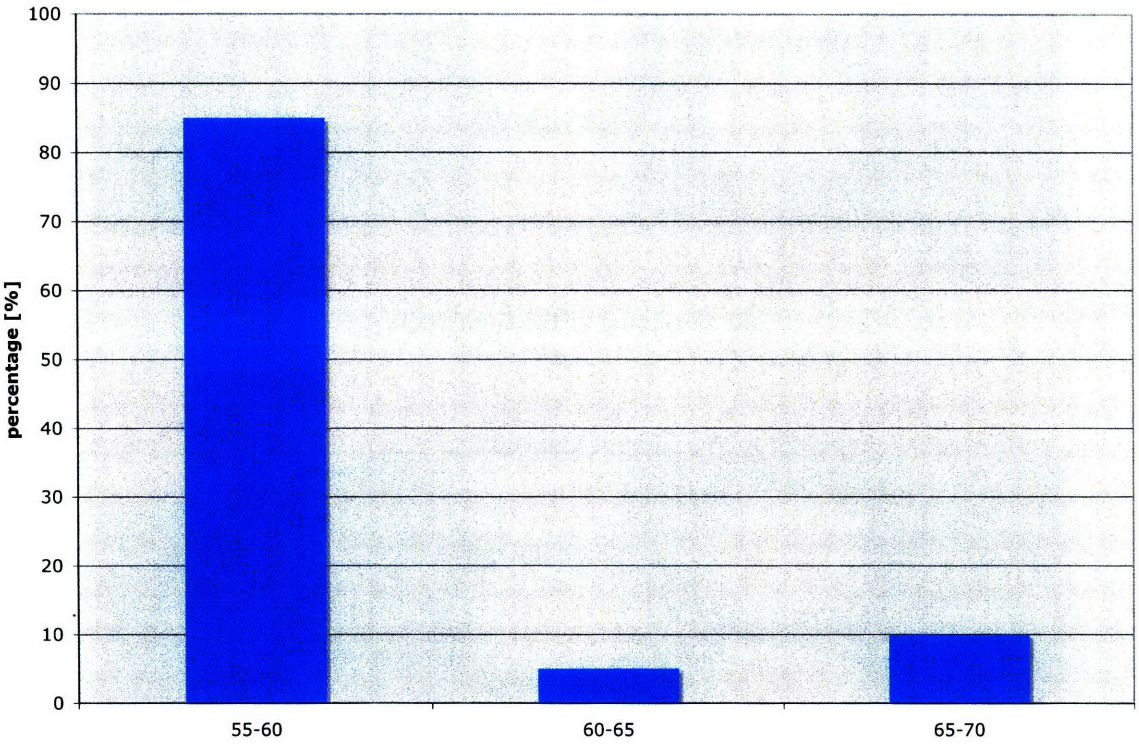
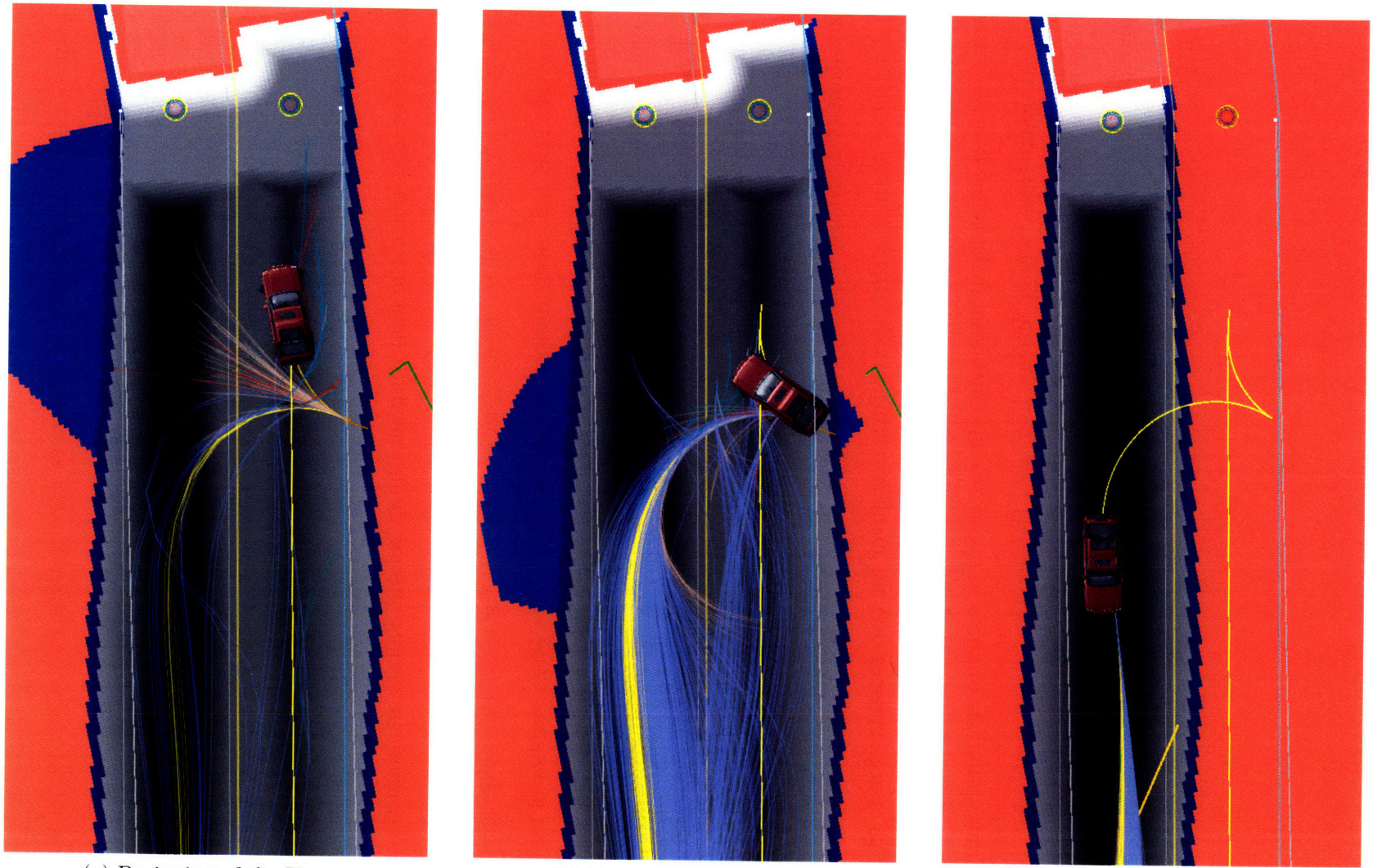


Figure 5-3: Path length distribution while planning in a full parking lot.

Scenario 2: Blocked Road

Figure 5-4 shows a sequence of instances corresponding to Talos performing a U-turn maneuver. The road where the maneuver takes place corresponds to one of the roads in the UCE course and is 10 m wide. The road is two lanes wide, but Talos is restricted from proceeding forward by the obstacles in both lanes in front of the vehicle (red rectangles in the road). In Figure 5-4a, Talos reverses away from the obstacles blocking the road and tries to move as close as possible to the right boundary of the road. As can be seen, even at this early stage, there are several trajectories that reach the goal (opposite lane, behind Talos), although the tree has still not grown very rich. In Figure 5-4b, Talos steers in the left direction as much as possible and starts to move forward to perform a U-turn. By this time, the tree has grown very rich and covers almost the whole road, with a large number of trajectories reaching the goal. In Figure 5-4c, Talos is already driving along the opposite lane of travel.

Notice the elegance of the path followed by the car during the execution of the U-turn maneuver. The particular width of this road allowed the planning of a U-turn in just two maneuvers. When the dimensions of the road are such that they allow the execution of a U-turn in just two segments, the two-segment maneuver is preferred and selected against a three-point turn maneuver for two reasons. First, the U-turn consisting of just two segments is shorter and therefore faster to execute. Second, during a three-point turn maneuver, the car needs to come to stop and change gears an additional time, which further increases the length of time spent executing the three-point turn. Therefore, the U-turn consisting of just two segments is faster and therefore preferred when the dimensions of the road allow it.



(a) Beginning of the U-turn.

(b) Starting the forward leg.

(c) Finishing the U-turn.

Figure 5-4: Robust RRT algorithm planning a U-turn maneuver.

Scenario 3: Rectangular Track with Obstacles

The Robust RRT algorithm yielded very good results for driving around a circuit where in addition to lane following, several passing maneuvers and a U-turn were also involved. Figure 5-5 shows the trace of the path taken by the vehicle. The test consisted in two clockwise and two counter-clockwise loops around a rounded rectangle. Three stationary cars were distributed along the course on both directions. One of these cars was parked over a very tight turn (bottom left), requiring a difficult passing maneuver from the passing vehicle. Even so, the paths indicate that Talos was able to execute the maneuver repeatably. In order to switch from the clockwise to the counter-clockwise looping direction, a U-turn was needed at one of the stub roads.

Figure 5-5 shows that as the vehicle repeated a given loop in the course, both paths are very close to each other. They are not identical as the paths are efficient but not optimal, even though arguably a human driver would probably not produce paths that are as similar to each other, particularly in passing when there is no lane center for the driver to follow. Notice how the turns are not very rounded, and nonetheless the paths described by the vehicle feature very smooth turns. This is possible thanks to the availability of a rich tree that covers most of the lane without following the piecewise linear characteristics of the road. When passing obstacles, the motion-planning algorithm produces paths that start the passing maneuver and come back to the lane at safe distances from the car being passed. Even in the case of the passing maneuver along the turn, the paths taken do not come close to the obstacle. Having no lane constraints at the end of the stub road, the planning algorithm easily finds a simple U-turn maneuver that positions the vehicle in the opposite lane for it to move in a counter-clockwise direction.

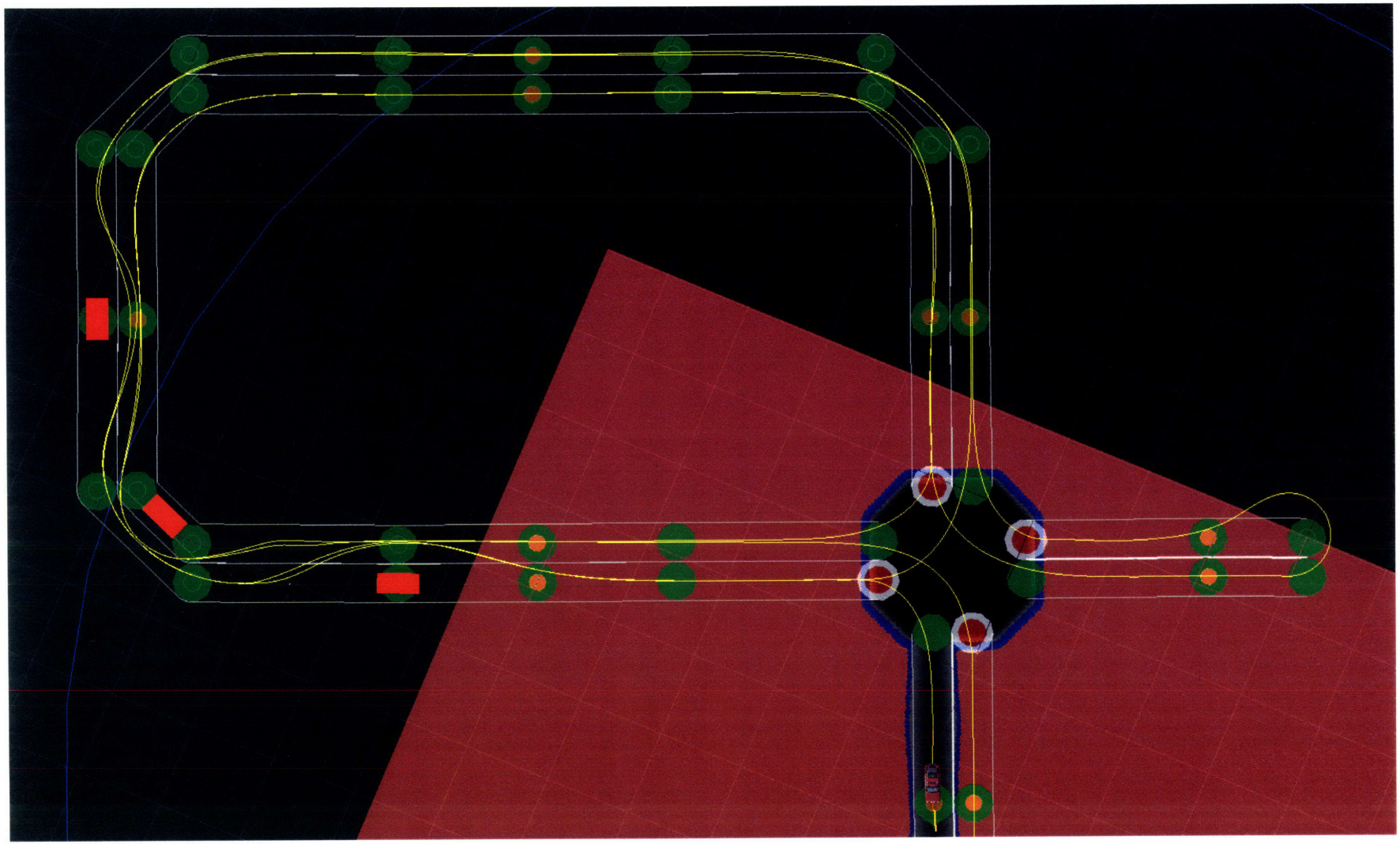


Figure 5-5: Talos went around a rectangular track four times, passing stationary vehicles and performing a U-turn.

Scenario 4: Dense Obstacle Field

The next set of tests were designed to analyze the length of the paths generated by the algorithm in a very dense obstacle field. Given that Robust RRT is a sampling-based motion planning algorithm that does not run any optimization routines, it is of great interest to determine whether the paths generated are of efficient length or not. The tests were run in the scenario shown in Figure 5-6. Each obstacle in the field has a size representative of that of a big car and is 2.5 m wide and 5 m long. The location of the goal is the same for all tests, whereas the initial condition of Talos will vary among tests and will be described as each particular test is presented. The tests analyze the length of paths for different initial conditions of the vehicle, where the initial orientation and location of the vehicle in the scenario are varied. In addition, a sensitivity analysis of the hazard weight factor described in Section 4.5.2 is also performed and recommendations for the value of the weight are provided.

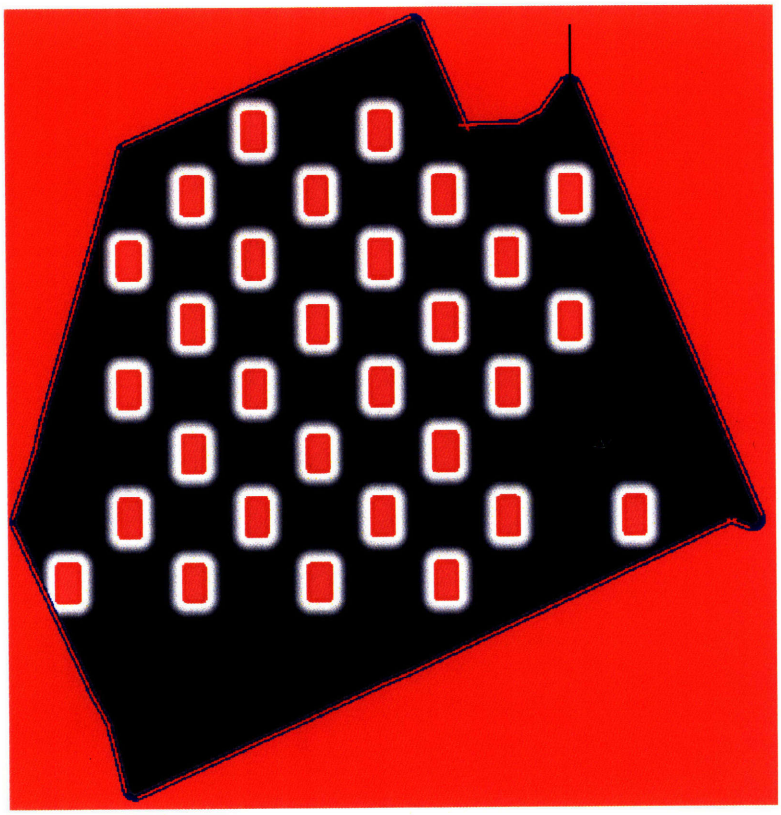


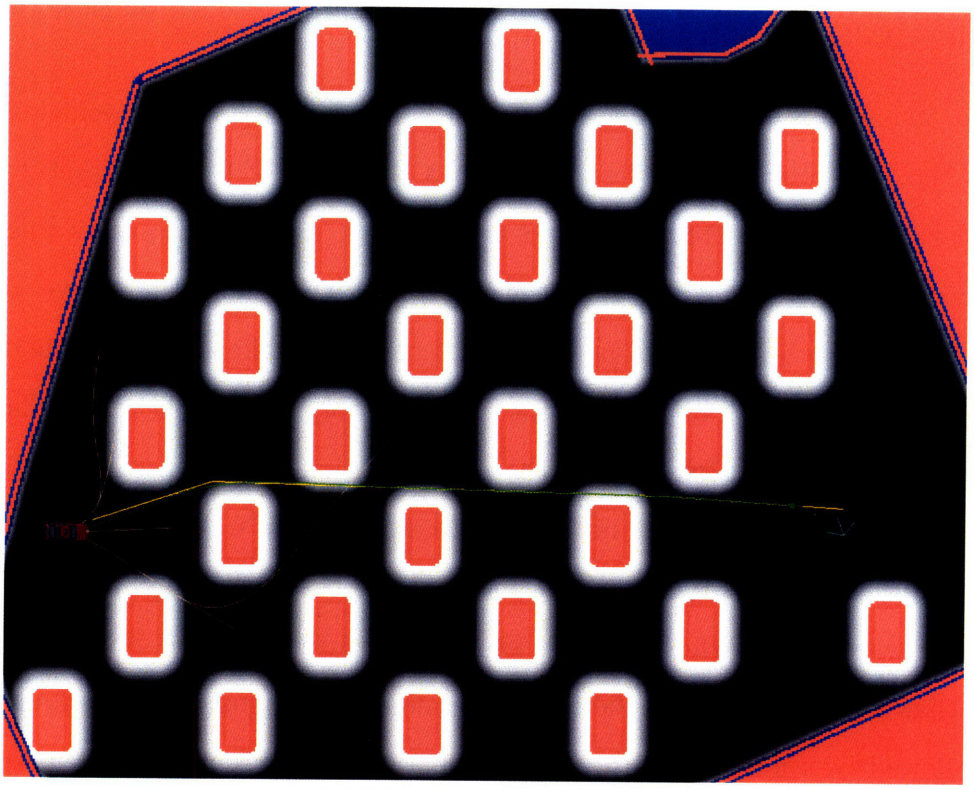
Figure 5-6: The obstacle avoidance scenario used to evaluate the length of paths for different initial conditions.

Before presenting each particular test, it is convenient to highlight important properties of the paths generated by the Robust RRT algorithm particularly as compared to the standard RRT algorithm. The particular findings corresponding to a given test in the set will be explained when that test is presented. In contrast to the disadvantages commonly attributed to RRT-based algorithms [21], the paths generated by Robust RRT are smooth and not squiggly, meaning that the paths do not bear excessive waviness throughout their length. This desirable behavior of the generated paths results from the closed-loop approach used in the generation of paths as well as the efficient heuristics used when connecting samples to the tree. Nonetheless, the Robust RRT algorithm does not loose the rapid exploration capabilities that is one of the main qualities of RRT. In the tests that follow, it can be seen that the tree covers most of the free space early in the planning process and reaches to far ends of the field.

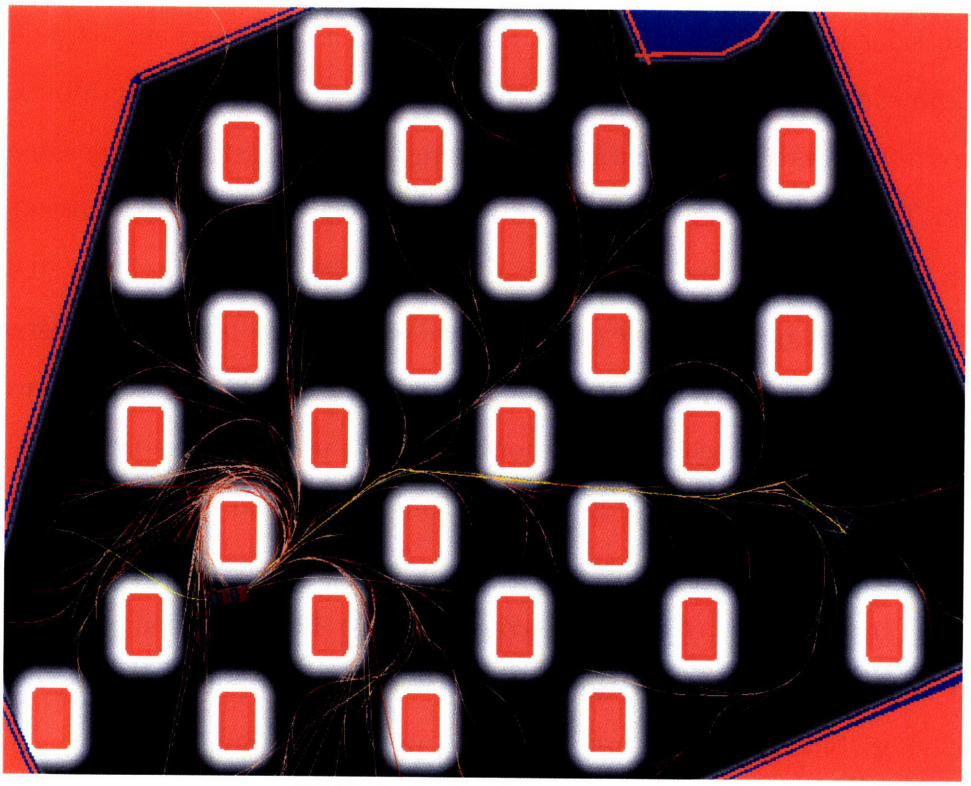
Scenario 4.1: Dense Obstacle Field, 0° Initial Orientation

The first test in this set corresponds to an initial condition with the vehicle located next to the left boundary of the zone with a direction of 0° with respect to the horizontal and is represented in Figures 5-7 and 5-8. Figure 5-7a shows this initial condition. Figures 5-7b and 5-8a show the evolution of the tree as Talos drives towards the goal. In Figure 5-7b, several paths come very close to the goal but none of them reach it because they do not satisfy the direction threshold constraint. Then, the path chosen as best and followed by the vehicle is that which comes closest to the goal but at the same time does not come excessively close to obstacles. Given this best estimate of how to proceed, Talos moves along the path diagonally to the left. About half way between the snapshots shown in Figures 5-7b and 5-8a, a path that reaches that goal by exploiting the free space along the diagonals is found and starts being followed by the vehicle. Figure 5-8a shows numerous paths that successfully reach the goal and will later be refined as the vehicle gets closer to the goal. At one point Talos turns right and reaches the goal by driving essentially straight to it. Figure 5-8b shows the entire path followed by Talos, which had a length of 95.44 m.

Figure 5-9 shows the distribution in the lengths of the paths for the same initial condition. The number of trials was set to 100. The length of the minimum-length Dubins path for this initial condition and scenario but **without obstacles** is 80.9028 m. This value would suggest that the optimal path length in the presence of obstacle will be around 85 m, since given the location of the obstacle the vehicle must drive around some of them to avoid a collision. According to Figure 5-9, more than 70% of the paths have a length of 110 m or less, or equivalently, are no more than 30% longer than the optimal path. Overall, the results show that the paths have a length that is efficient. In particular, one should consider that the paths that are selected as best and followed by the vehicle are not the shortest paths found but those that are relatively short but also safe by maintaining a prudent distance from obstacles.

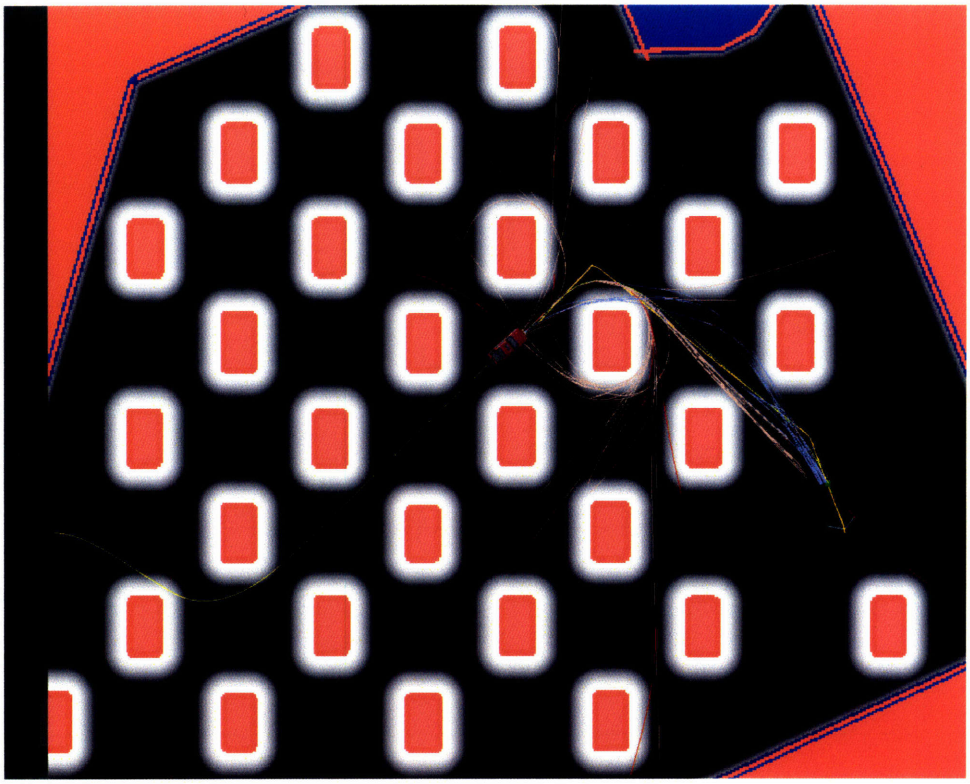


(a) Initial condition of Talos.

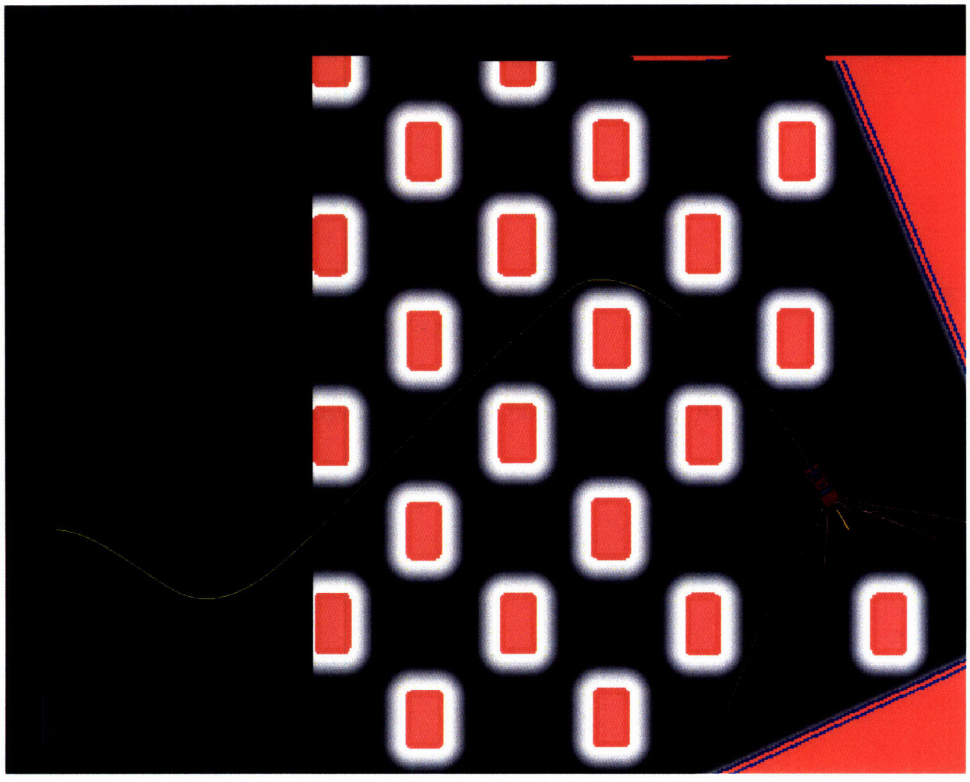


(b) Early in the planning process

Figure 5-7: Planning through a dense obstacle field with an initial condition corresponding to far left and 0°.



(a) Final stages of the planning process.



(b) Talos having reached the goal.

Figure 5-8: Planning through a dense obstacle field with an initial condition corresponding to far left and 0°.

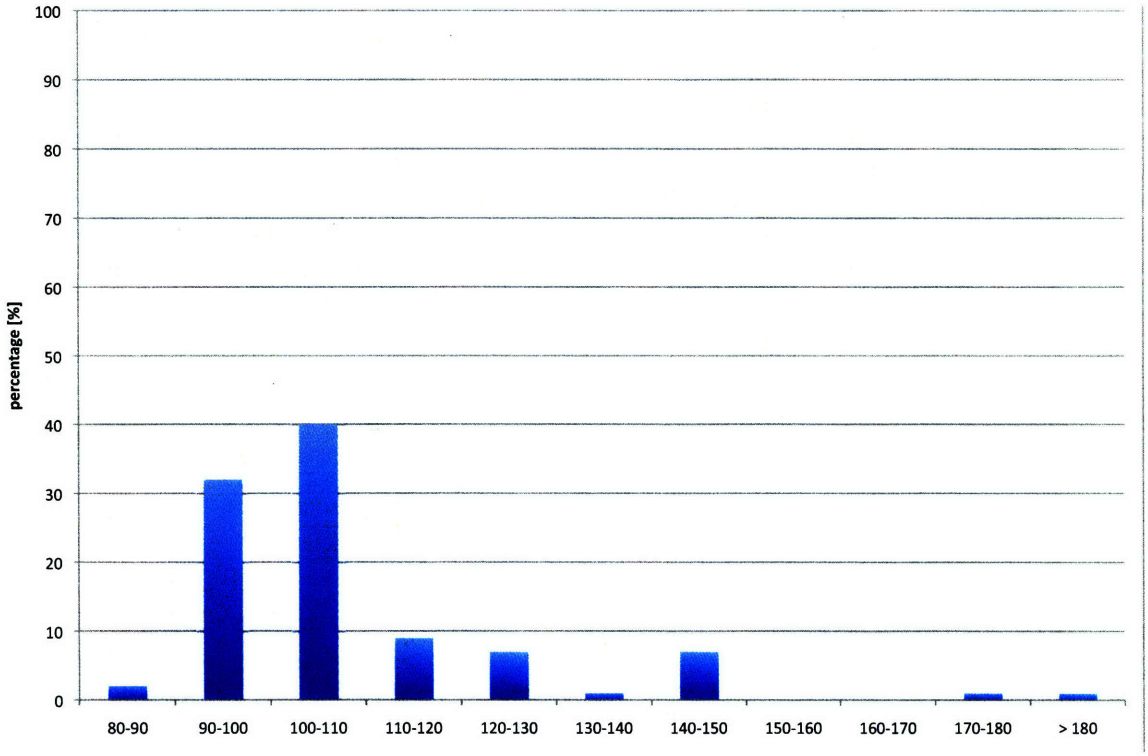
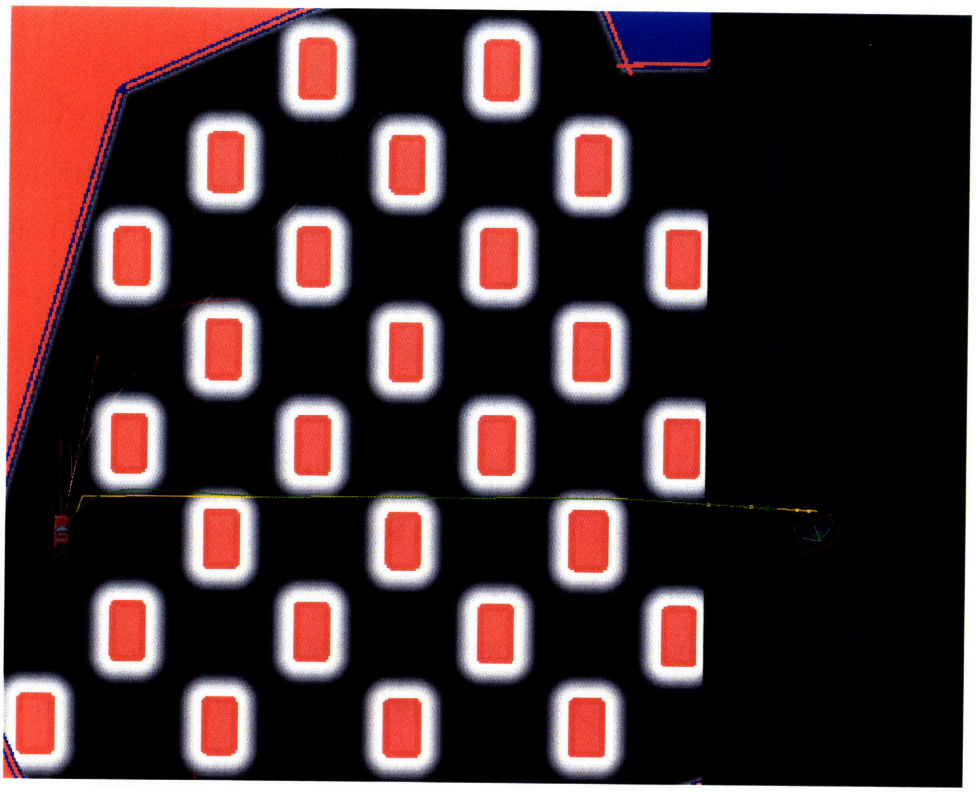


Figure 5-9: Distribution of path lengths for the initial conditions corresponding to far left and 0°.

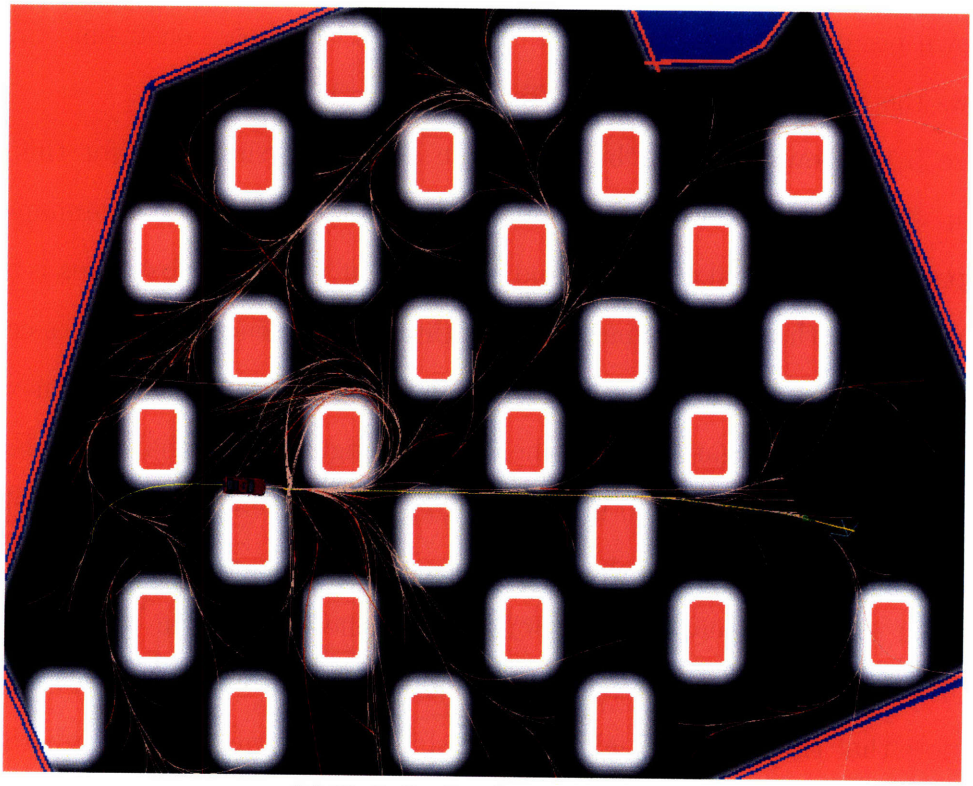
Scenario 4.2: Dense Obstacle Field, 90° Initial Orientation

The second test in this set corresponds to an initial condition with the vehicle in the same location as before but with a direction of 90° with respect to the horizontal and is represented in Figures 5-10 and 5-11. Figure 5-10a shows this initial condition. Figures 5-10b and 5-11a show the evolution of the tree as Talos drives towards the goal. In Figure 5-10b, some paths come very close to the goal but they fail to reach it because they do not satisfy the direction threshold constraint. Note that, as a result, the tree is very expansive as the algorithm explores almost the entire zone looking for a feasible solution. Even without a feasible path to the goal it is apparent that Talos is heading towards the goal along a reasonable path. In Figure 5-11a, now further into the planning process, several paths now successfully reach the goal. A path going initially diagonally up and towards the goal and later diagonally straight to the goal was found and therefore this path was selected as the best available route. Therefore, even though the car was originally heading straight to the goal, it ended following the path shown because that path satisfied the direction threshold constraint. Figure 5-11b shows the entire path followed by Talos. Its length is 102.53 m.

Figure 5-12 shows the distribution in the lengths of the paths for this initial condition. The number of trials was set to 100. The length of the minimum-length Dubins path for this initial condition and scenario but **without obstacles** is 83.6 m. One might expect that the optimal path length in the presence of obstacles could be considerably larger than this value, as the car must avoid several obstacles in order to reposition itself and navigate towards the goal. The figure shows that more than 80% of the paths have a length of 120 m or less. Considering the numerous obstacles and the adverse initial orientation of the car (it does not even face towards the goal), these paths appear to be relatively efficient. For example, the length of the path in Figure 5-11b is 102.5 m, and it is clear that it does not wander around obstacles aimlessly. We can note that the harder initial condition shifts the distribution towards the right as compared to the previous cases, which indicates that, on average, the paths have slightly longer length.

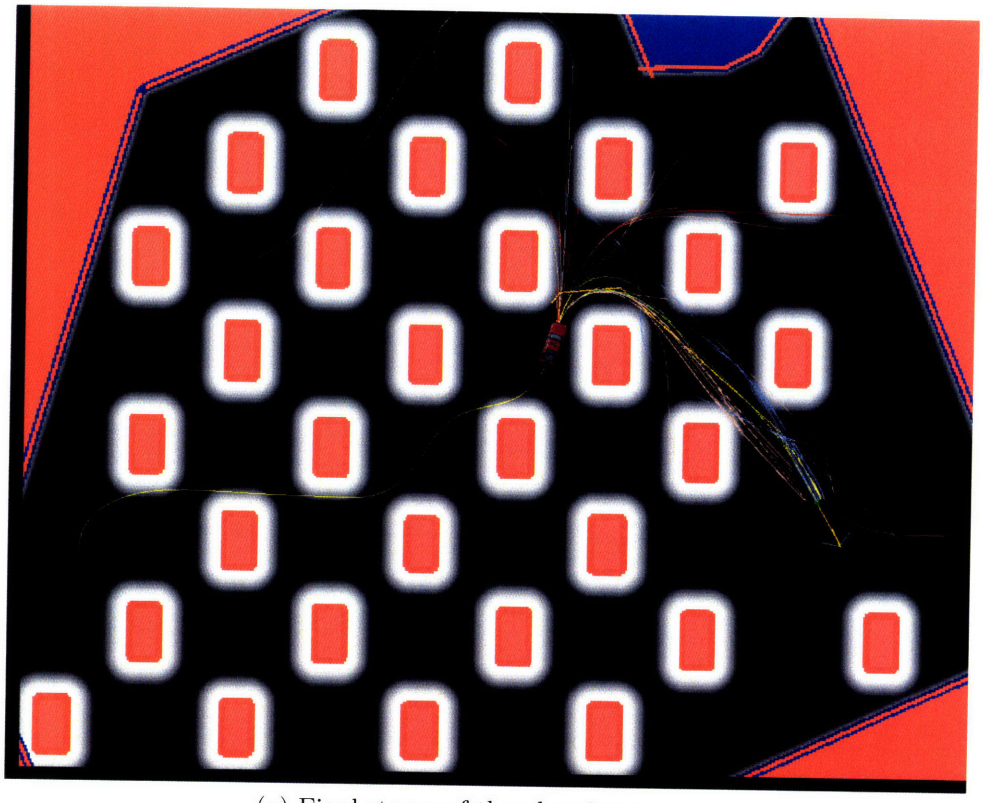


(a) Initial condition of Talos.

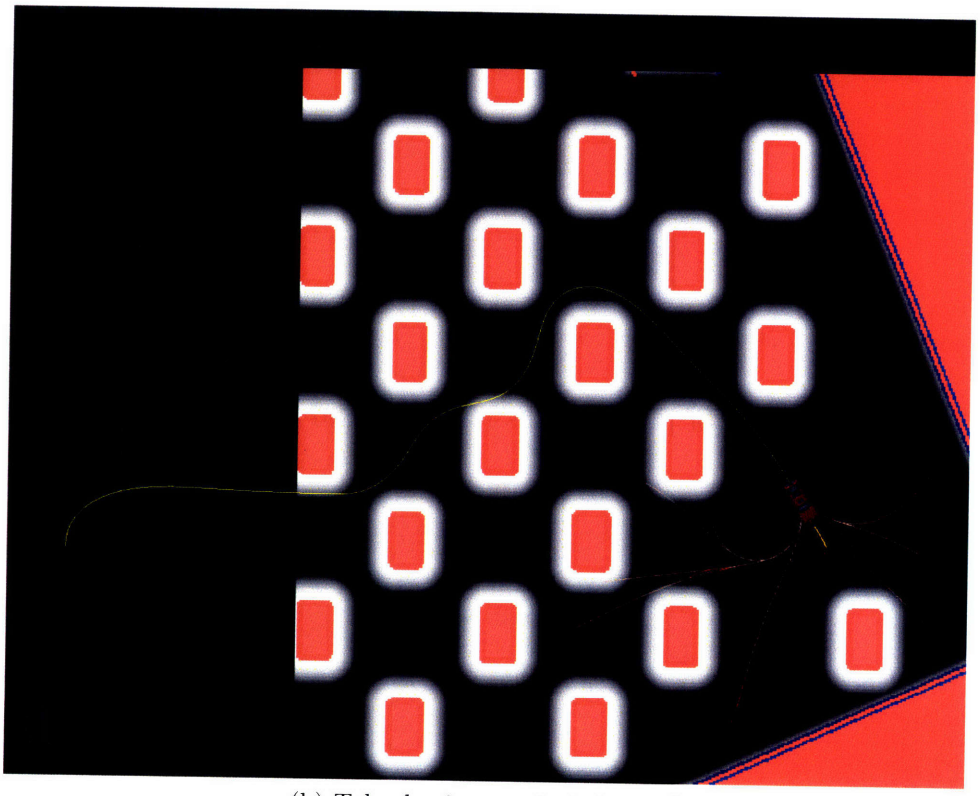


(b) Early in the planning process.

Figure 5-10: Planning through a dense obstacle field with an initial condition corresponding to far left and 90°.



(a) Final stages of the planning process.



(b) Talos having reached the goal.

Figure 5-11: Planning through a dense obstacle field with an initial condition corresponding to far left and 90°.

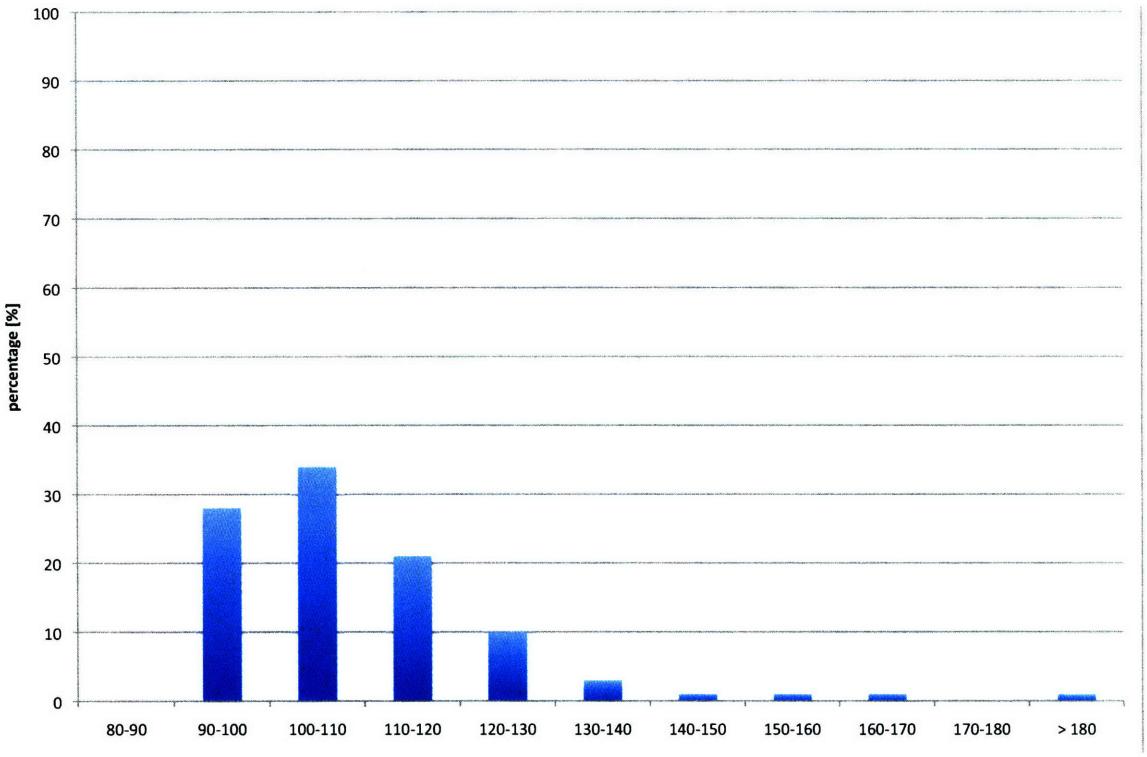
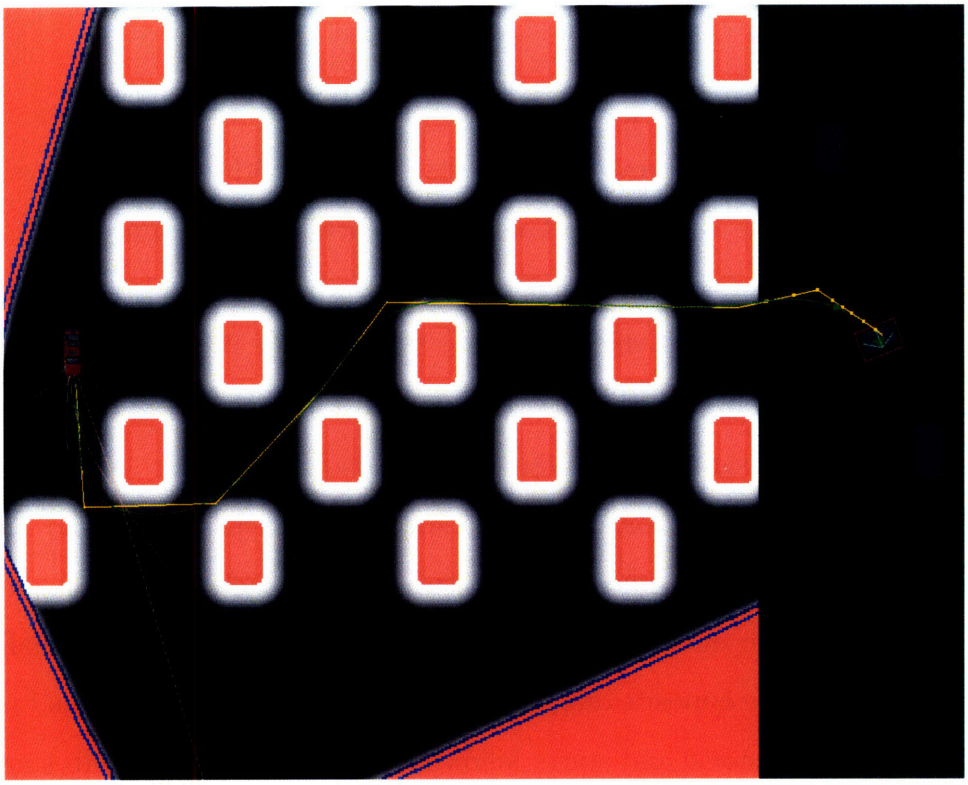


Figure 5-12: Distribution of path lengths for the initial conditions corresponding to far left and 90°.

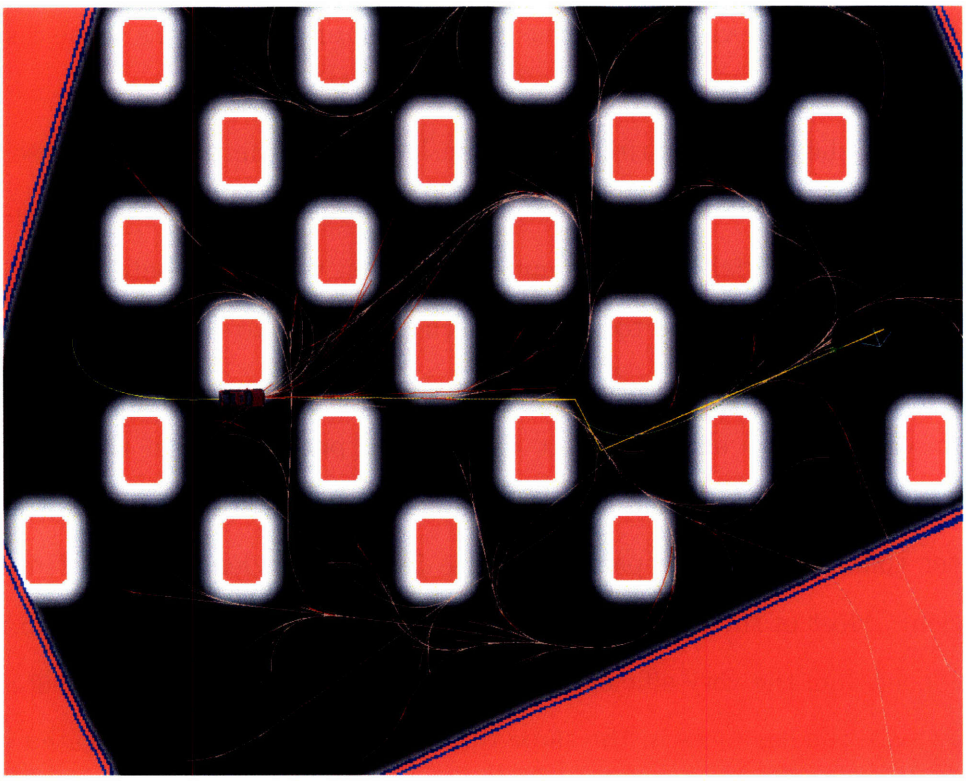
Scenario 4.3: Dense Obstacle Field, 270° Initial Orientation

The third test in this set corresponds to an initial condition with the vehicle located in the same position as the previous two tests but now with an initial direction of 270° with respect to the horizontal. This test is represented in Figures 5-13 and 5-14. Figure 5-13a shows this initial condition. Figures 5-13b and 5-14a show the evolution of the tree as Talos drives towards the goal. In Figure 5-13b, as in the two previous tests, several paths come very close to the goal but still none of them reach it because of not satisfying the direction threshold constraint. Notice that the tree covers a substantial portion of the free space and reaches to far and intricate places. In Figure 5-14a now numerous paths successfully reach the goal. A path was found that satisfied the direction threshold constraint by going up diagonally and then coming back in the opposite diagonal direction. As a consequence, the car veers to its left before an obstacle and starts following this path. Figure 5-14b shows the entire path followed by Talos, which has a length of 98.7 m. We will see that this path happened to be very short based on the distribution of path lengths to be discussed next.

Figure 5-15 shows the distribution in the lengths of the paths for these initial conditions. The number of trials was set to 100. The length of the minimum-length Dubins path for this initial conditions and scenario but **without obstacles** is 84.0 m. It is impossible for the vehicle to follow a path even close to the obstacle-free path, as there are numerous obstacles between the initial position of the vehicle and the goal. Consequently, we should expect the optimal path length in the presence of obstacles to be noticeably larger. The figure shows that more than 80% of the paths are shorter than 140 m. The path in Figure 5-14b has a length of 98.7 m and is very efficient. Most of the paths have a length in the range 110–120 m, which is only 10–20% longer than the very efficient path shown. Note that the initial orientation in this test, where the car faces in the same direction of the goal but is located behind it, is the hardest among the tests that have been considered. The results clearly show the corresponding increase of the path lengths.

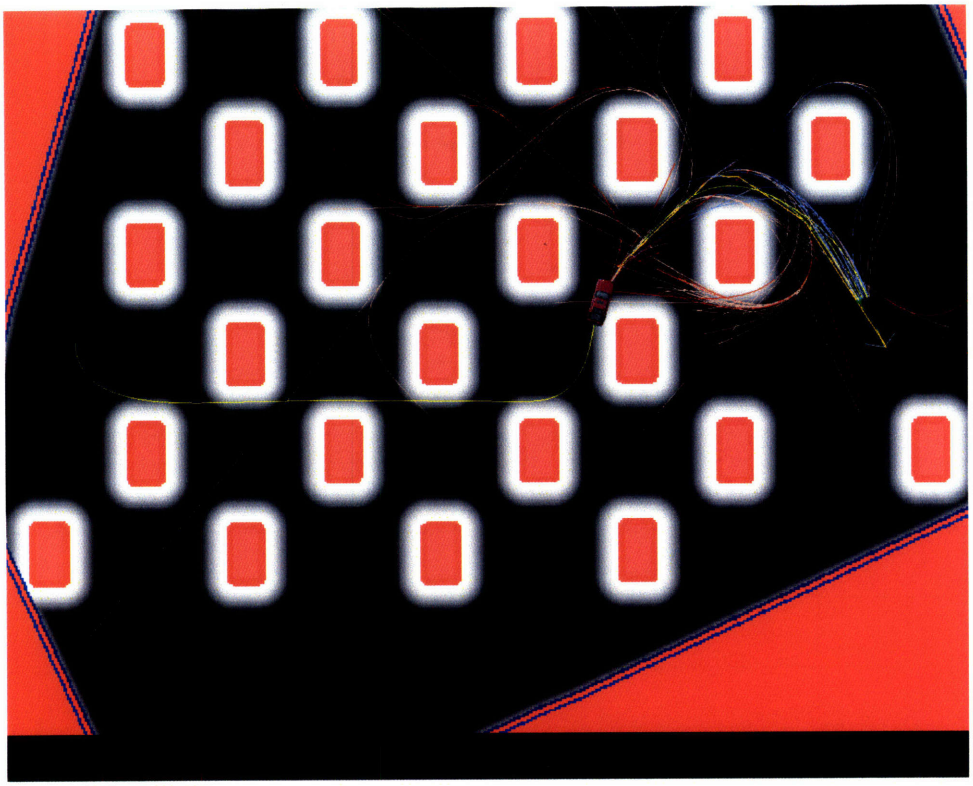


(a) Initial condition of Talos.

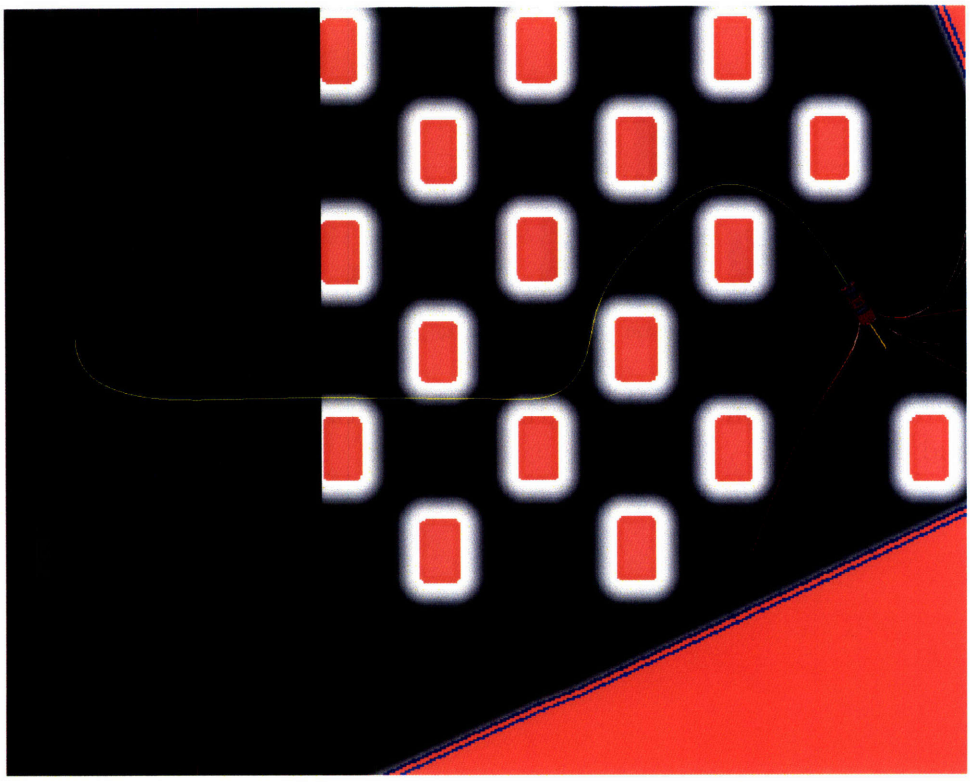


(b) Early in the planning process.

Figure 5-13: Planning through a dense obstacle field with an initial condition corresponding to far left and 270°.



(a) Final stages of the planning process.



(b) Talos having reached the goal.

Figure 5-14: Planning through a dense obstacle field with an initial condition corresponding to far left and 270°.

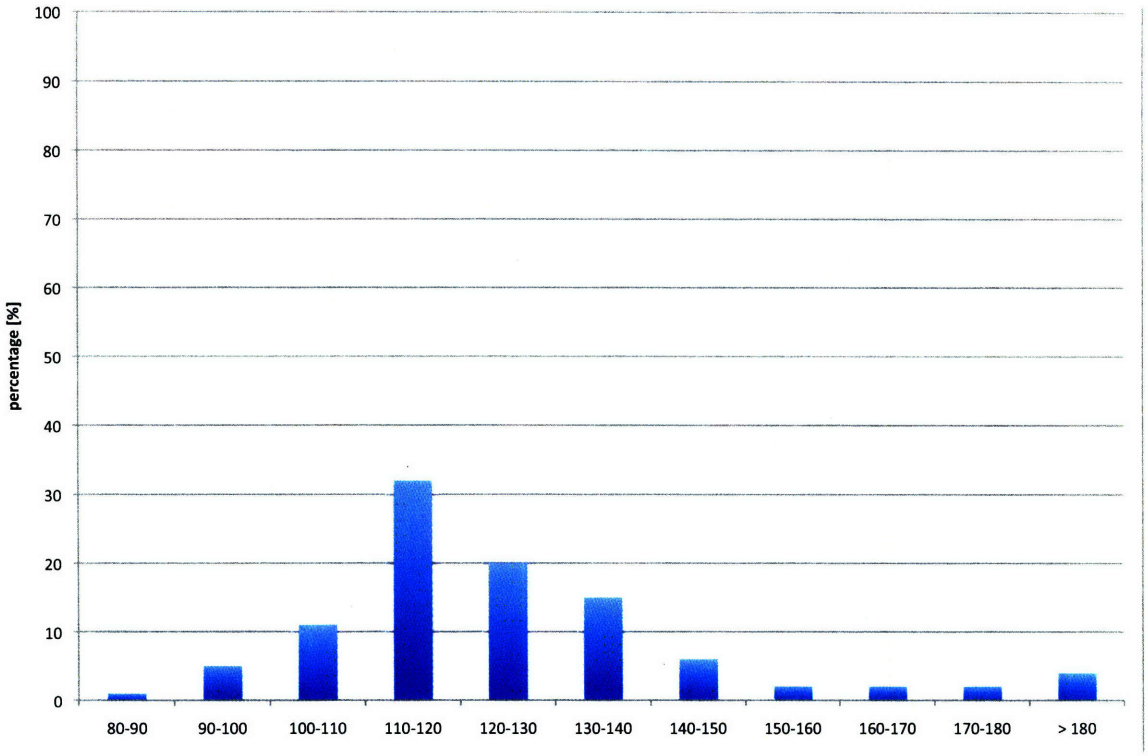


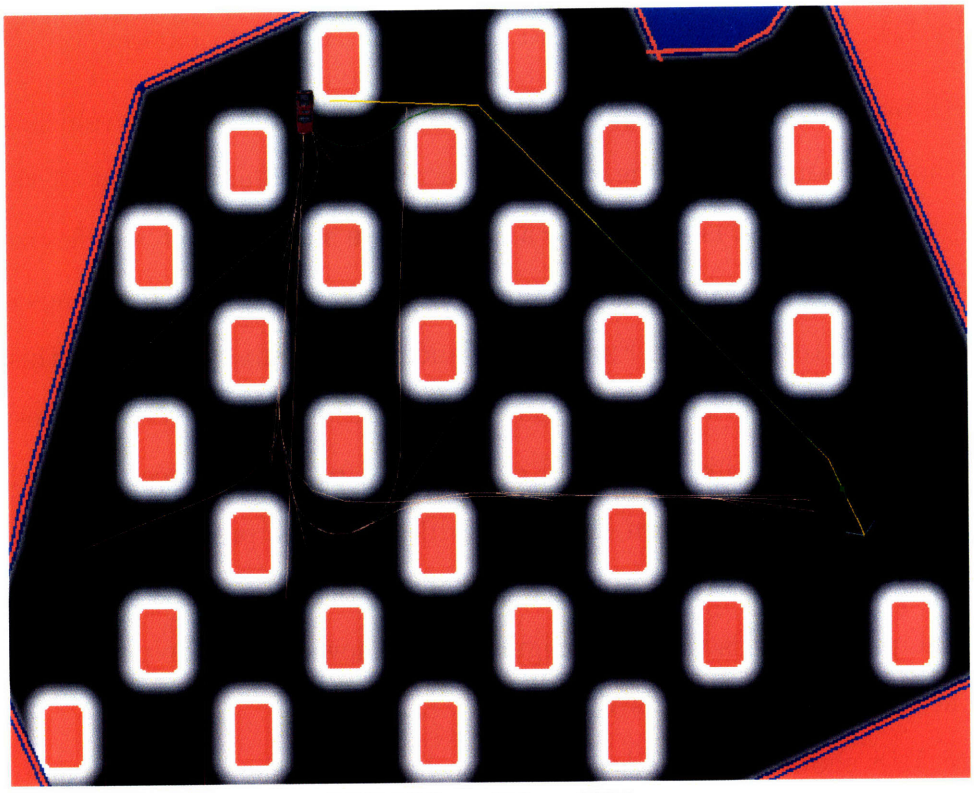
Figure 5-15: Distribution of path lengths for the initial conditions corresponding to far left and 270°.

Scenario 4.4: Dense Obstacle Field, Different Initial Location

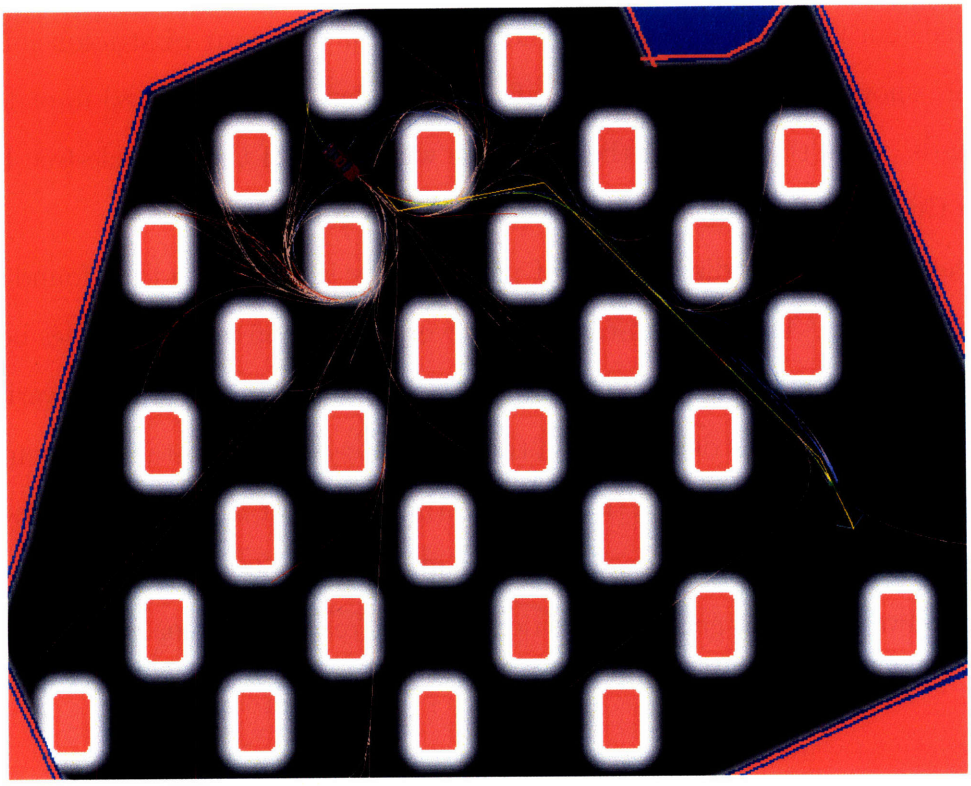
The fourth test in this set corresponds to an initial condition with the vehicle located next to the top boundary of the zone with a direction similar to that of the goal. The purpose of this case is to serve as a control case showing that a different initial condition of the vehicle would still produce similar distributions to the ones already found. The initial results are in Figures 5-16 and 5-17. Figure 5-16a shows this initial condition. Figures 5-16b and 5-17a show the evolution of the tree as Talos drives towards the goal. For this case, even early in the planning process several paths already reach the goal, as shown in Figure 5-16b. As time progresses and Figure 5-17a shows, the paths that reach the goal are further refined and the vehicle follows a straight line path towards the goal. Figure 5-17b shows the entire path followed by Talos. The length of this path is 74.5 m.

Figure 5-18 shows the distribution in the lengths of the paths for these initial conditions. The number of trials was set to 50. The length of the minimum-length Dubins path for this initial conditions and scenario but **without obstacles** is 73.0 m. The figure shows that more than 90% of the paths have a length of 80 m or less. In other words, more than 90% of the paths are within 9% of the obstacle-free Dubins path. In this case, the lengths of the paths are very efficient.

We can see that the distribution of path lengths is similar to that found in the previous test cases. This initial location, however, resulted in fewer obstacles between the initial position of the car and the location of the goal. As a result, the variability in the length of the paths decreased. In the limiting case of no obstacles, the motion-planning algorithm would plan a trajectory to the goal as straight as possible, subject to the car dynamics constraints. This is because as described in Section 4.6.2, the algorithm always tries to generate a trajectory that reaches the goal directly.

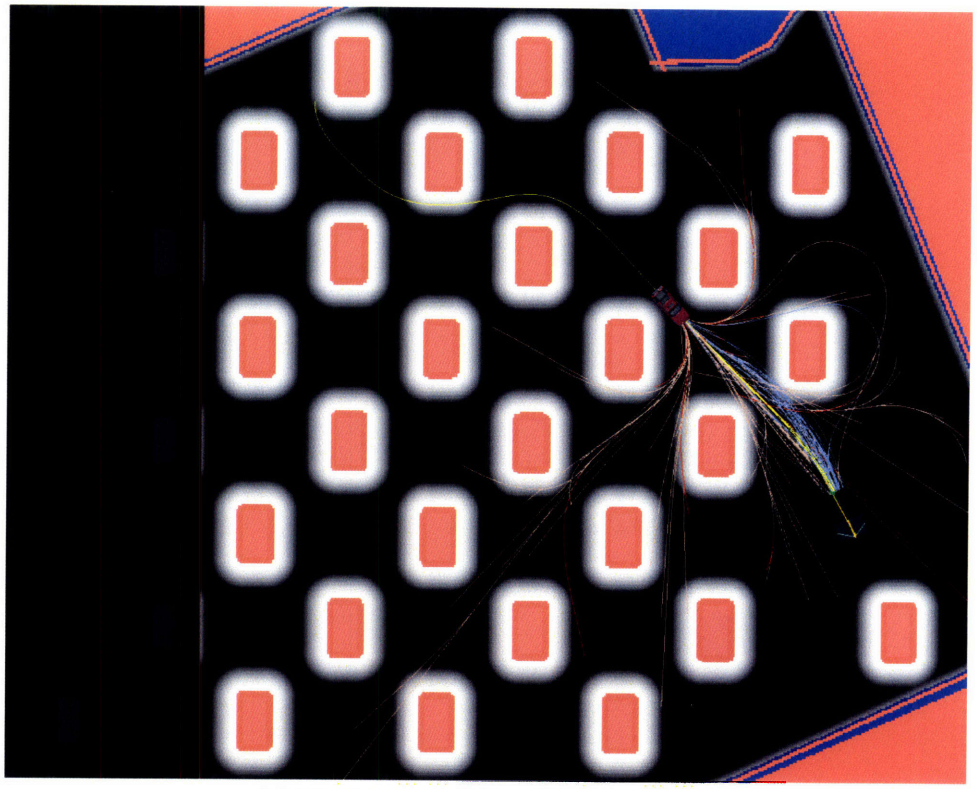


(a) Initial condition of Talos.

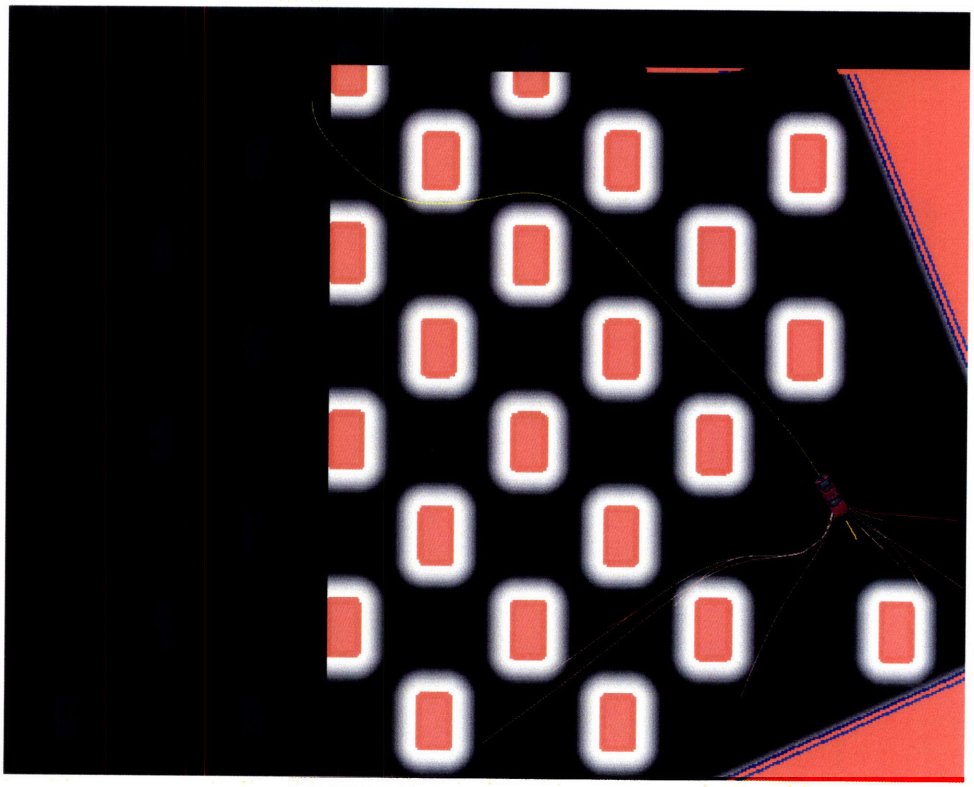


(b) Early in the planning process.

Figure 5-16: Planning through a dense obstacle field with an initial condition corresponding to far top and facing the goal.



(a) Final stages of the planning process.



(b) Talos having reached the goal.

Figure 5-17: Planning through a dense obstacle field with an initial condition corresponding to far top and facing the goal.

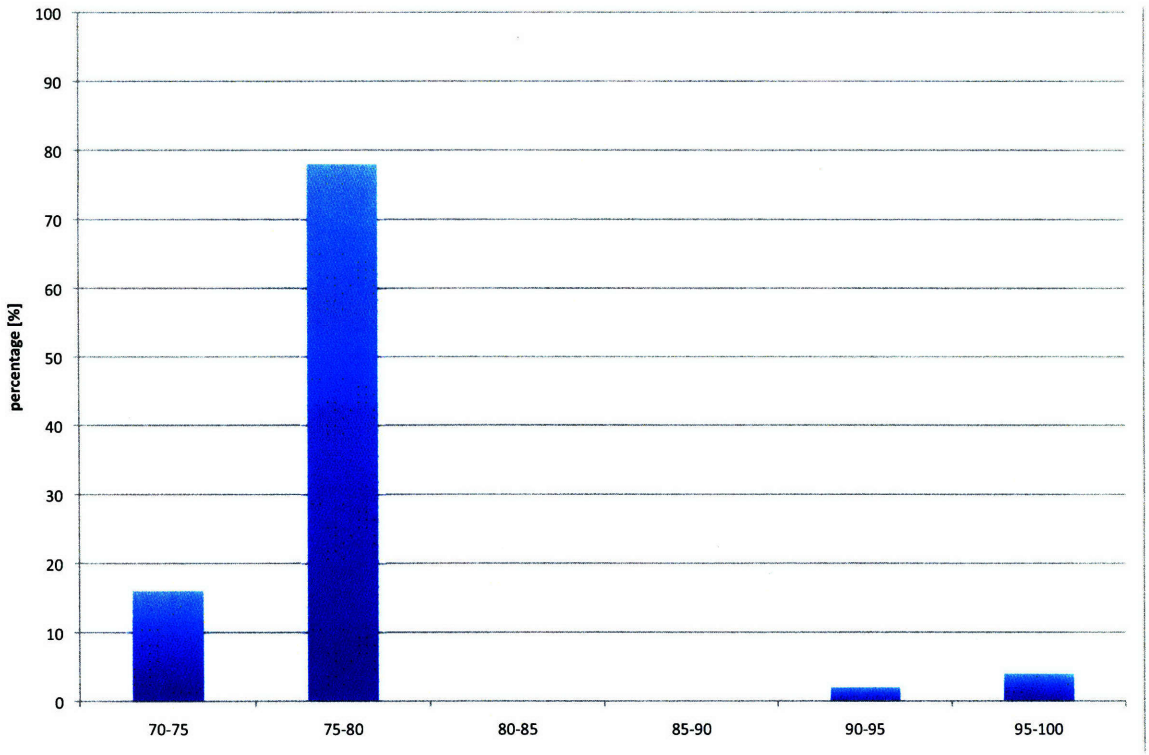


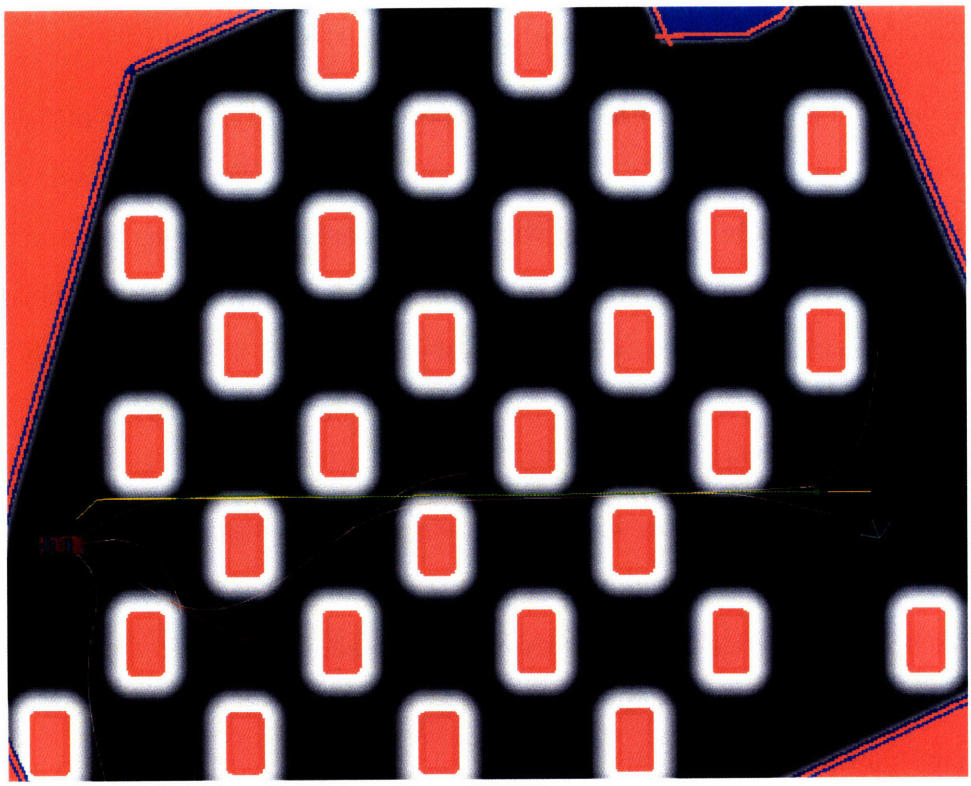
Figure 5-18: Distribution of path lengths for the initial conditions corresponding to far top and facing the goal.

Scenario 4.5: Dense Obstacle Field, Different Hazard Weights

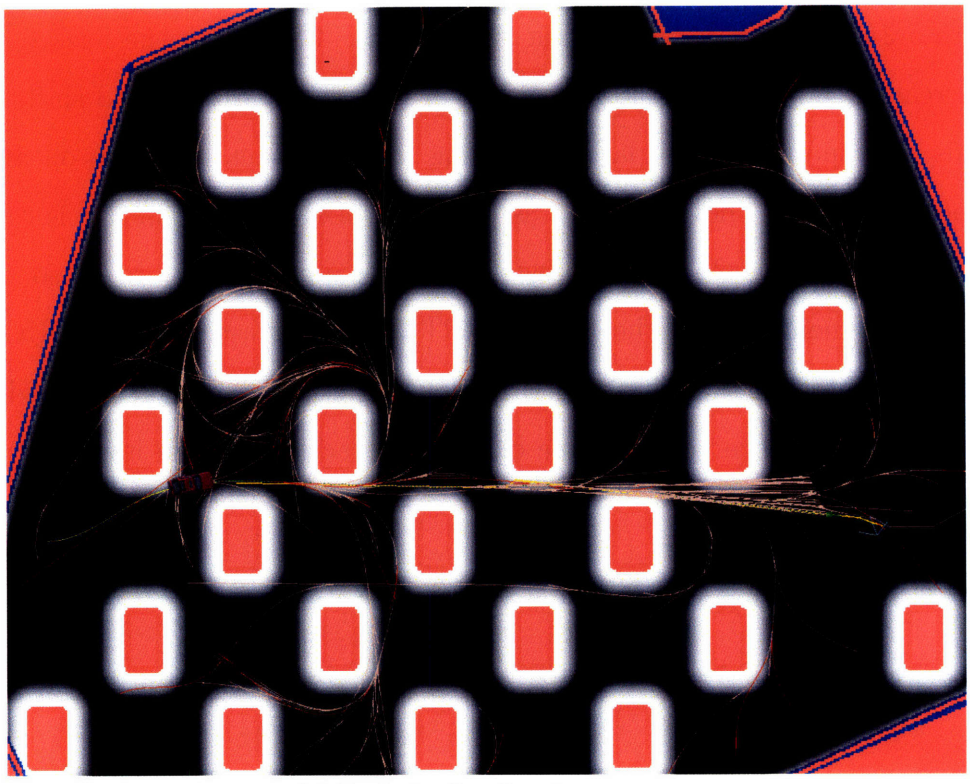
The fifth and final test in this set has as its purpose to analyze the impact of the hazard weight w described in Section **??** on the length of the paths. For the obstacle scenario already presented and the car starting with initial conditions of being located next to the left boundary of the field with an direction of 0° with respect to the horizontal, the value of the hazard weight w was decreased to $w/2$, $w/5$, $w/10$, and finally $w = 0$. The number of trials corresponding to each different value of w was 50.

Figures 5-19 and 5-20 show a random trial with $w = 0$. It is apparent that, compared with the case of 0 deg initial direction shown in Figures 5-7 and 5-8 where w was not altered from the value used during the DUC ($w = 0.005$), the car tended to come closer to obstacles and took advantage of a shorter path to the goal. In this case, the length of the path shown in Figure 5-20b is 90.7 m as compared to 95.4 m for the path shown in Figure 5-8b. This tendency is confirmed by considering a large number of trials, as is presented next.

Figure 5-21 shows the results of the simulations. Clearly, as the hazard weight value decreases, the distribution of path lengths becomes narrower and shifts to the left. This corresponds to the paths becoming more efficient in length on average. It is important to note, however, that as the value of the hazard weight is reduced, the car will tend to travel closer to obstacles if it is advantageous to do so. In the particular implementation of the Robust RRT algorithm used for the DUC, each obstacle represented in the drivability map had an extra 30 cm on each side, and therefore although the car might come close to obstacles, collision itself should not be an issue.

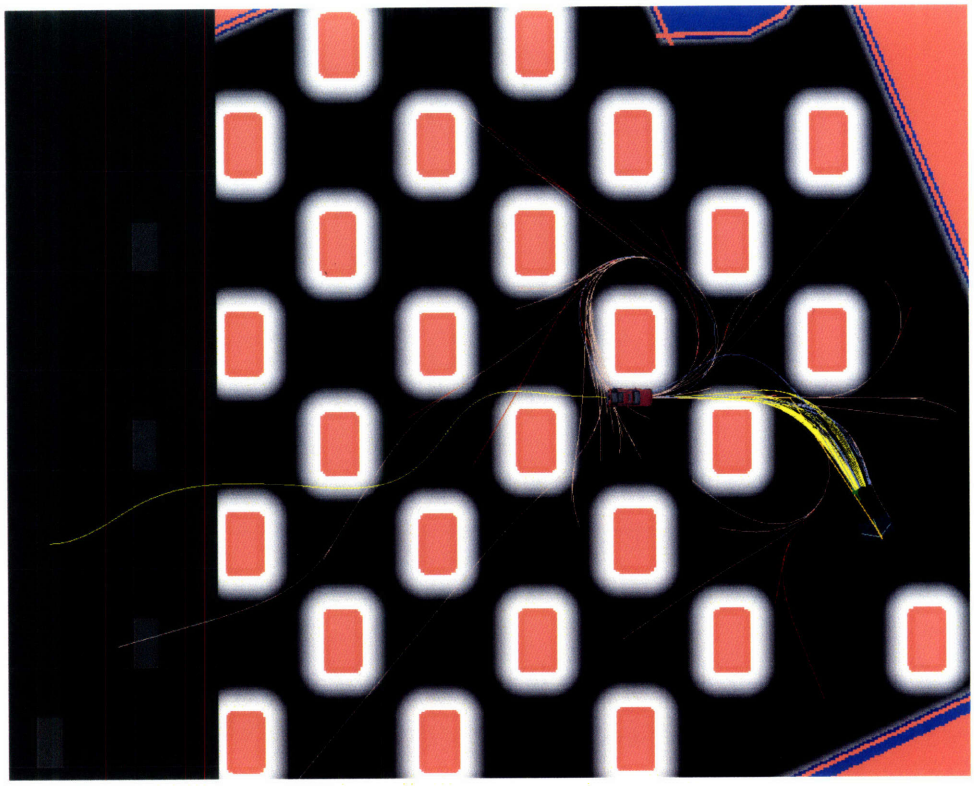


(a) Initial condition of Talos.

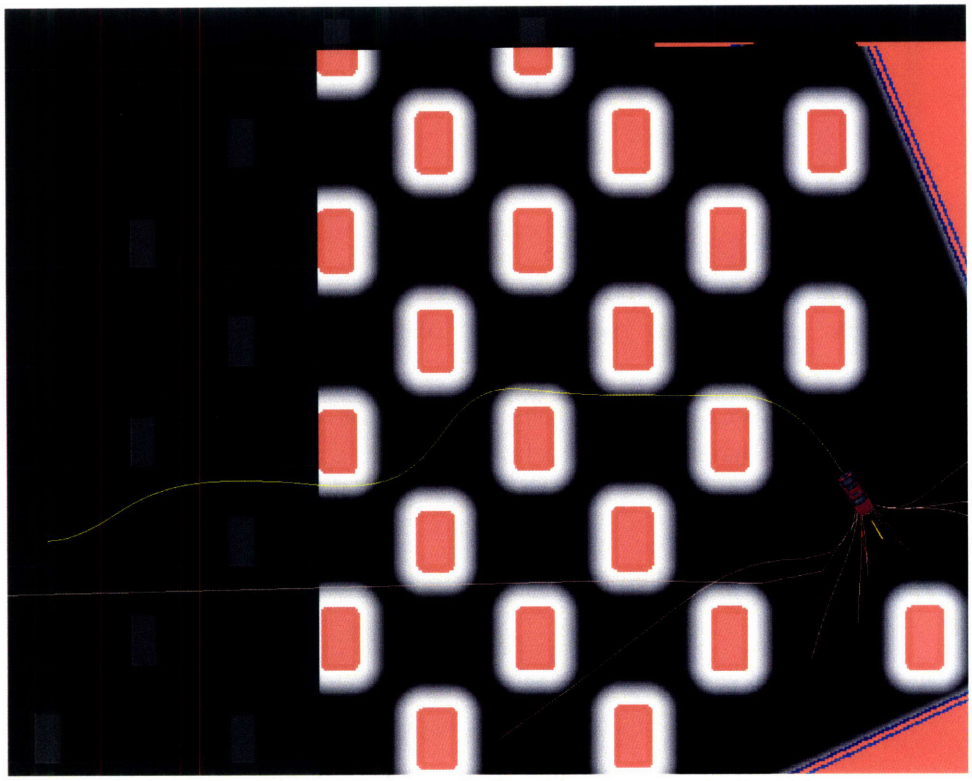


(b) Early in the planning process.

Figure 5-19: Planning through a dense obstacle field with hazard weight set to $w = 0$ and an initial condition corresponding to far left and 0°.



(a) Final stages of the planning process.



(b) Talos having reached the goal.

Figure 5-20: Planning through a dense obstacle field with hazard weight set to $w = 0$ and an initial condition corresponding to far left and 0°.

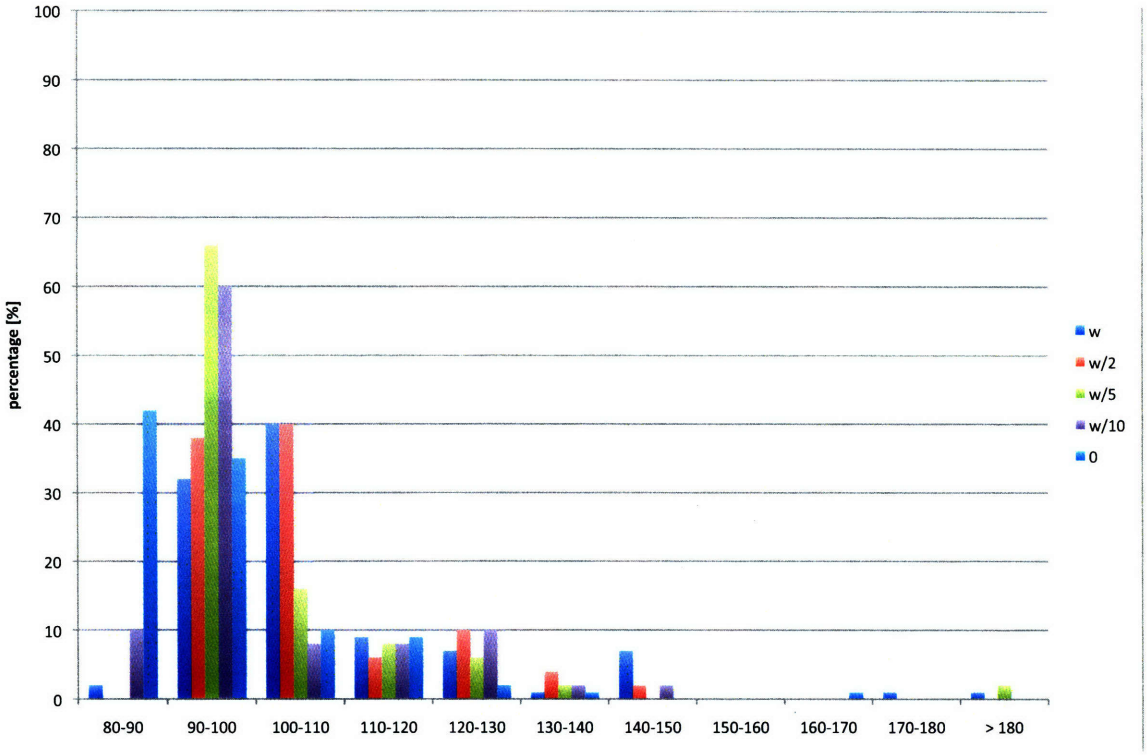


Figure 5-21: Distribution of path lengths for different values of the hazard weight w and initial conditions corresponding to far left and 0°.

Scenario 5: Narrow Passage

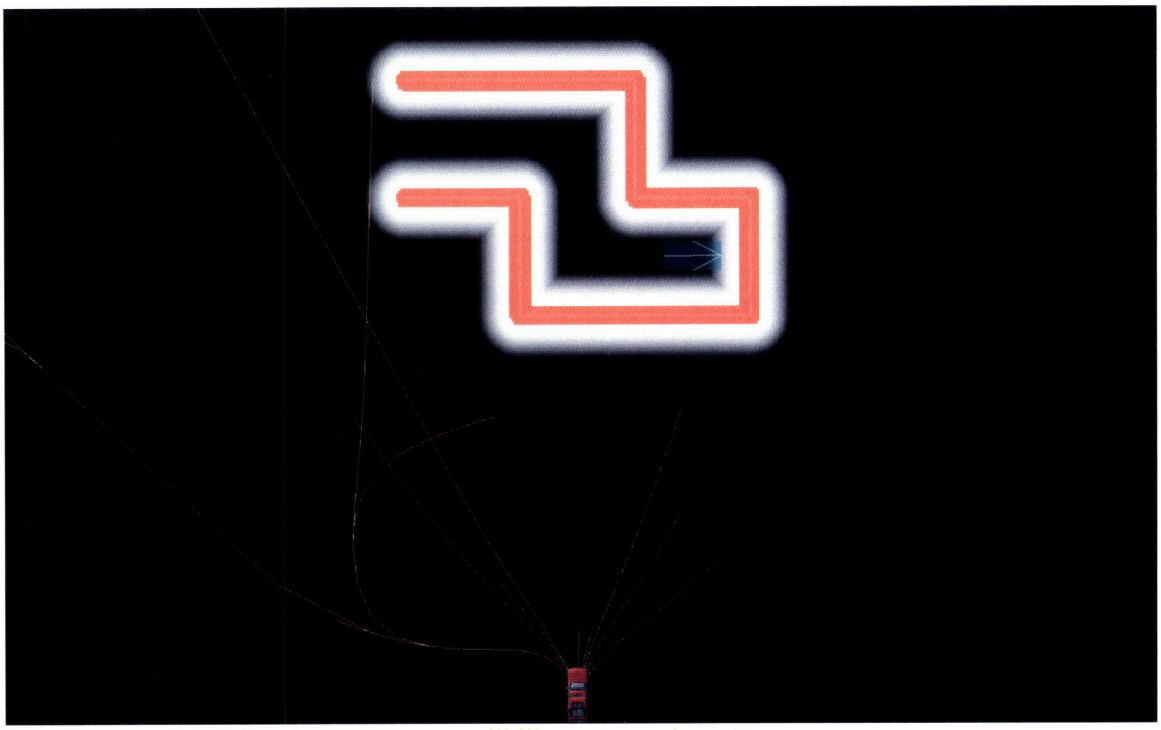
The next set of simulation tests has the objective of evaluating the performance of the Robust RRT algorithm in hard scenarios that extend beyond the plausible circumstances to be encountered in a urban environment. These results will clearly show the adaptability of the Robust RRT algorithm to different scenarios without requiring any modifications to the code from what was implemented for the DUC.

We present the results of testing the planning capabilities of the Robust RRT algorithm for a narrow passage as well as for very wide passage with a dead end. These two scenarios can be considered as opposites in terms of the requirements that they impose on the motion-planning algorithm. While the narrow passage requires the capability to find paths along a very narrow and potentially long region, the dead end scenario requires the capability to be able to plan broadly in the space.

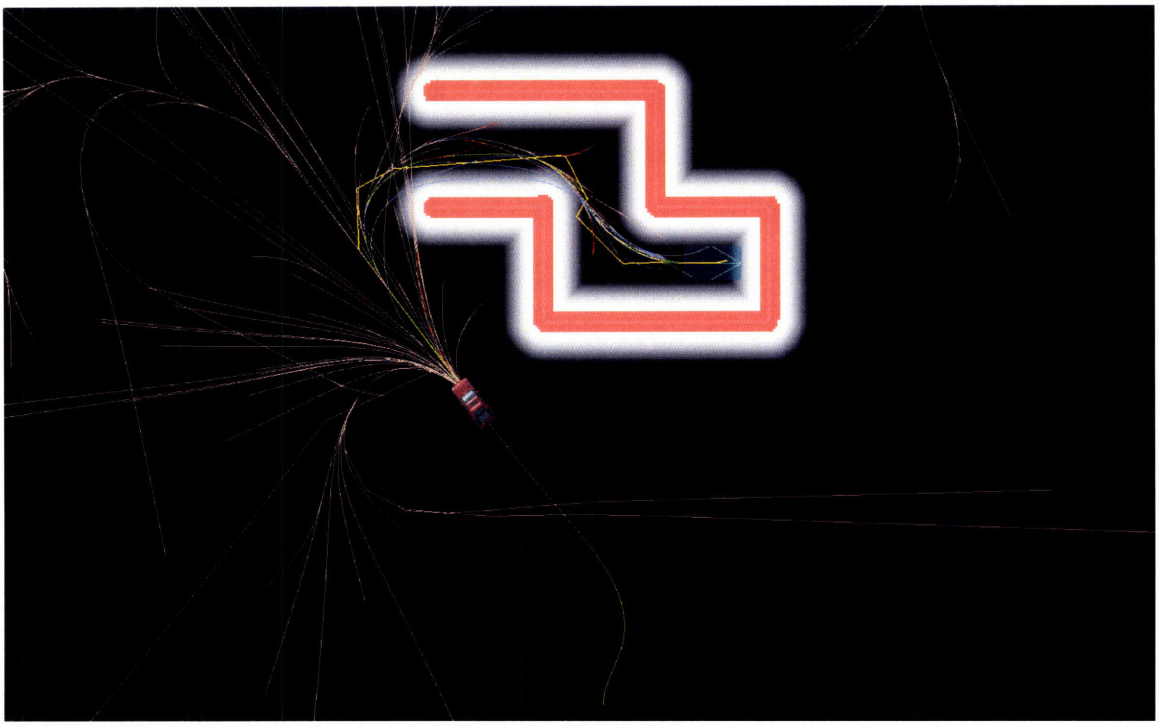
Figures 5-22–5-24 show the narrow passage scenarios used in the tests. In all of scenarios, the width of the passage is 10 m. The initial condition of the vehicle is a location 30 m from the lowest part of the structure and facing towards it. The difference among the narrow passage scenarios is the length of the segment corresponding to the entrance into the passage. For the case in Figure 5-22, the length of the entrance segment was set to 10 m, for the case in Figure 5-23 the length was set to 15 m, and finally for the case in Figure 5-24 this length was set to 20 m.

Figures 5-22a, 5-23a and 5-24a show the initial conditions of the vehicle for each case. Figures 5-22b, 5-23b and 5-24b show, for each case, a random snapshot as Talos drives towards the goal inside the passage with the purpose of showing the tree.

For each of these scenarios, we evaluated the success rate of the Robust RRT algorithm in finding a path to the goal in less than 10 s. The time interval of 10 s was chosen because it was considered as giving the algorithm moderate time to find a solution. The number of trials used in each scenario was 50. Figure 5-25 summarizes the results found, showing the percent of times that a solution was or was not found in less than 10 s. For the case corresponding to an entrance segment length of 10 m, a path that reached the goal was found almost 80% of the time, but this success rate

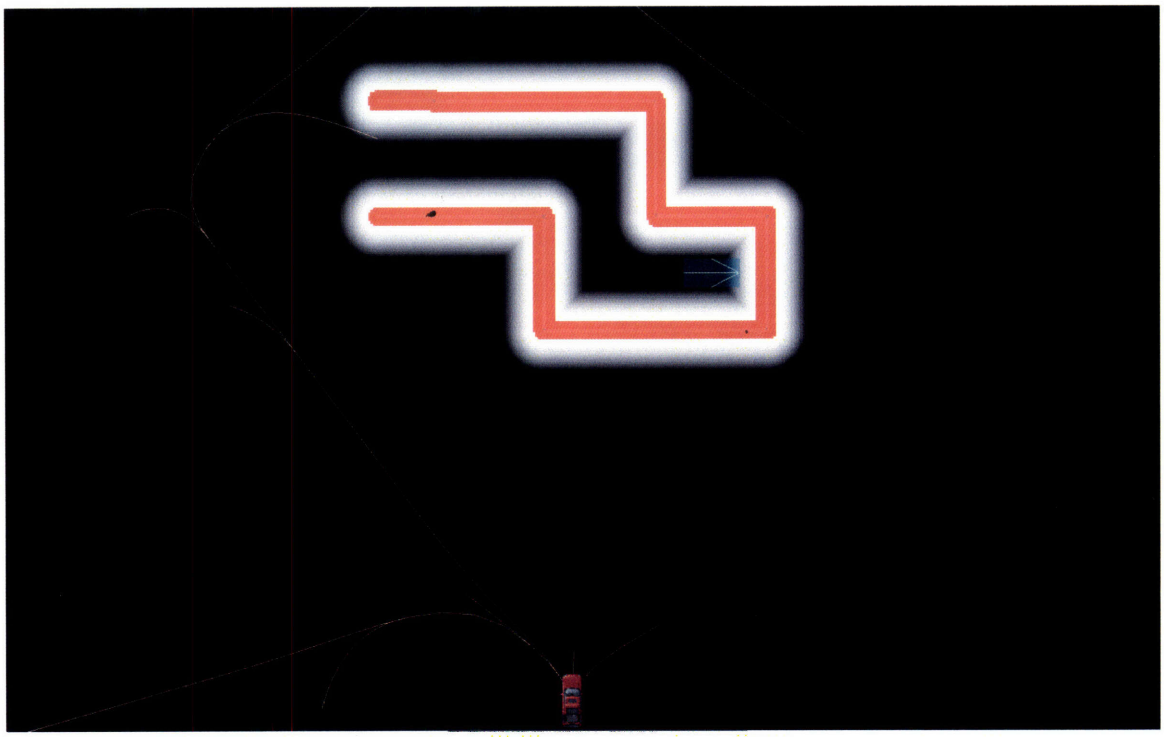


(a) Initial condition.

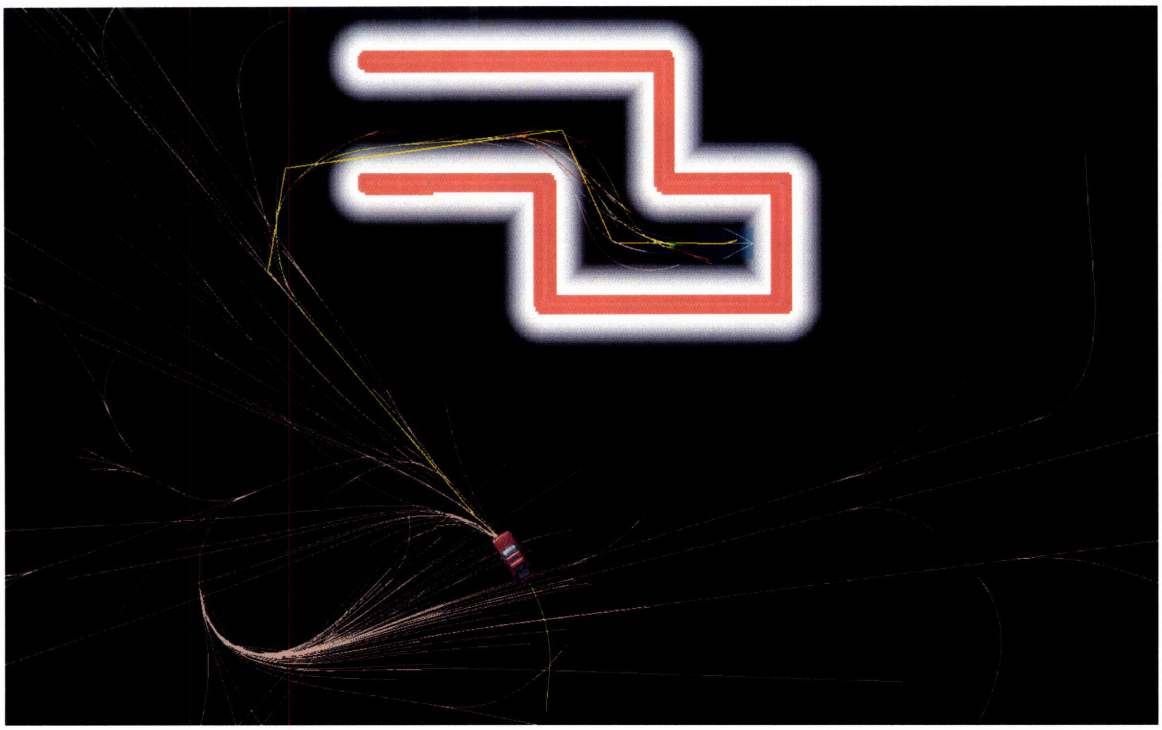


(b) Path found.

Figure 5-22: Planning in a narrow passage scenario with entrance segment set to 10 m.

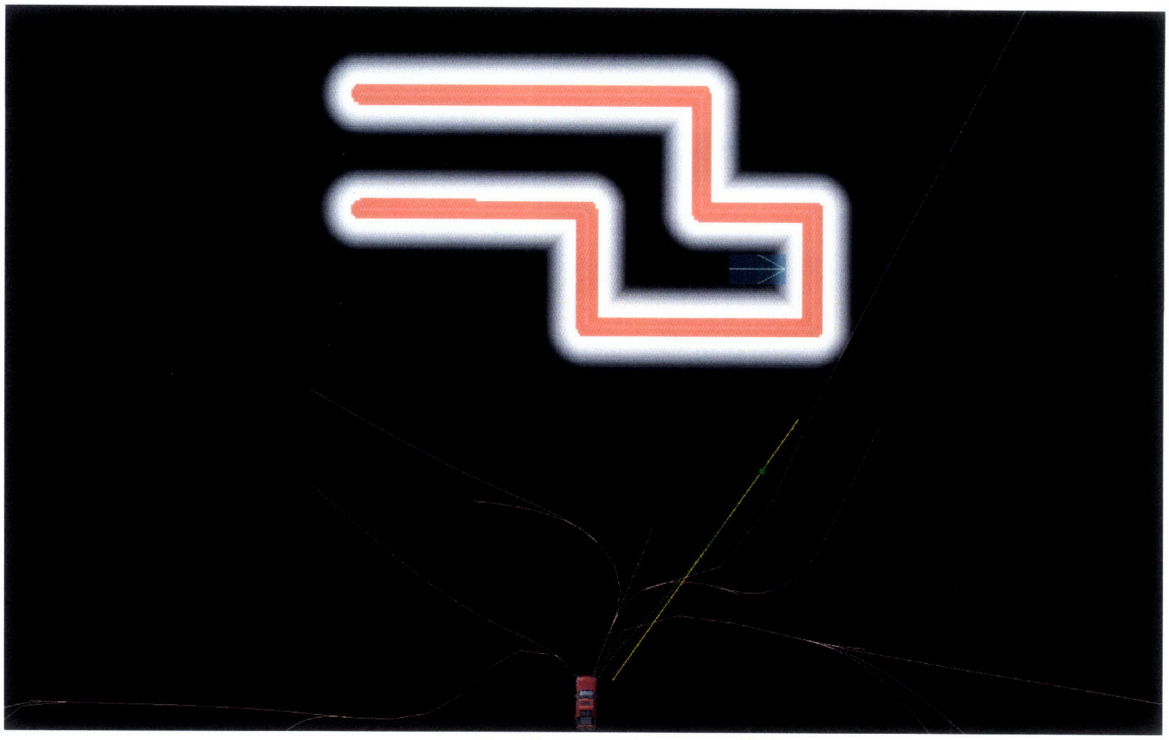


(a) Initial condition.

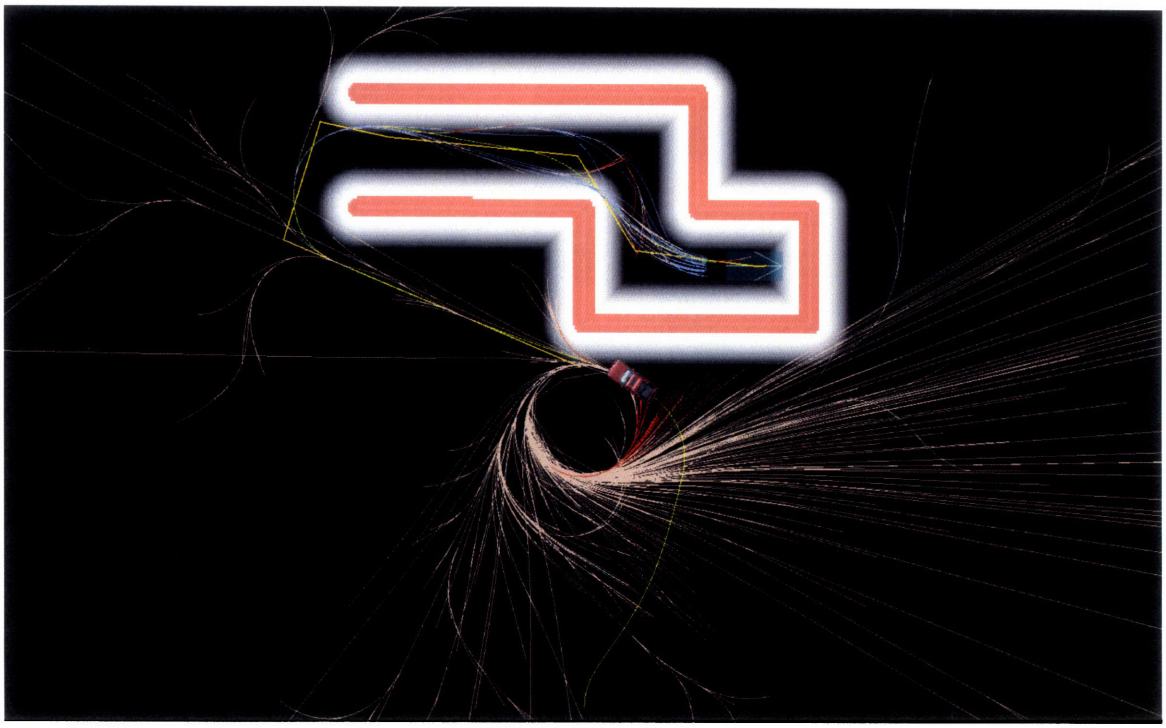


(b) Path found.

Figure 5-23: Planning in a narrow passage scenario with entrance segment set to 15 m.



(a) Initial condition.



(b) Path found.

Figure 5-24: Planning in a narrow passage scenario with entrance segment set to 20 m.

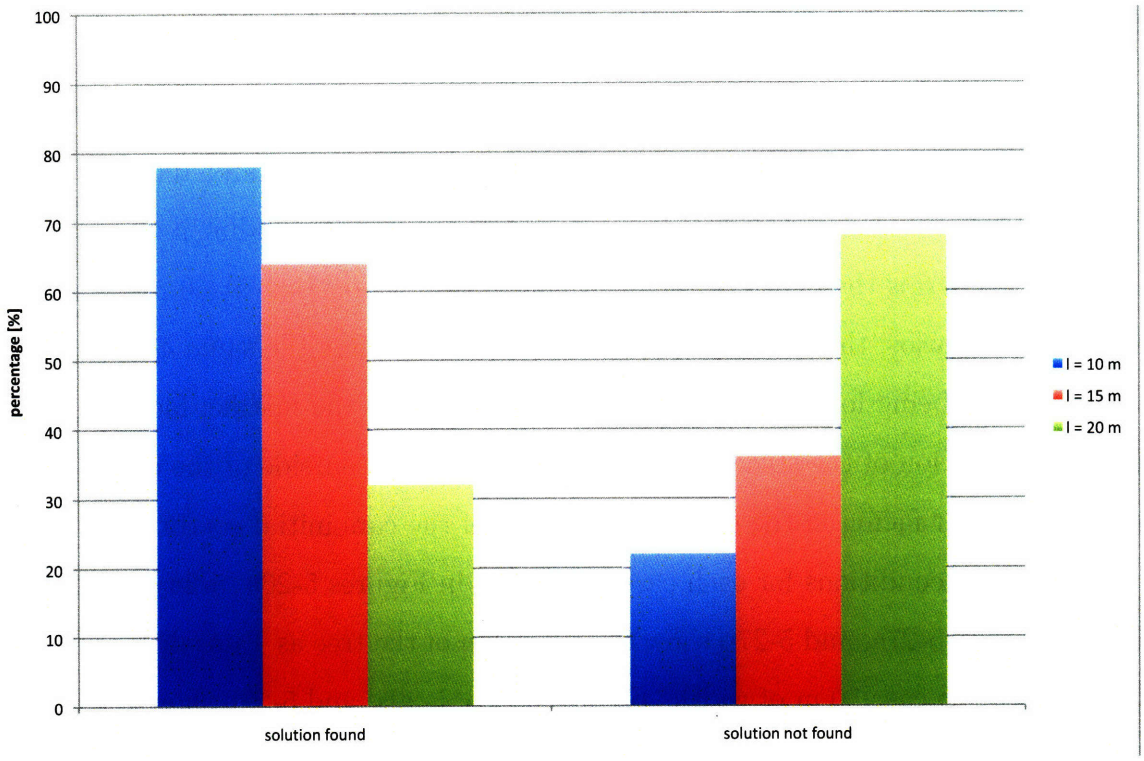


Figure 5-25: Success rate of the algorithm for a narrow passage with varying length of the entrance segment into the passsage.

dropped to slightly above 30% for the case corresponding to an entrance segment length of 20 m.

These results indicate that the Robust RRT algorithm struggles to find a path that reaches the goal in narrow passages that are long (or deep). A potential solution to this shortcoming would be to extend the algorithm to perform a bidirectional search, i.e., to grow two simultaneous RRTs, one from the initial state q_I and the other from the goal state q_G [48, p. 195]. However, this is not a simple modification in the case where the system to be controlled is a car. Propagating the dynamics equations of the car backwards would certainly yield an unstable system and this complication must be addressed in future work.

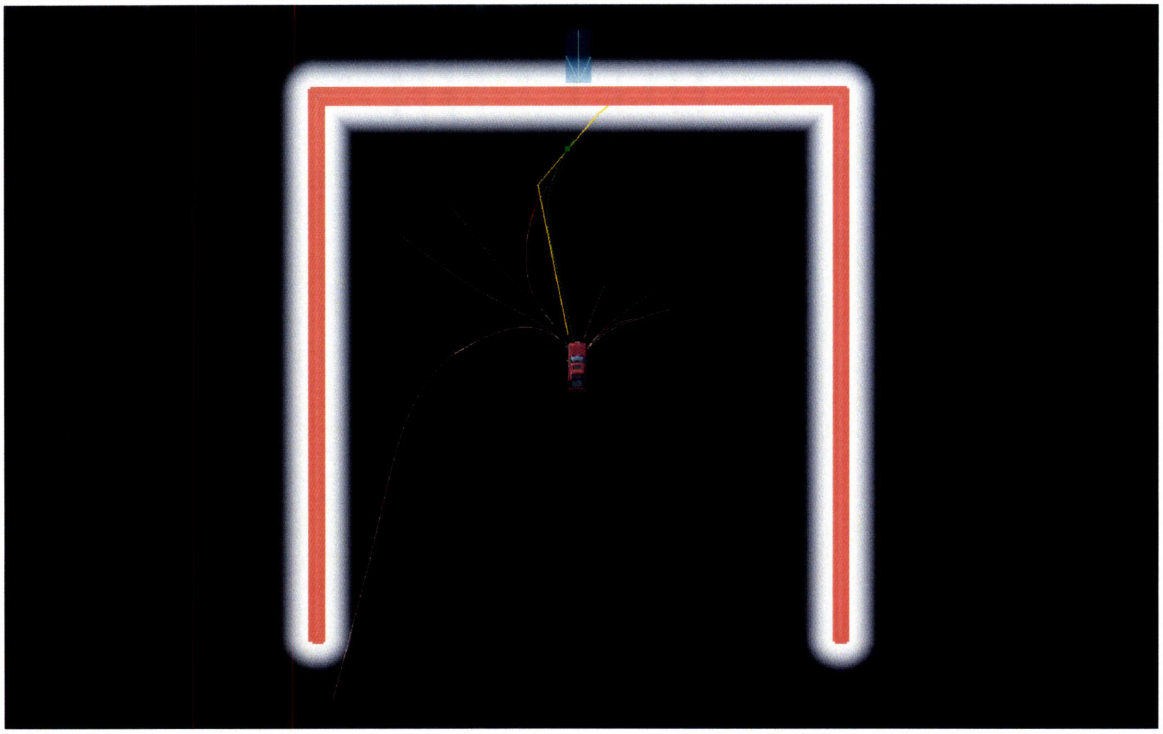
Scenario 6: Dead End

Now consider the dead end tests. In these tests, the variable changed was the width of the dead end road. A range of values was used (50 m, 75 m, and 100 m). Wider dead ends simplify turning around, but also greatly complicate finding a way out. For all of the scenarios the depth of the dead end remained constant at 50 m. Again for all of the scenarios, the initial conditions of the vehicle were the same and consisted of the vehicle being located 25 m inside the dead end on the middle, facing towards the closed portion of the structure. The goal was located outside of the dead end, at the middle, but facing in the opposite direction of the cars initial condition.

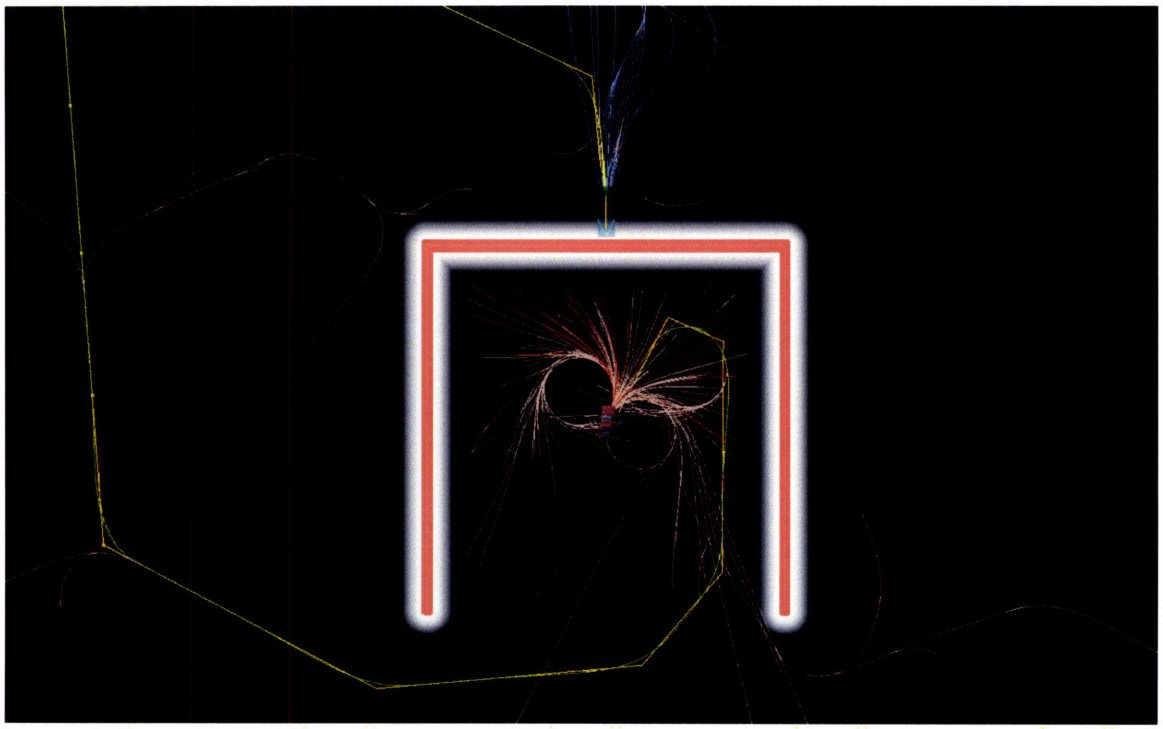
The initial conditions for each case are shown in Figures 5-26a, 5-28a, and 5-30a. Figures 5-26b, 5-27a, and 5-27b show the evolution of the tree as Talos drives towards the goal for the case 50 m of width. Figures 5-28b, 5-29a, and 5-29b show the case of 75 m of width. Figures 5-30b, 5-31a, and 5-31b show the case of 100 m of width.

Although the focus of these tests is on success rate, this figures are provided as additional prove of the exploration and later refinement capabilities of the Robust RRT algorithm. Each sequence of figures shows an original path that is arguably inefficient, but reaches the goal. This path is then gradually refined into a much more efficient path in term of length. Figures 5-27b, 5-29b, and 5-31b show the trace of the path followed by the vehicle for each of the different cases.

For each of the dead end scenarios, we evaluated the success rate of the Robust RRT algorithm in finding a path to the goal in less than 10 s. The number of trials used in each scenario was 50. Figure 5-32 summarizes the results found, showing the percent of times that a solution was or was not found in less than 10 s. For the cases corresponding to a dead end width of 50 m and 75 m, a path to the goal was found 100% of the time. For the case in which this width was set to 100 m, the success rate dropped slightly but still remained above 95%. These results reinforce the excellent quality of the Robust RRT algorithm in that it is able to very quickly find feasible paths in wide open spaces.

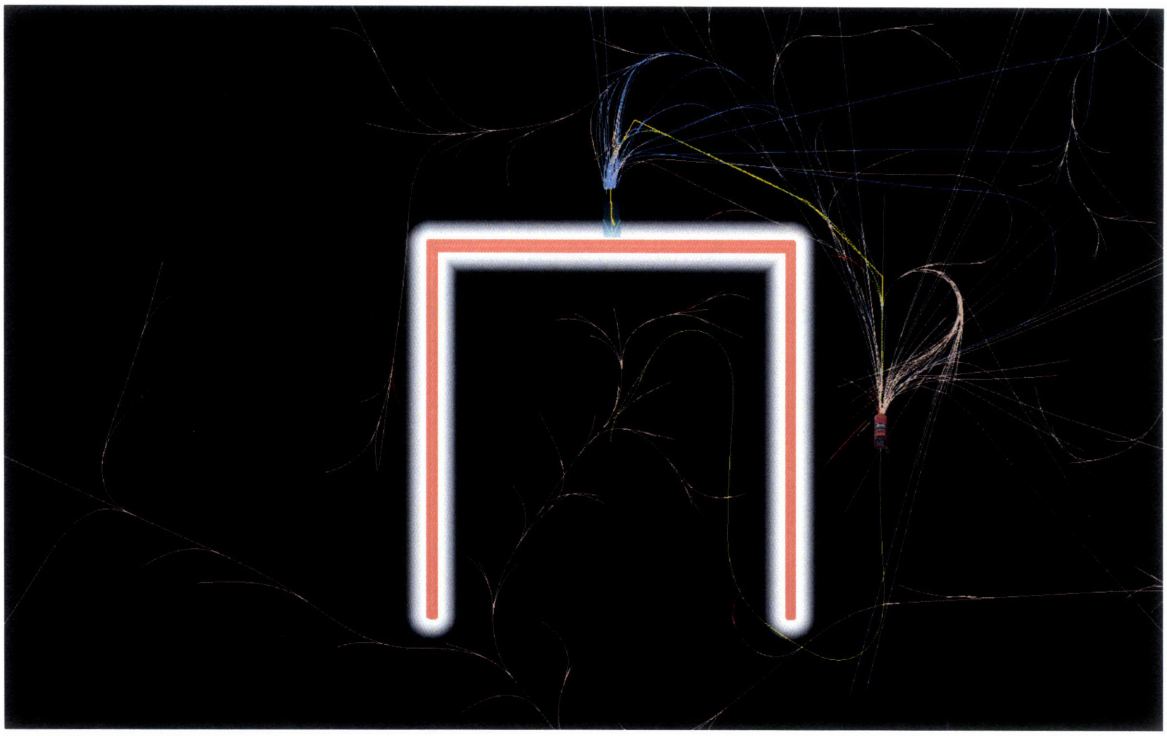


(a) Initial condition of Talos.

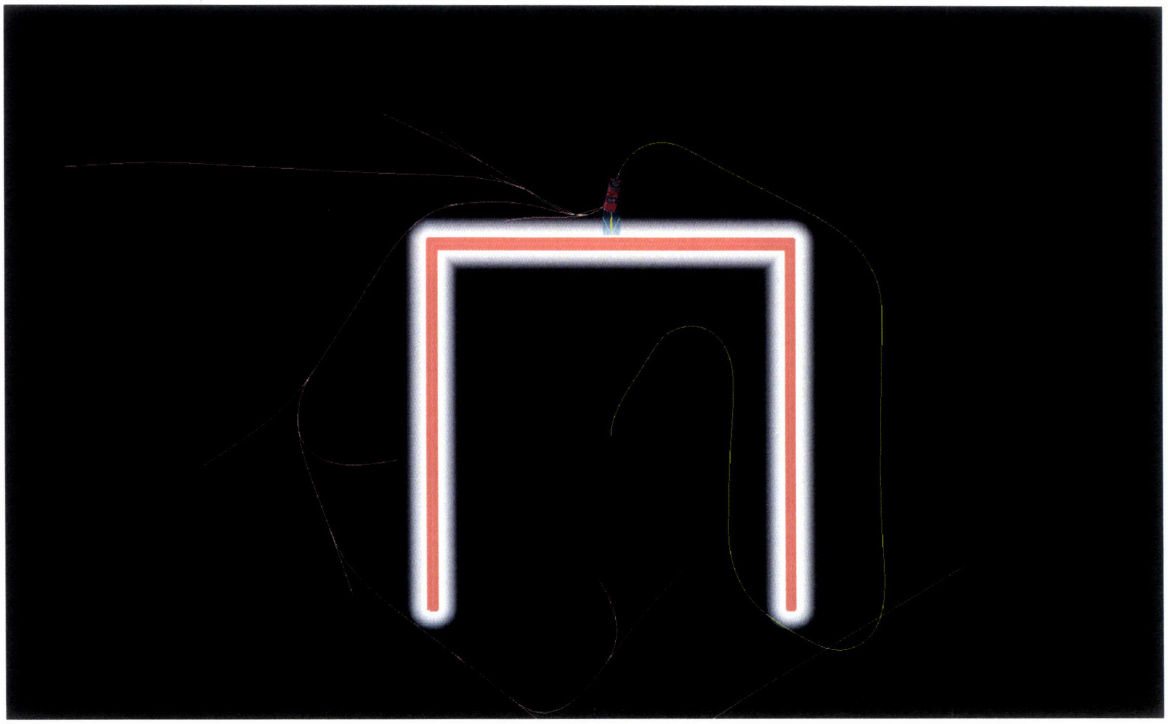


(b) Early in the planning process.

Figure 5-26: Planning in a dead end scenario with dead end width set to 50 m.

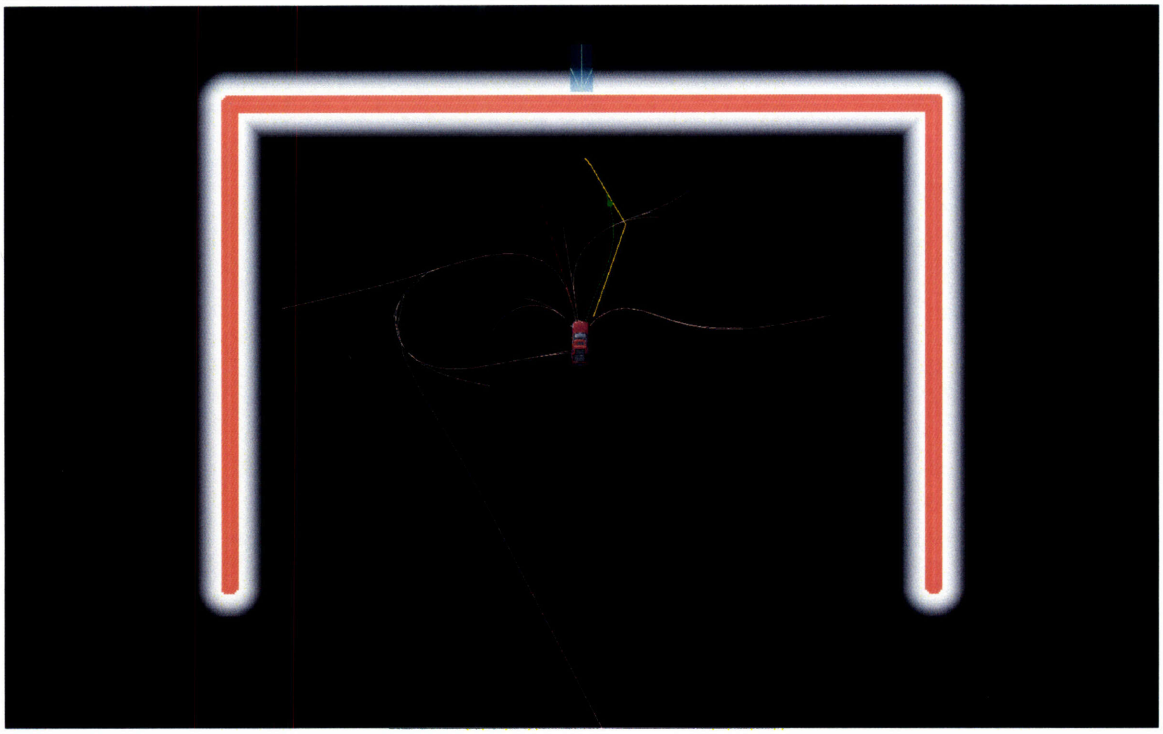


(a) Final stages of the planning process.

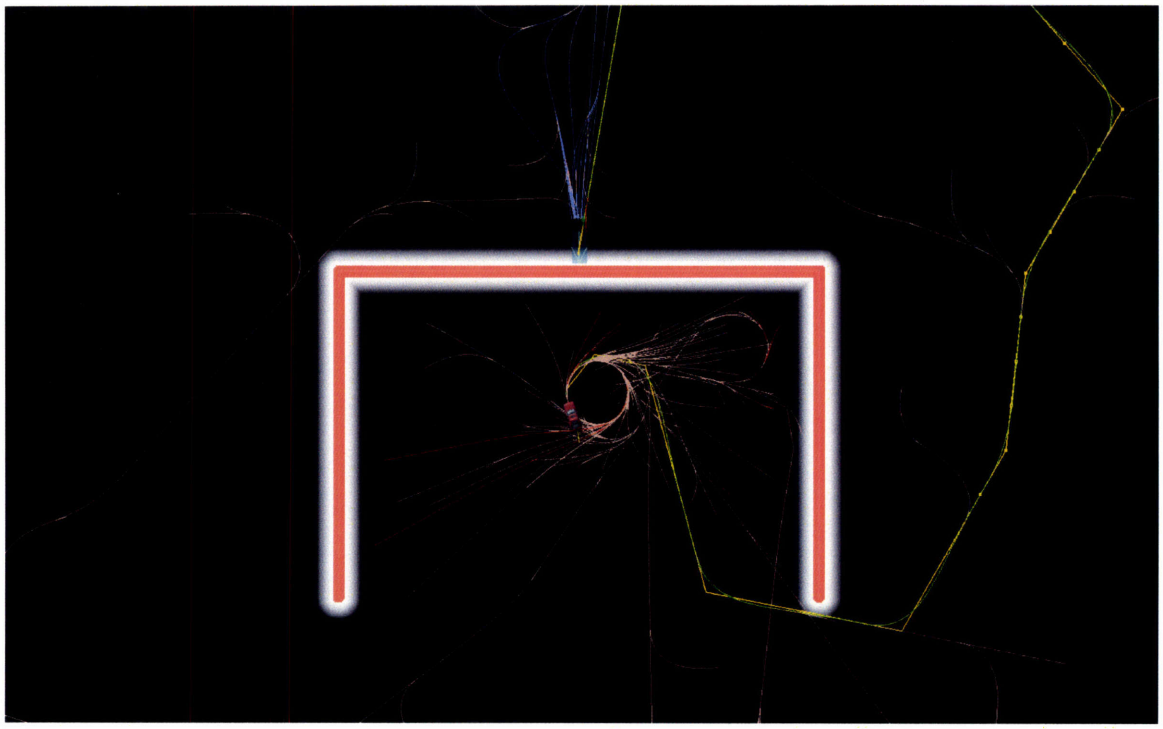


(b) Talos having reached the goal.

Figure 5-27: Planning in a dead end scenario with dead end width set to 50 m.

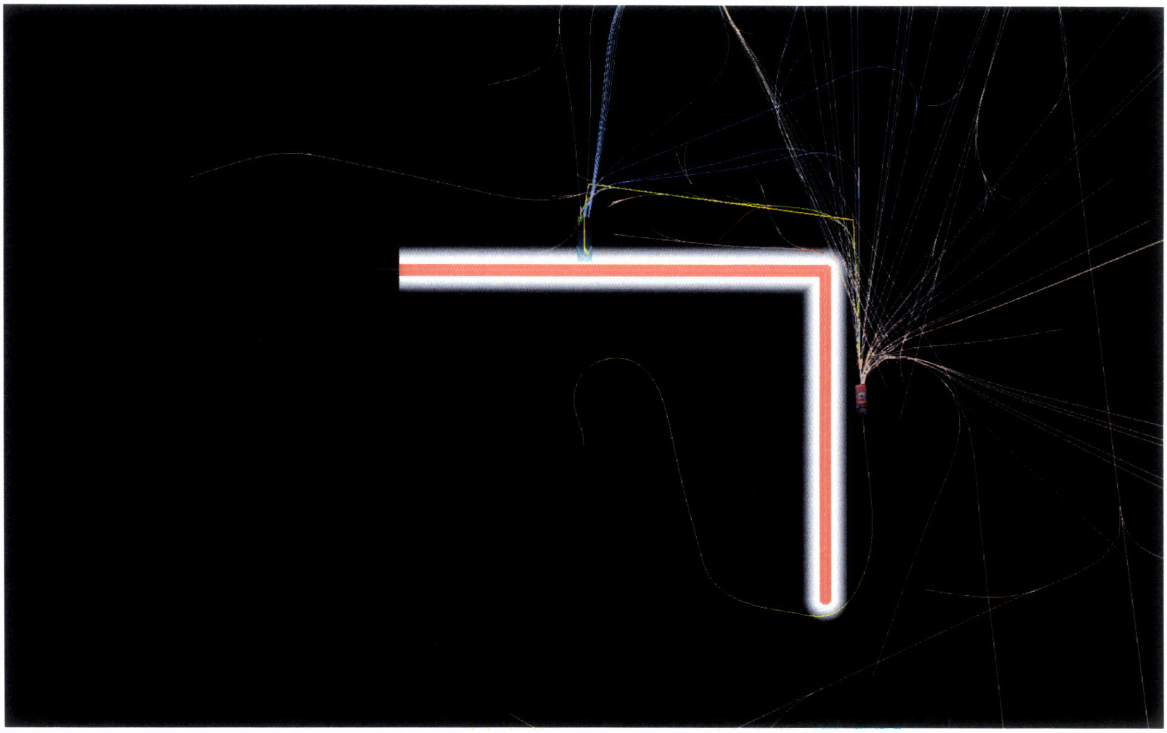


(a) Initial condition of Talos.

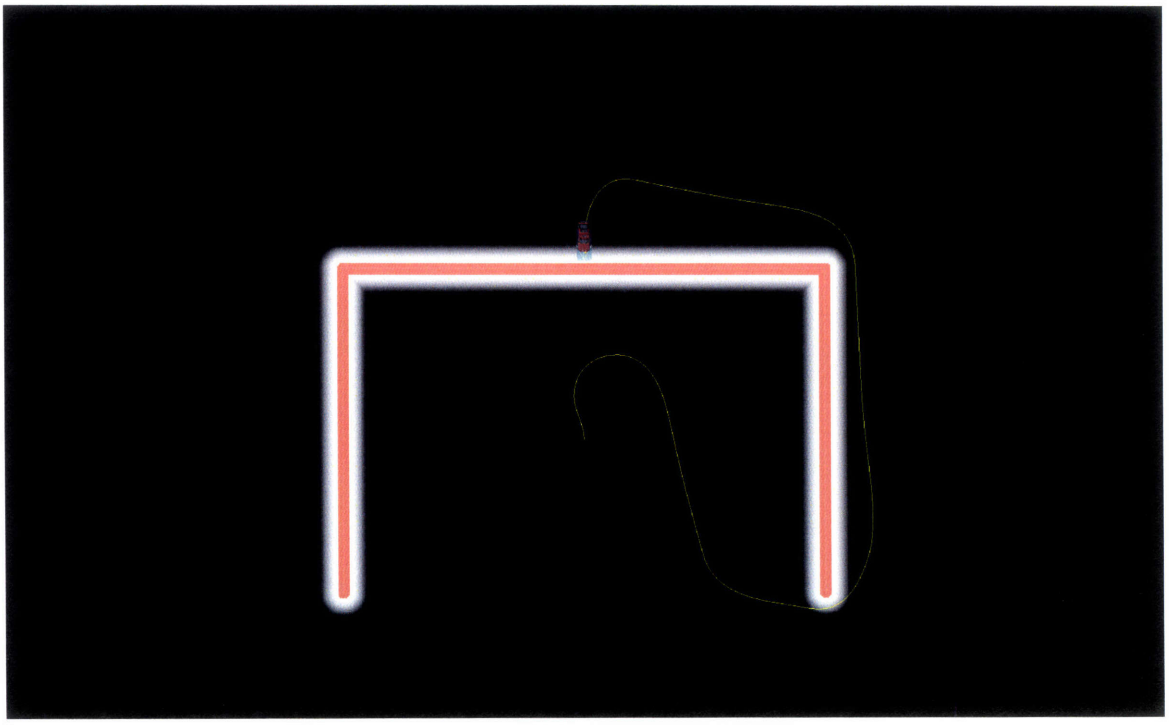


(b) Early in the planning process.

Figure 5-28: Planning in a dead end scenario with dead end width set to 75 m.

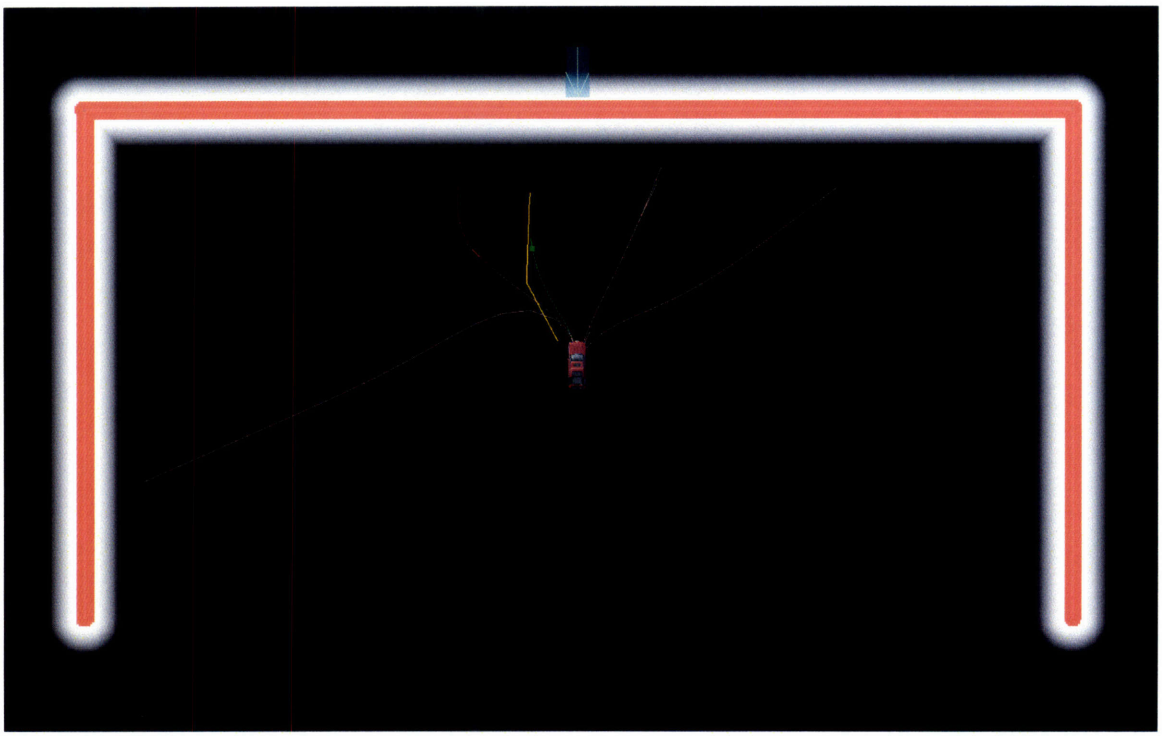


(a) Final stages of the planning process.

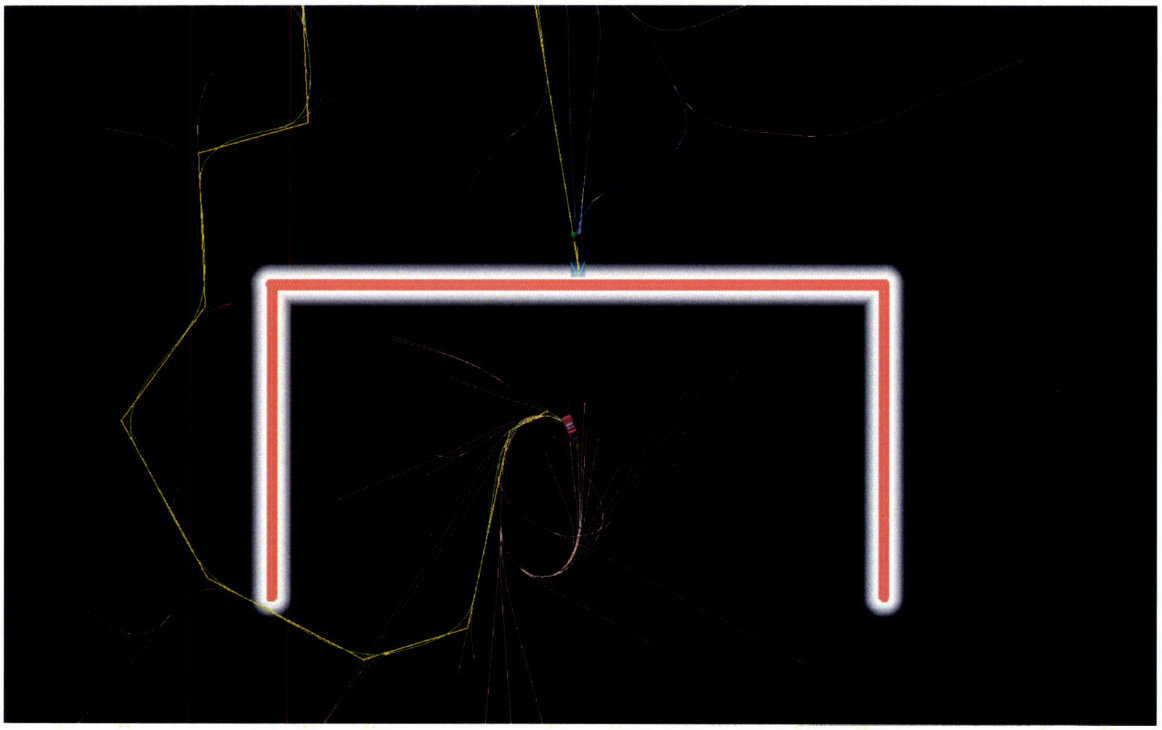


(b) Talos having reached the goal.

Figure 5-29: Planning in a dead end scenario with dead end width set to 75 m.

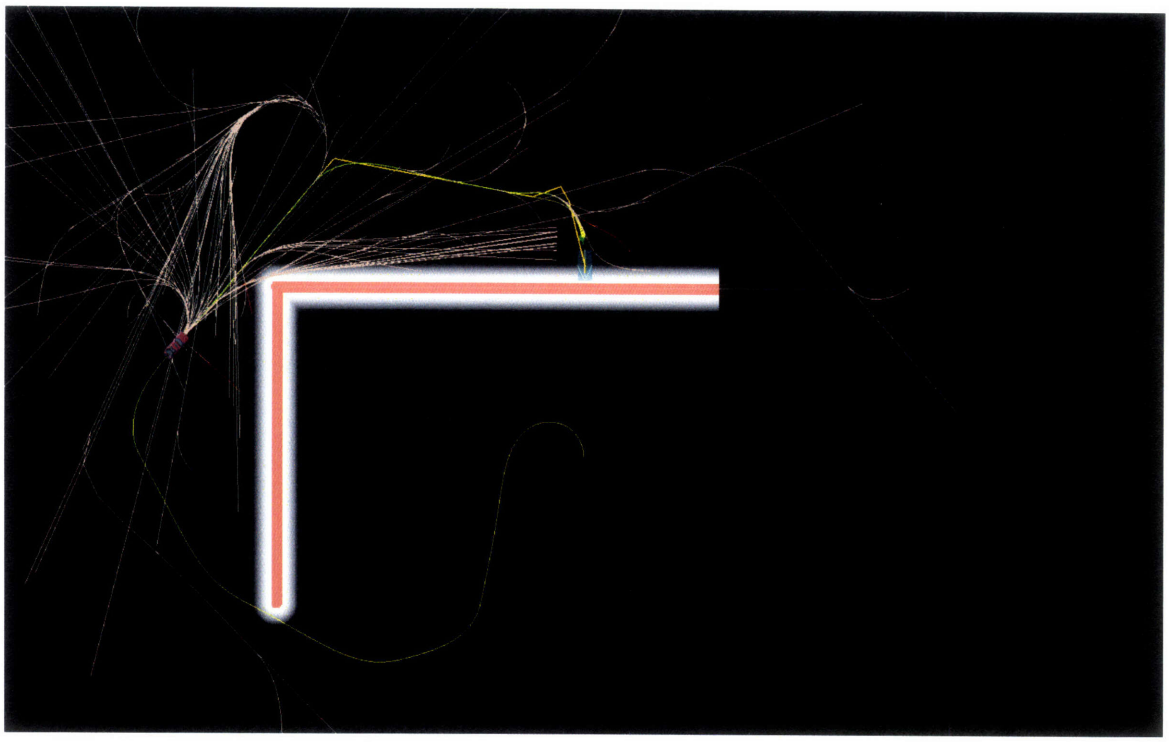


(a) Initial condition of Talos.

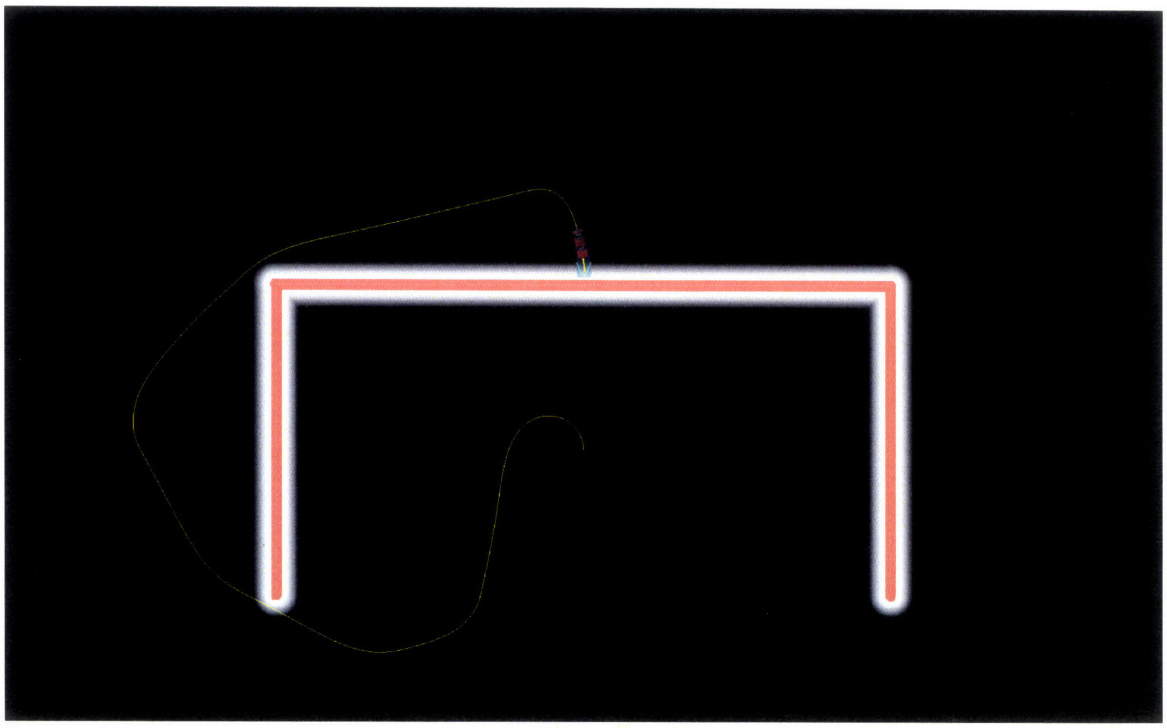


(b) Early in the planning process.

Figure 5-30: Planning in a dead end scenario with dead end width set to 100 m.



(a) Final stages of the planning process.



(b) Talos having reached the goal.

Figure 5-31: Planning in a dead end scenario with dead end width set to 100 m.

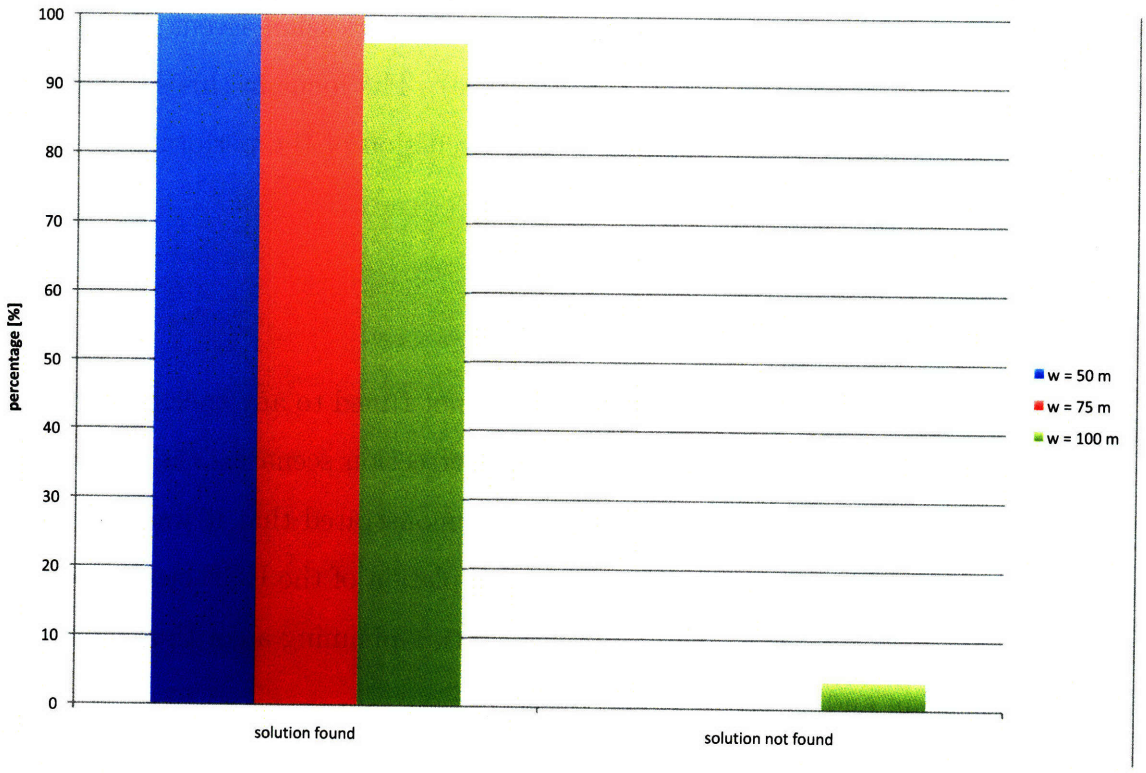


Figure 5-32: Success rate of the algorithm for the dead end scenario for widths of 50 m, 75 m, and 100 m.

5.2 DARPA Urban Challenge Data

The Robust RRT algorithm was tested at the 2007 DARPA Urban Challenge. As implemented on Talos, the motion planner ran on a dual-core 2.33 GHz Intel Xeon processor at 10 Hz. The average number of samples generated was about 700 samples per second. Remember that the algorithm samples the input to the controller and not the input to the vehicle directly, and as a result a single sample could create a trajectory as long as a few seconds. Next, we analyze the performance of the Robust RRT algorithm both during the National Qualification Event (NQE) and during the Urban Challenge Event (UCE, the race). The focus in this section is on the performance of the motion planner. For a discussion of the performance of the whole system during the DUC, refer to [52].

5.2.1 National Qualification Event

During the NQE, the Robust RRT algorithm was not tuned to any specific test area. Furthermore, there were numerous traffic and intersection scenarios that had never been tested before, and yet the motion planner demonstrated that it was capable of handling these situations well. The successful completion of the UCE clearly demonstrated that Robust RRT is a general purpose motion-planning algorithm capable of handling uncertain and dynamic driving scenarios.

Parking

Figure 5-33 shows Talos' parking maneuver inside the zone during the third run of the Area B test. The zone was not paved, and as a result the dust picked up by Talos' wheels resulted in numerous phantom obstacles. These obstacles adversely affected the ability of the motion planning algorithm to find feasible paths. Figure 5-33a shows Talos inside the parking spot. Talos tried to enter the spot from the right first, but due to an inconvenient angle, it failed to reach the goal. The motion planner then directed the car to reverse (notice in the figure the numerous paths with cyan color indicating reverse trajectories), and subsequently to move back in. In this second try,

now coming into the spot straighter, Talos successfully reached the goal.

In Figure 5-33b, Talos is out of the spot and starting to move forward. The motion planner generated a path in reverse that left the vehicle facing towards the goal and in easy position to start moving forward to exit the zone. A human driver would arguably have reversed in the opposite direction when leaving the spot, i.e., downwards in Figure 5-33b. The motion planner tried this maneuver, but however, it became infeasible due to a phantom obstacle as a result of dust. This semi-circular maneuver going updwards in the figure was the best alternative path found. A phantom obstacle can also be seen right next to the car (represented by red blob) in this case, although this time it did not touch the vehicle (albeit extremely close). Notice how the tree features numerous trajectories that cover most of the free space.

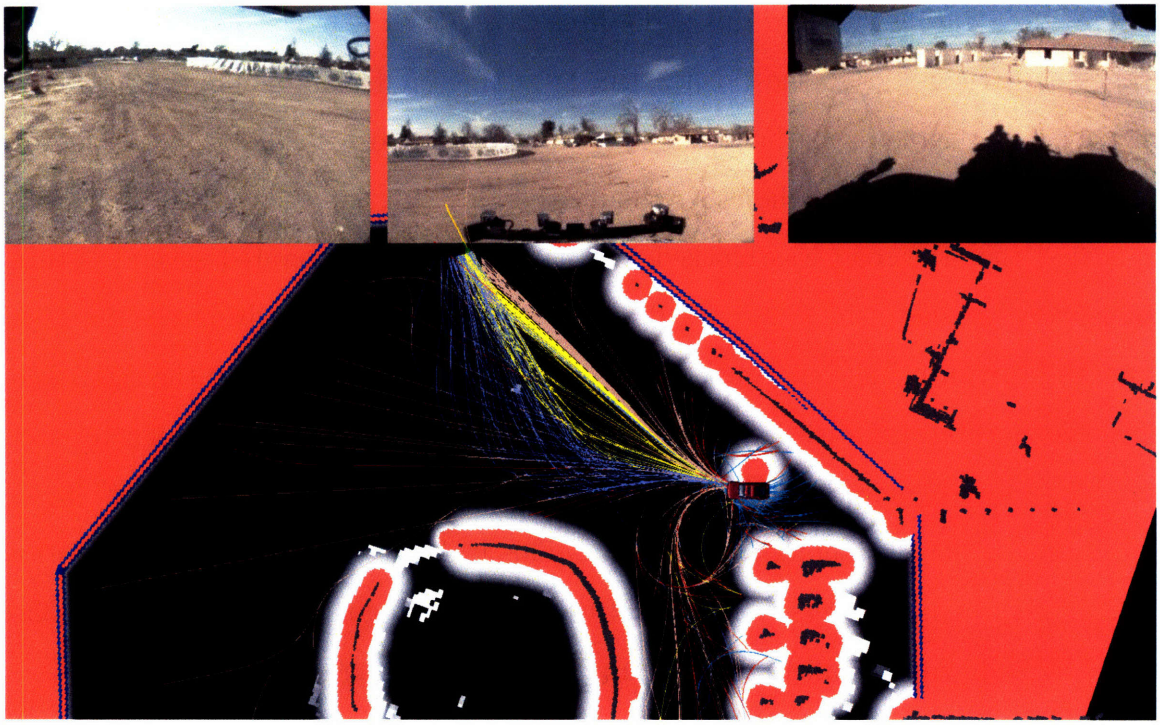
Road Blockage

Figures 5-34 and 5-35 illustrate the behavior o Talos during the second U-turn that was required during the third run of the Area C test. Figure 5-34a shows the location from where Talos started the U-turn. Due to an uneven distribution of the restricted region before the blockage, the vehicle switched to the opposite lane since that location was closer to the goal on the other side of the blockage. As soon as the Navigator indicated that the road was blocked, the motion planner generated several U-turn maneuvers, as shown in Figure 5-34b. Since the vehicle is already in the opposite lane, the U-turn maneuvers consist of backing up with full steering to the right, and then moving forward with full steering to the left. After reversing enough to later be able to avoid going over the curbs during the forward leg of the maneuver, Talos starts moving forward along the new lane. This is depicted in Figure 5-35a. Notice how a rich tree now covers the width of the lane to the goal. Finally, Figure 5-35b shows Talos having completed the U-turn and driving along the lane.

Lane Following

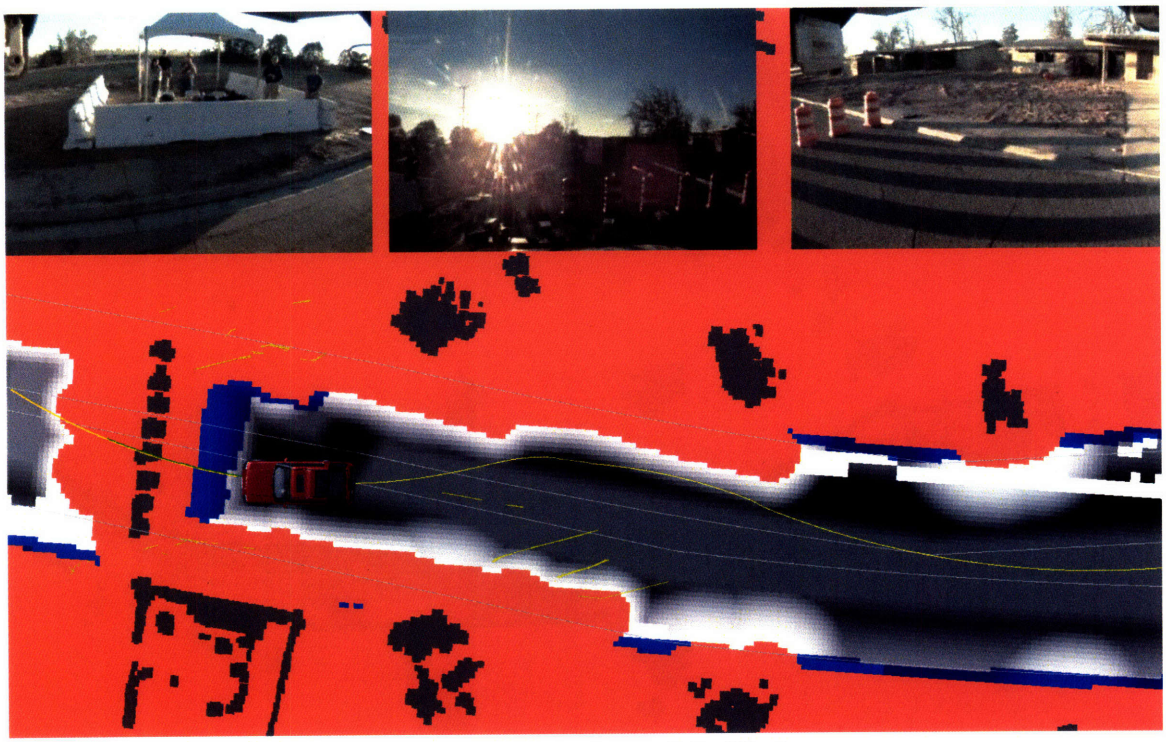
Figure 5-36 shows an instance during the third run of the Area B test of Talos driving along a lane with some parked cars along its side. We can see that the motion planner

(a) Talos successfully parks inside the spot.



(b) Having left the spot, Talos starts driving towards the exit.

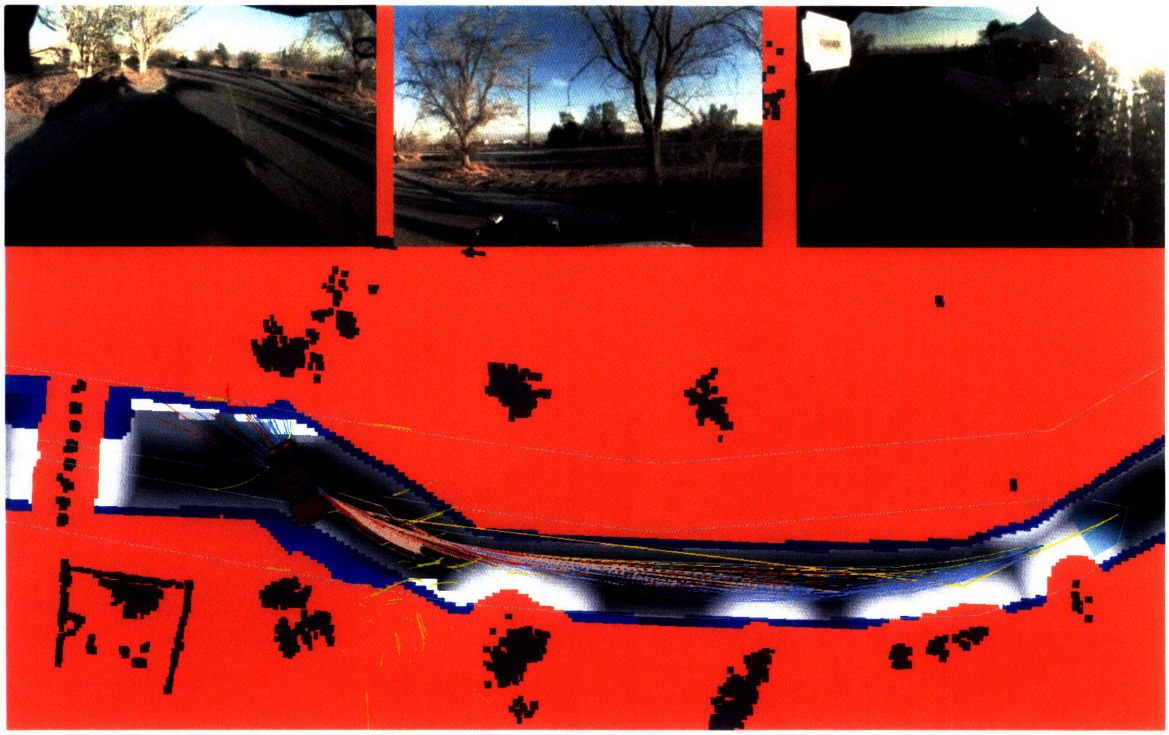
Figure 5-33: Talos parking inside the dirt zone during the third run of the Area B test.



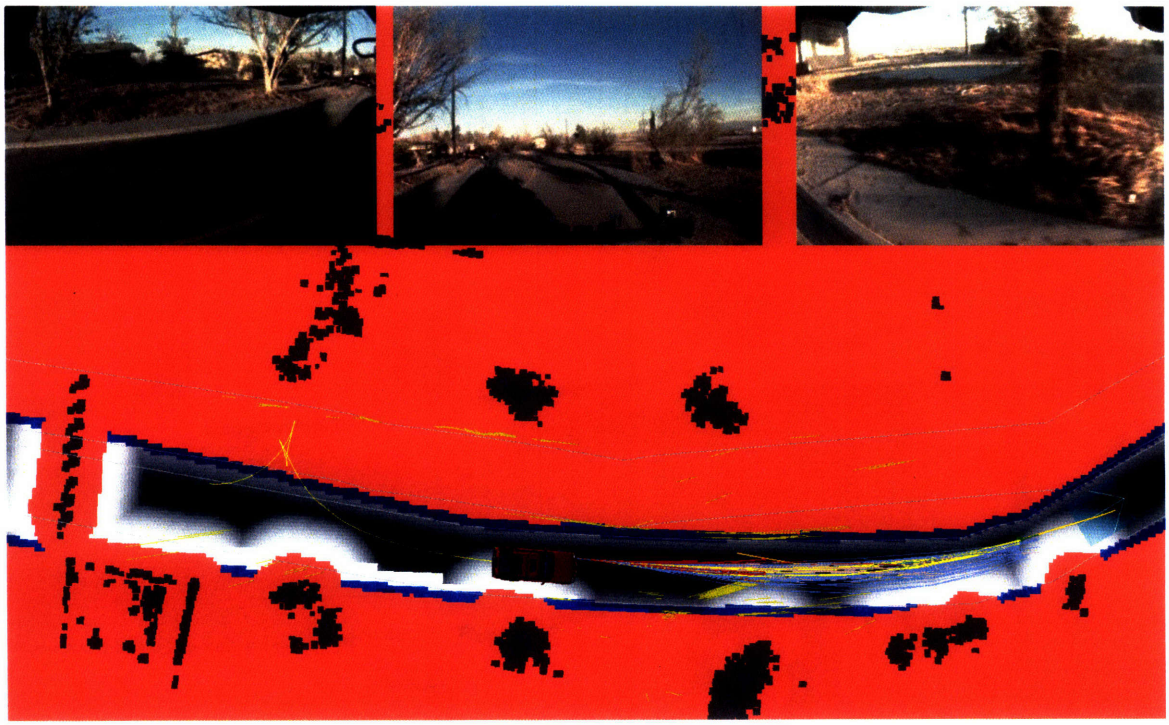
(a) Talos stopped in front of the blockage.

(b) Talos starts executing the U-turn maneuver.

Figure 5-34: Talos performing a U-turn at the second blockage during the third run of the Area B test.



(a) Talos moves forward to the opposite lane.



(b) The U-turn is executed successfully.

Figure 5-35: Talos performing a U-turn at the second blockage during the third run of the Area B test.

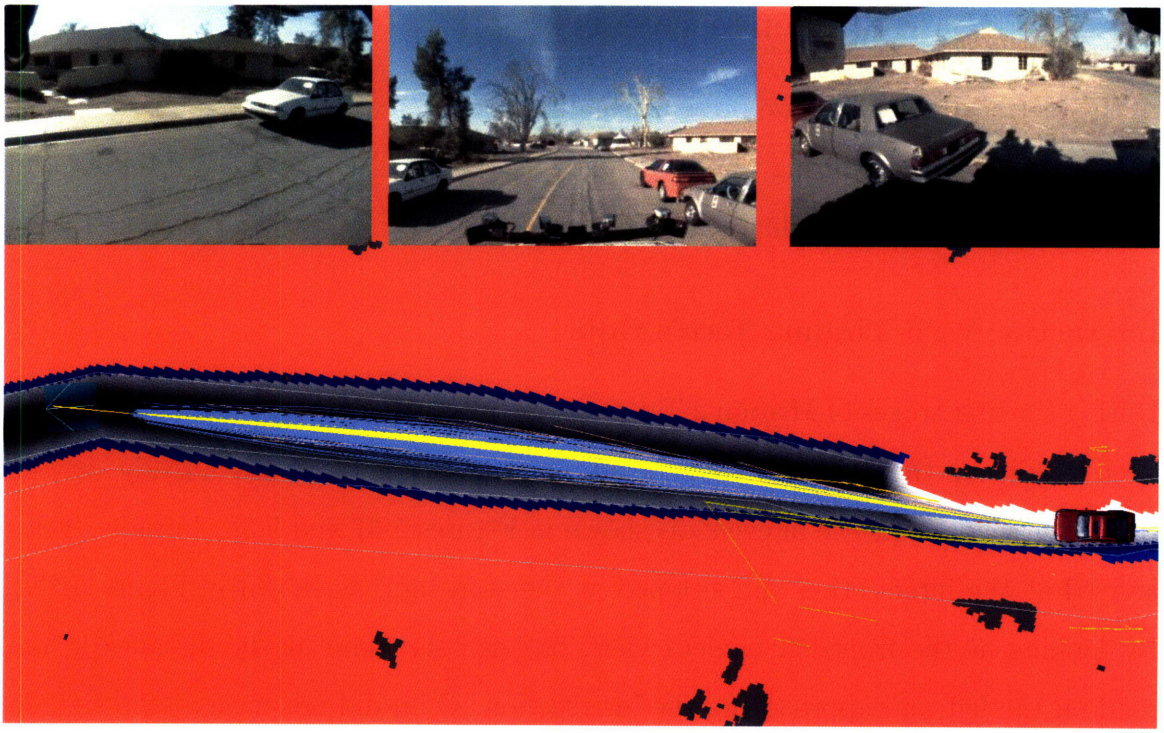


Figure 5-36: Motion planner planning along a lane.

generated a rich tree of smooth trajectories that covers most of the the lane. The Robust RRT algorithm easily plans effectively along lanes.

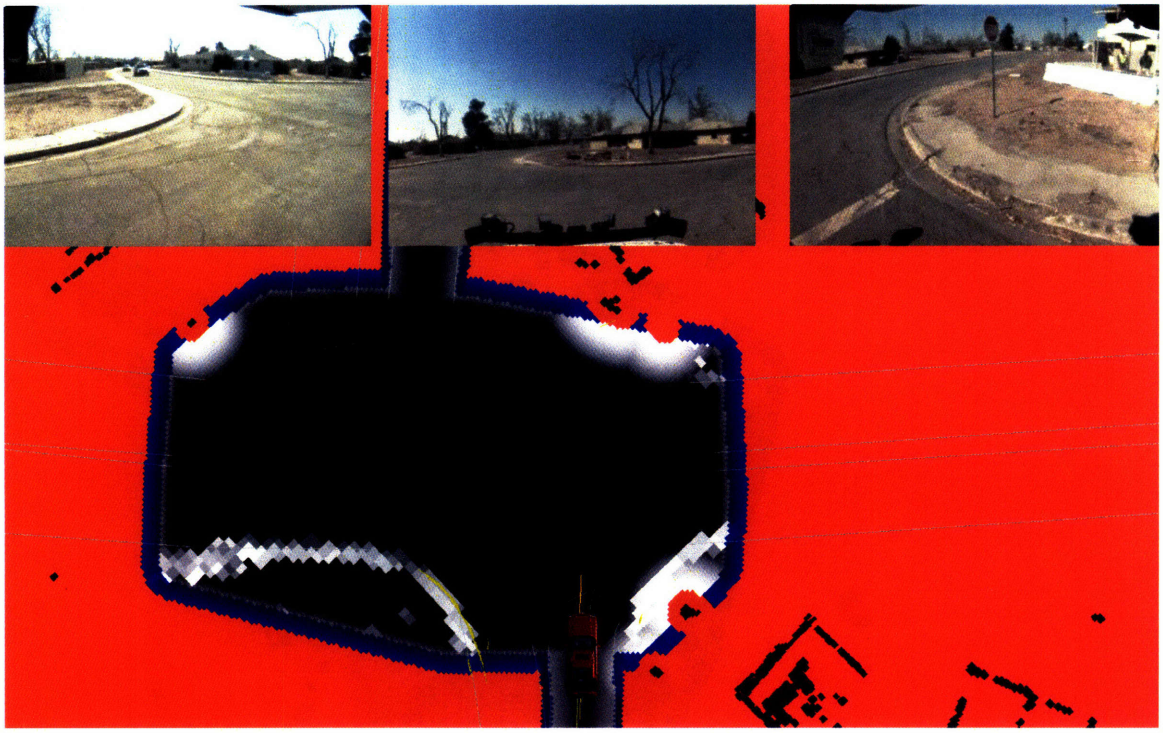
5.2.2 Urban Challenge Event

In the sections that follow we present five different cases where Talos interacted with other robotic vehicles during the UCE. The reactions of Talos from the standpoint of the motion planner are evaluated and discussed.

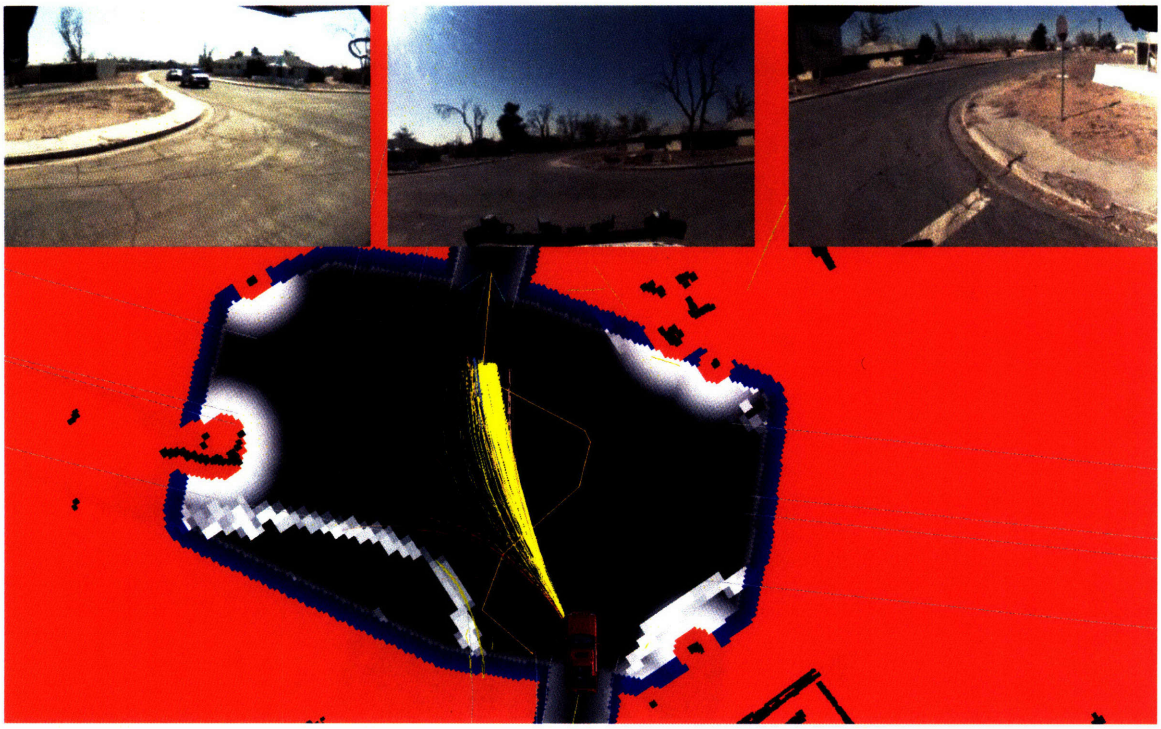
Interaction with Virginia Tech's Odin

During the first stages of Mission 2 Talos successfully avoided a collision with Virginia Tech's Odin as the latter entered an intersection when Talos was already driving through it. The sequence of events is shown in Figures 5-37 and 5-38. Initially, Talos arrived at the intersection and stopped at the stop sign, as shown in Figure 5-37a. By looking at the front-left camera view, one can see Odin at the distance in the road on the left and approaching the intersection. Odin did not have a stop sign on its lane but nonetheless it stopped when it reached the intersection. Odin remained motionless for about 3 s before starting moving again. With the intersection clear, and all vehicles stopped, Talos assumed precedence over Odin and started driving through the intersection. Just as Talos started however, Odin also started driving, heading towards the same lane as Talos, which is shown in Figure 5-37b. As Odin kept moving forward and started turning to its left to enter the lane, Odin completely blocks Talos' path, as seen in Figure 5-38a where Odin is represented by the collection of grey squares in front of the lane.

With an obstacle right in front of Talos, the motion planner tried to find a path close to the goal but at the same time that is a safe distance away from obstacles. The result was the path taking advantage of the narrow free space between Odin and the sidewalk to the left of Talos. Following this path, Talos is guided safely away from Odin and stops for about 1 second before moving again. Finally, as Odin passes through the intersection and drives away, the motion planner will guide the vehicle back to its original route through the path with the least cost, which is one of the paths with efficient cost (shown in yellow in Figure 5-38b).



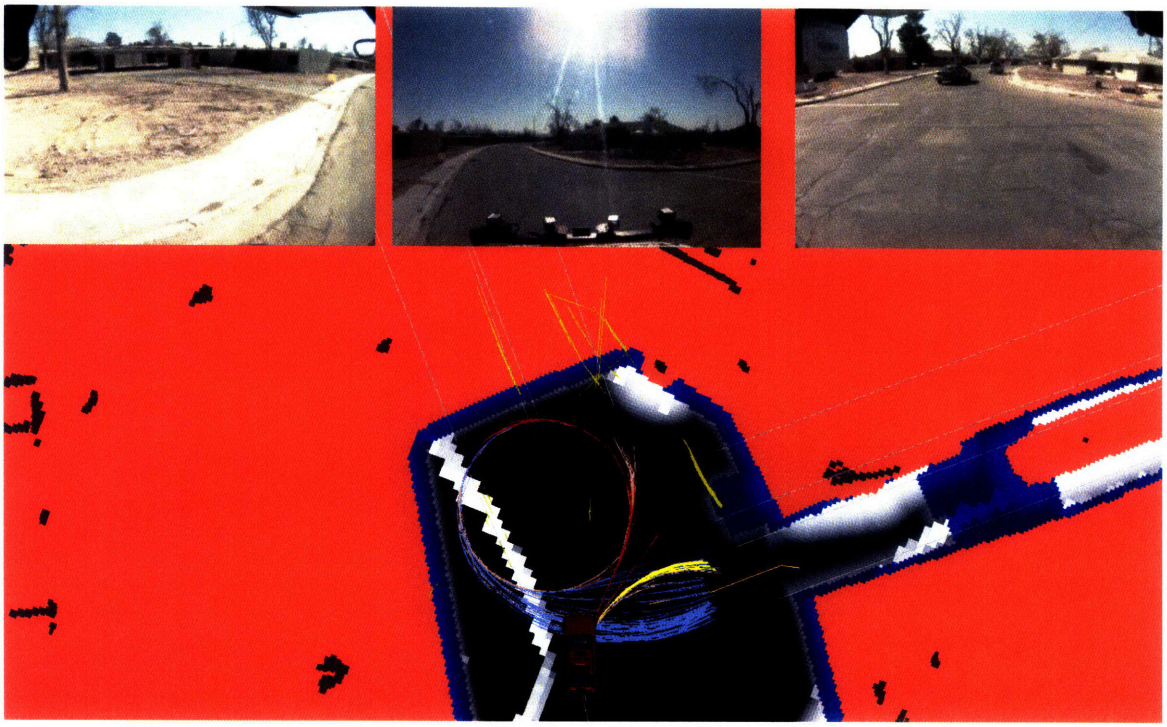
(a) Talos has precedence to proceed.



(b) Odin enters the intersection with Talos there.

Figure 5-37: Talos interacts with Virginia Tech's Odin at an intersection.

(a) The motion planner replans safely.



(b) Talos gets back on route.

Figure 5-38: Talos successfully avoids Odin.

Encounter of a Gate Blockage

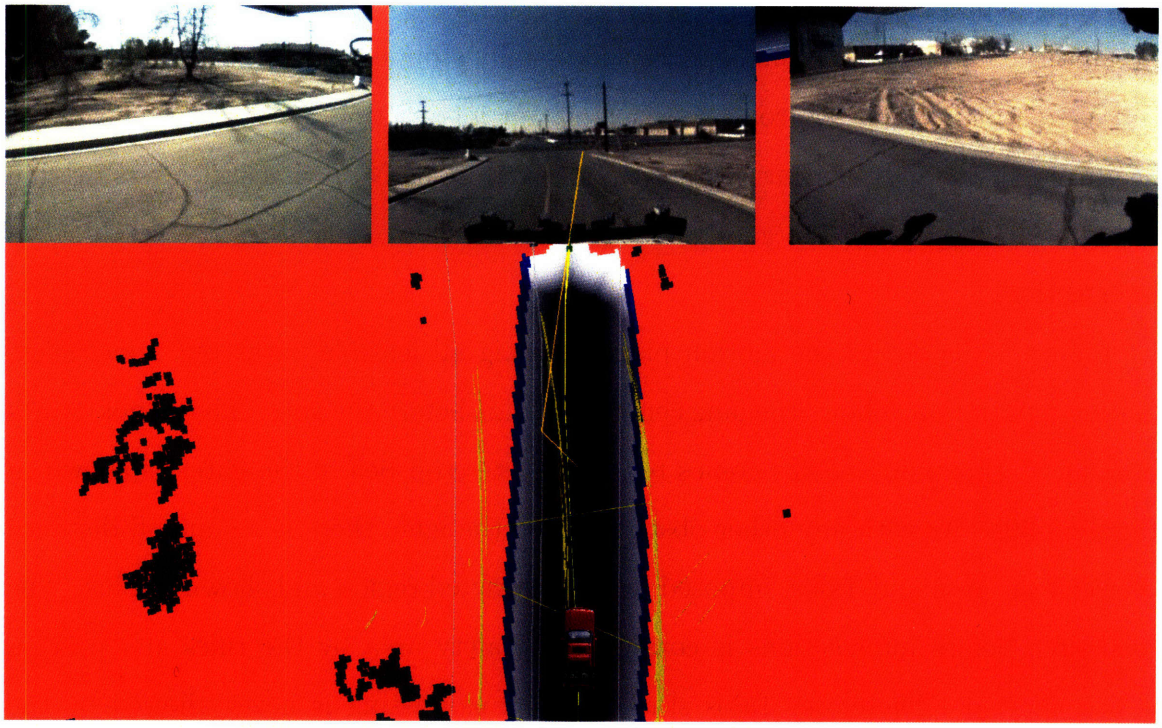
Around the middle of Mission 3, Talos successfully stopped in front of an unexpected tubular gate blocking the road right before an intersection. Notable events while dealing with the gate appear in Figures 5-39 and 5-40. Note that Talos was driving at about 20 mph when the gate was perceived on the road, just before reaching an intersection.

Figure 5-39a shows the motion planner losing all feasible paths but nonetheless finding a path that follows the lane center and ends before the gate. The vehicle starts braking at this point. Talos comes to a complete stop before the gate, as shown in Figure 5-39b. As with any other obstacle along the lane, there is a standoff distance between the gate and Talos, imposed by the restricted region and shown in blue. The road area before intersections is considered a safety area by the rules and passing is not allowed. Therefore, Talos stays stopped as passing is never allowed by the Navigator.

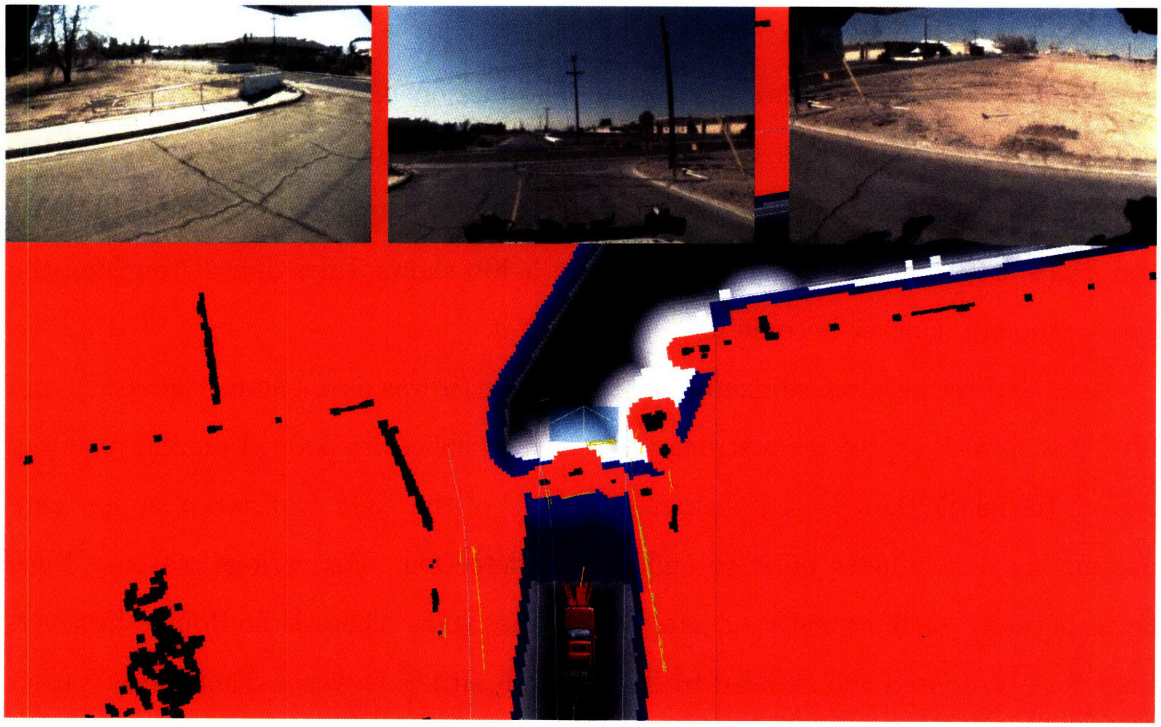
About three minutes later, as seen in Figure 5-40a, the gate is lost by the laser sensors and the goal becomes reachable. The motion planner readily finds several paths to the goal and Talos starts moving forward. Suddenly, the gate is perceived again by the lasers, at which point the motion planner commands emergency braking and Talos stops just before the gate. Figure 5-40b shows Talos almost touching the gate, although it never become in contact with it.

Note that the gate was not supposed to be there (it was blown across the road), and thus this episode represents a strong validation of the entire perception and planning system to handle unexpected obstacles. Area C of the NQE featured a similar gate but having 3 stop signs on it to enhance its detection. For various reasons, Talos never actually encountered that gate during the NQE. There is no doubt, however, that the gate sensed and avoided by Talos in this situation was a significantly harder test.

The gate was eventually removed by DARPA while Talos was paused, and then Talos proceeded along the route.

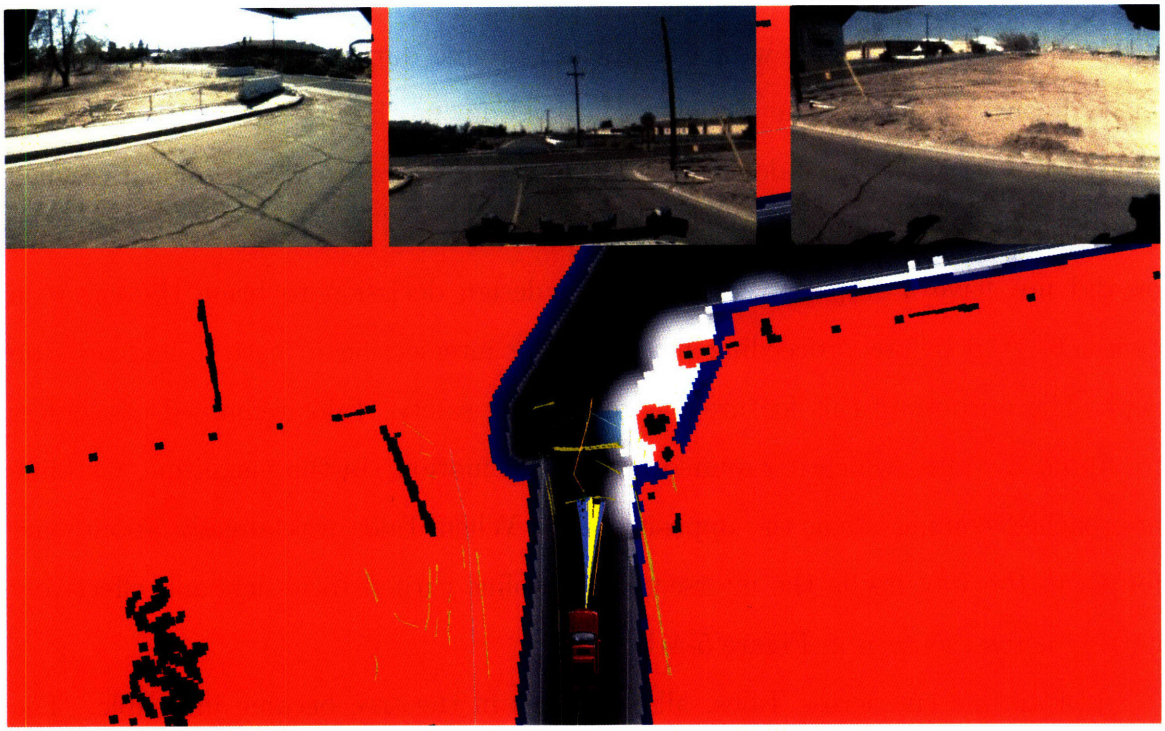


(a) The gate is seen and the motion planner reacts.



(b) Talos keeps a safe distance away from the gate.

Figure 5-39: Talos successfully brakes in front of an unexpected gate in the middle of the road.



(a) The lasers stop detecting the gate and Talos moves forward.

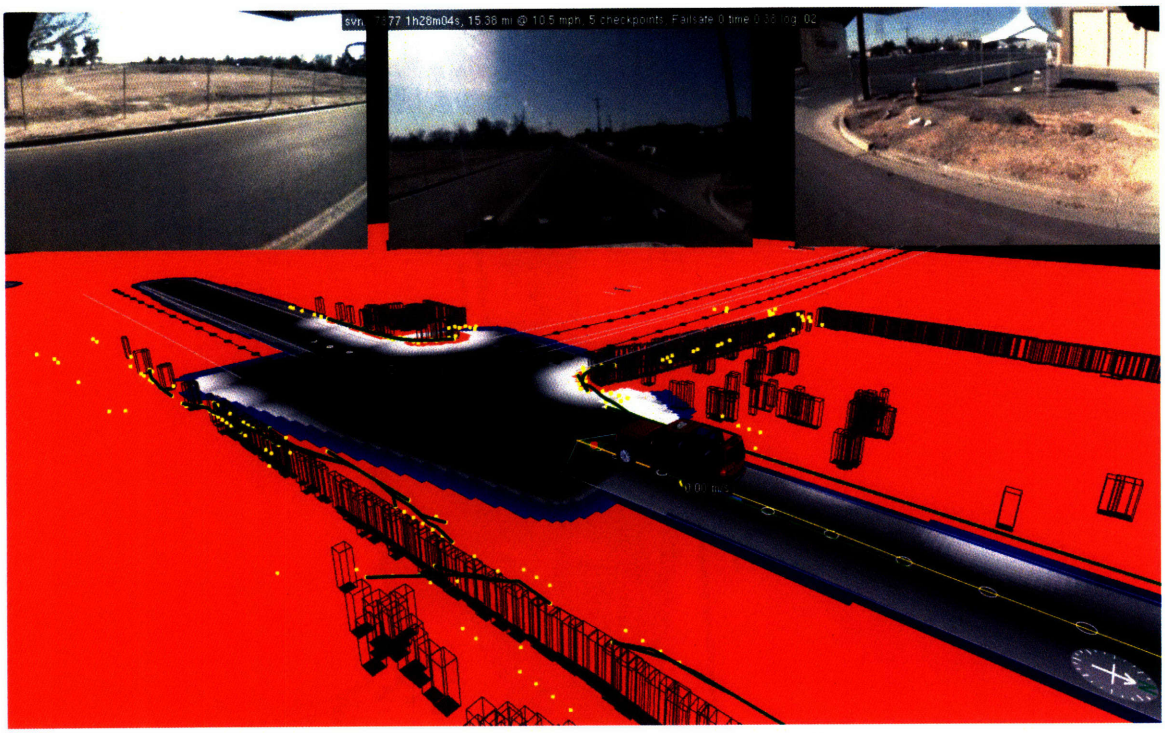
(b) The gate is detected again resulting in emergency braking.

Figure 5-40: Talos successfully brakes in front of an unexpected gate in the middle of the road.

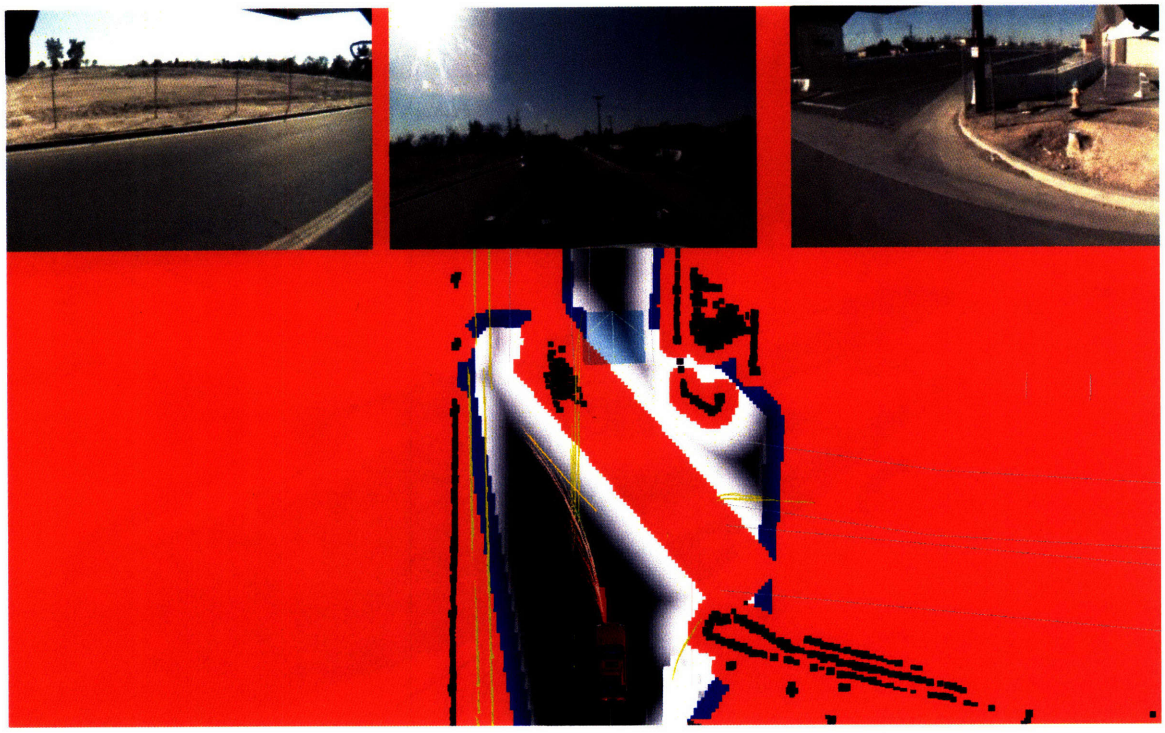
First Interaction with CarOLO's Caroline

Talos had two interactions with CarOLO's Caroline during the DUC, both taking place during Mission 1. The sequence of events during the first interaction is shown in Figures 5-41 and 5-42. Talos was driving along a two-way road and when it reached the intersection it stopped, depicted in Figure 5-41a. The lane that Talos was driving on did not have a stop sign, but it was a design decission to bring the car to a complete stop before proceeding further at an intersection in case it had no stop sign. The purpose of this feature was to increase safety. Talos remained motionless for about 3 s after which it resumed its forward motion. In the meantime, Caroline approached coming along the opposite lane. When Talos was already about one-fourth of the way inside the intersection, Caroline turned to its left and effectively cutting across, as seen in Figure 5-41b.

As Caroline cut across Talos, all the paths in the tree became infeasible. The motion planner then begins to find paths in the open space to the left of Caroline. Caroline kept turning, and at one instant it becomes regarded as a static obstacle. Without the extra infeasible region appended along the direction of motion of moving obstacles to account for their future position, the space in front of Talos became free. As shown in Figure 5-42a, the motion planner readily builds a rich tree along the intersection and the lane in front, and the vehicle proceed in that direction as supposedly there is no dynamic obstacle. About one second later, Caroline becomes regarded as a moving obstacle again. Being too close to Caroline this time, the motion planner commands emergency braking just before DARPA operators issued a Pause command to both vehicles. This is shown in Figure 5-42b. Finally, we give an additional view of the situation in Figure 5-43.



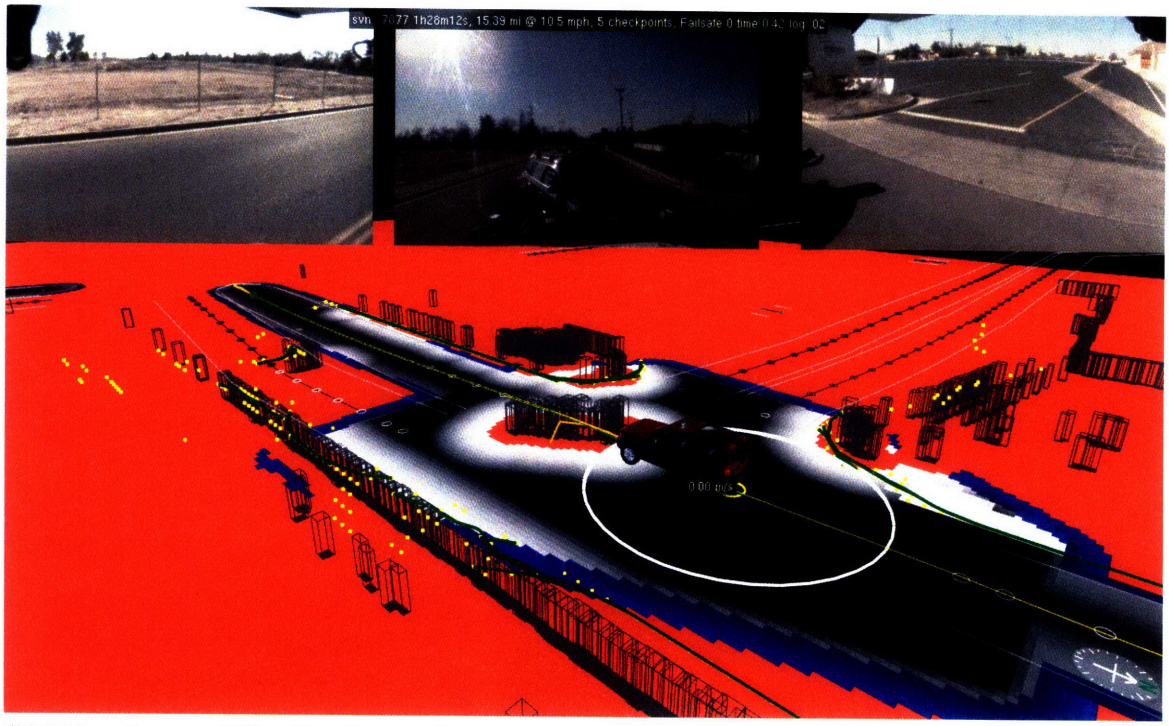
(a) Talos stops upon entering the intersection.



(b) Talos detects the moving object and plans around it.

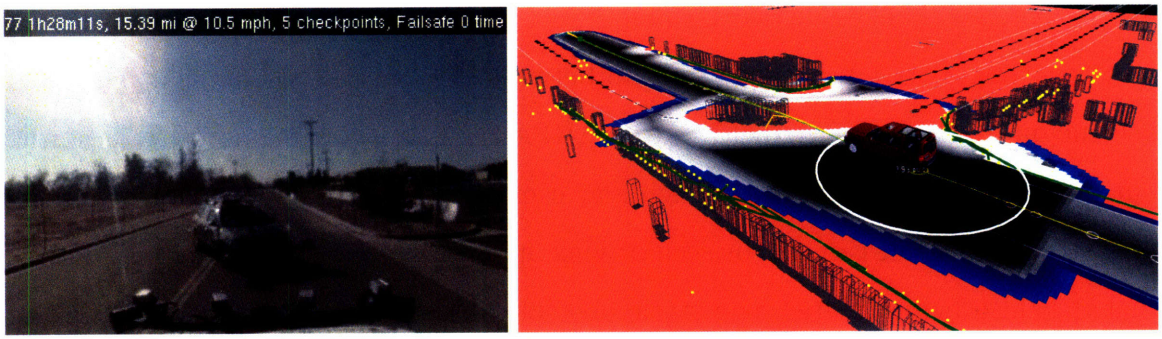
Figure 5-41: First interaction with CarOLO's Caroline.

(a) The motion planner commands an emergency stop.

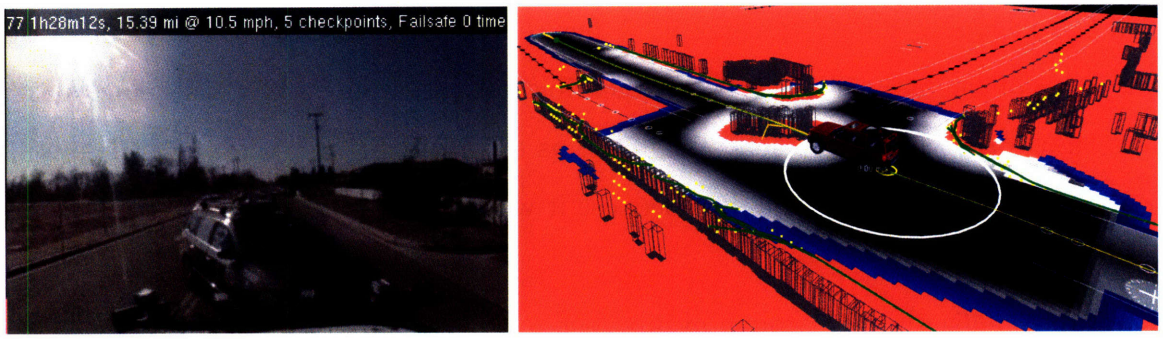


(b) Talos stops and Caroline does no longer appear to be moving and is regarded as a static obstacle.

Figure 5-42: First interaction with CarOLO's Caroline



(a) Caroline is approaching.



(b) Caroline cuts across Talos.

Figure 5-43: Reaction of the motion planner to Caroline's cutting across showing views from the front-center camera in Talos' roof.

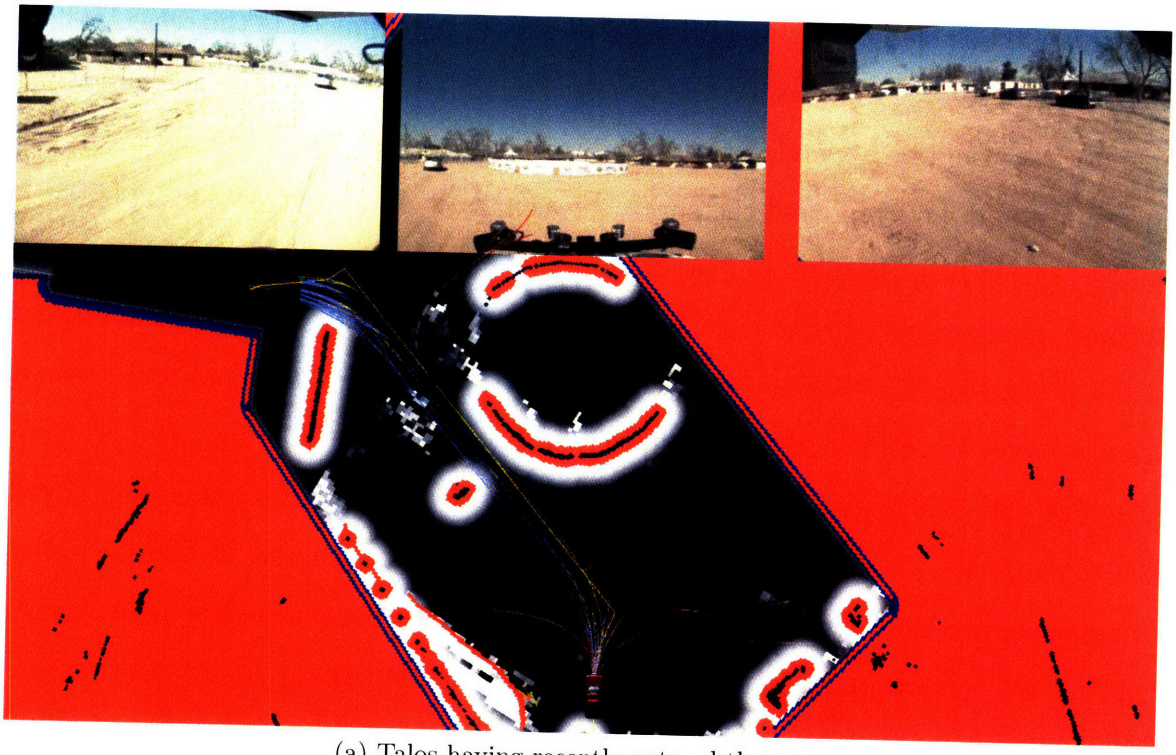
Second Interaction with Caroline

The second interaction with Caroline occurred inside a zone and resulted in both vehicles coming into contact with each other. The sequence of events dealing with the incident appears in Figures 5-44 – 5-46. Initially, Talos was driving through the zone and was going to pass between a circular fence with DARPA Urban Challenge advertising and a DARPA manned car, as shown in Figure 5-44a. However, this car was being regarded as a static obstacle due to its slow speed. A few seconds later, the motion planner changed its decision and directed Talos to the left of the DARPA car, as seen in Figure 5-44b because the gap between the car and the circular fence had become quite narrow. Nonetheless, the DARPA car was still being detected as static, but of course, its location was changing at every iteration as it was actually moving very slowly. Caroline can also be seen driving apparently toward the zone exit. Notice that it is also being treated as a static obstacle, without any extra infeasible region along its direction of motion. Figure 5-45a shows Talos still advancing to the exit and Caroline now almost blocking it. A few seconds later, Caroline covered the exit and most of the paths in the tree are infeasible. The motion planner still found a few feasible trajectories that reached the exit, as shown in Figure 5-45b. Caroline still kept driving in a straight line, but because it was slow, it was still being regarded as a static obstacle.

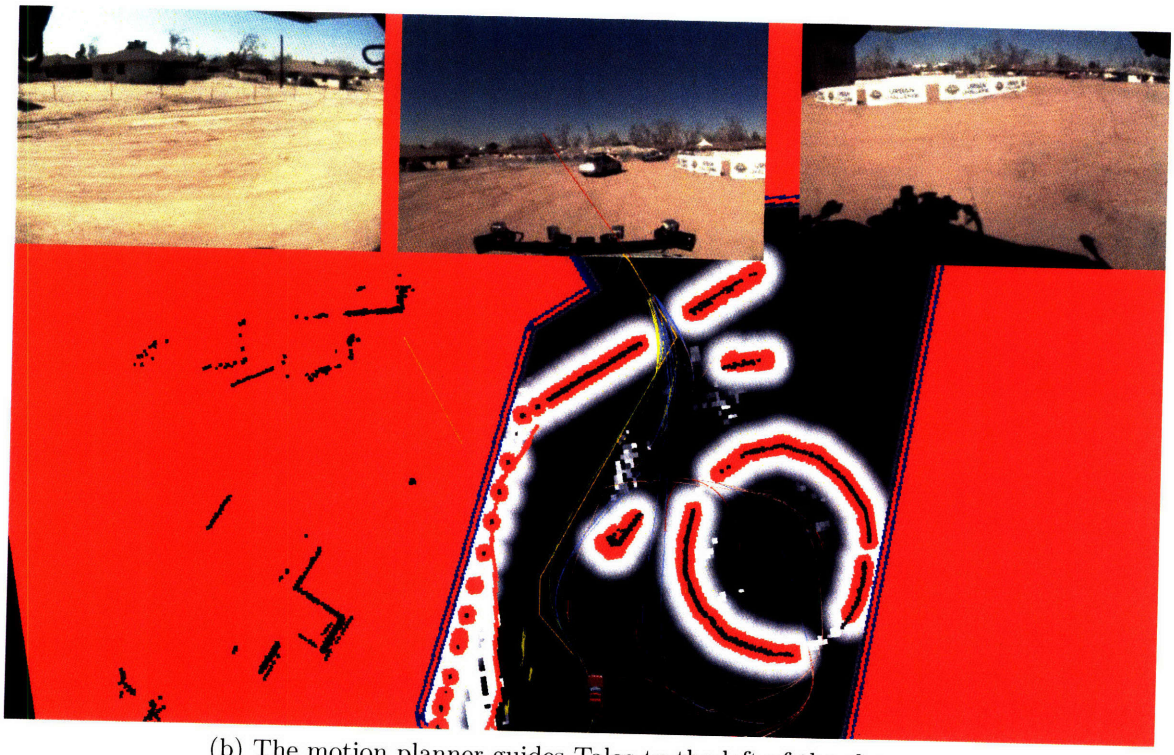
Figure 5-46a shows Caroline even closer to Talos, and it can be guessed that if both vehicles continued on their current paths, the probability of a collision would be very high. By this time, the planner can not find a path to the goal, but guides the vehicle as close as possible to it through a path between Caroline and the fence on Talos' left. At this point, for some reason, Caroline appears to accelerate slightly while still driving in a straight line. Talos kept moving diagonally forward and to its left. However, since Caroline is not being considered a moving obstacle, the space between both cars never became infeasible until both cars were virtually in contact with each other. At this point, the motion planner commanded emergency braking, but it was too late to stop in time. Figure 5-46b shows both cars with their front

bumpers in contact.

It should be noted that this collision would not have happened in the case Caroline would have been regarded as a moving obstacle. This is because the extra infeasible region that would be appended along its direction of motion to account for its future location would have rendered the space between Talos and Caroline infeasible much sooner. Given this, the motion planner would have searched for and found other paths through the free space in the zone. Similarly, this collision would have not happened either if Caroline had been a static obstacle. In this case, the motion planner would have planned a path around the obstacle, as it was constantly doing. The problem was that the free space was becoming narrower at each iteration, and once it did become infeasible, it was too late for the motion planner to plan an alternative route. In addition, the fact that Talos was choked against the fence on the left made it very hard for the motion planner to find alternative paths that could go around Caroline to Talos' right.

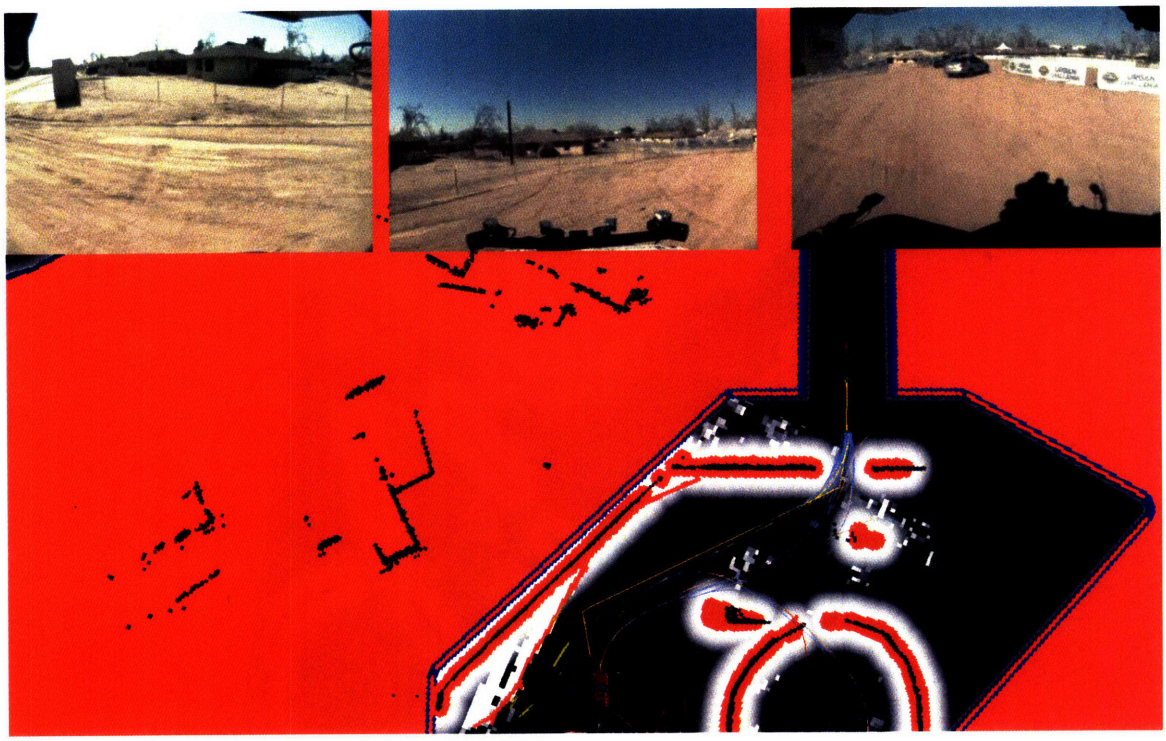


(a) Talos having recently entered the zone.



(b) The motion planner guides Talos to the left of the chase car.

Figure 5-44: Talos approaches Caroline in a zone.

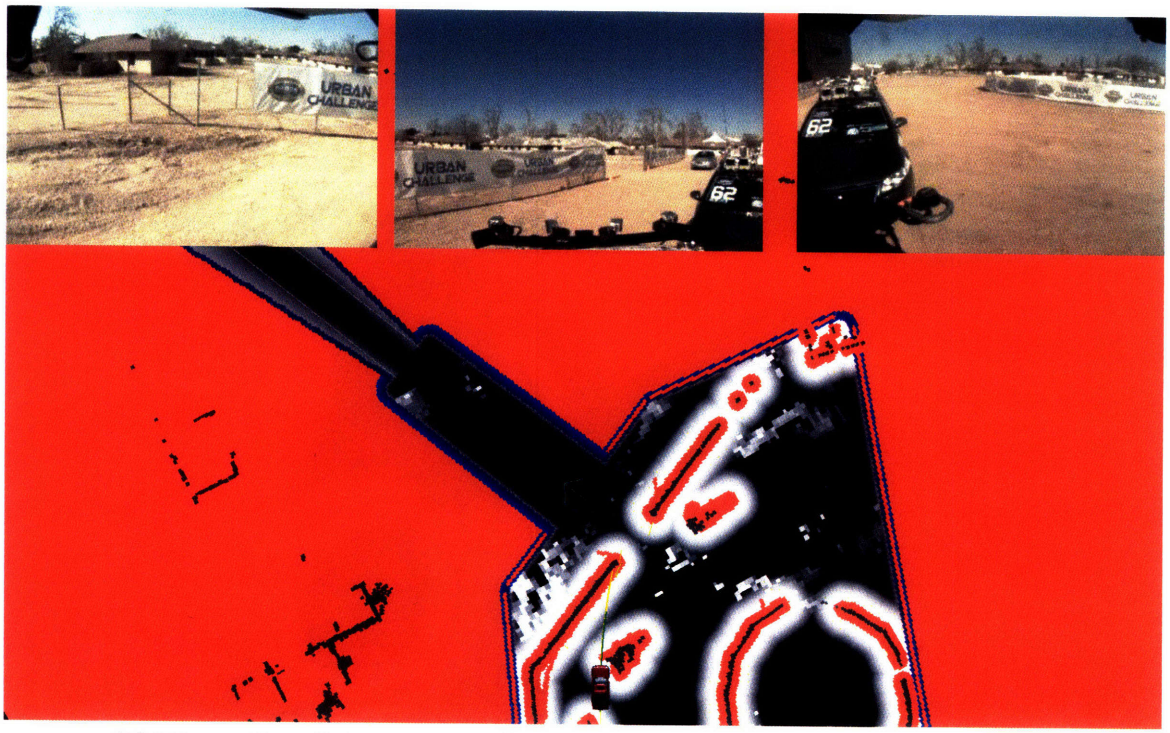


(a) Talos keeps on its path and Caroline approaches parallel to the fence.

(b) Caroline keeps driving in a straight line parallel to the fence.

Figure 5-45: Caroline chokes Talos against a fence inside the zone.

(a) Caroline still does not turn to an open space and chokes Talos against the fence.



(b) The motion planner commands emergency braking but Caroline kept driving.

Figure 5-46: Caroline and Talos make contact in a minor collision.

Interaction with Cornell's Skynet

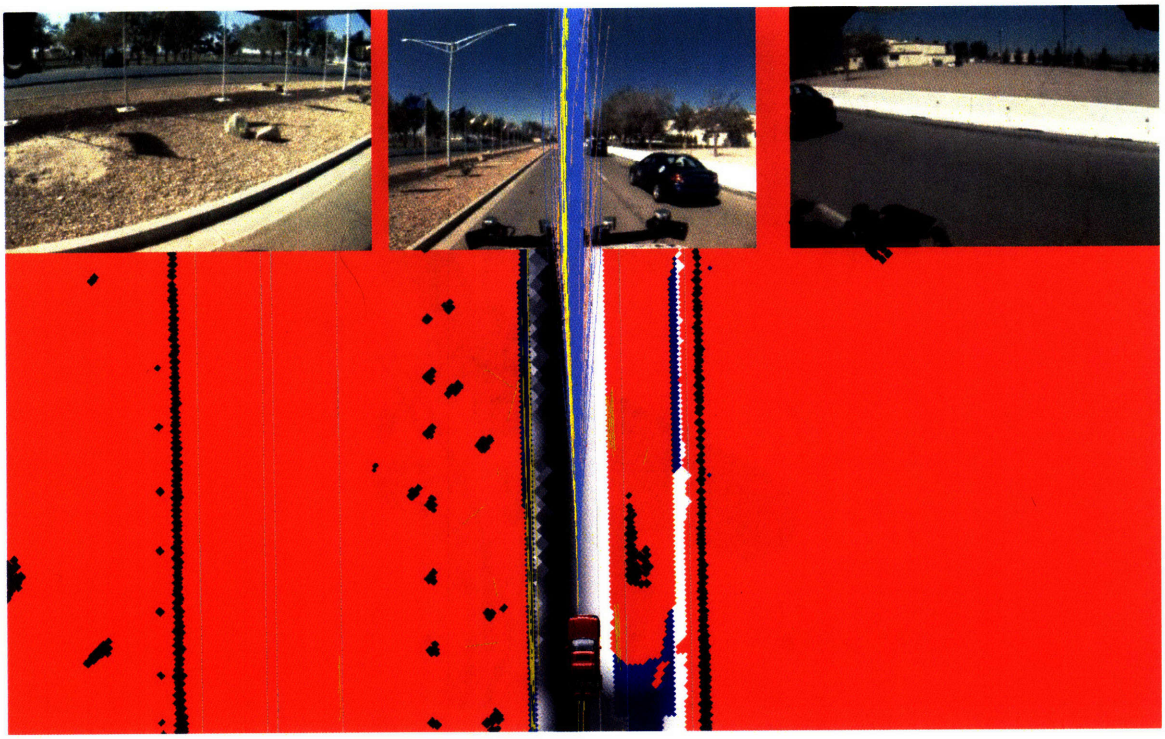
Talos' minor collision with Cornell's vehicle, Skynet, was another notable incident during the UCE, and is illustrated in Figures 5-47–5-49. Initially, Talos was driving along the left lane in a two-lane road with the intent of passing Skynet's chase car, which was following Skynet as both of them were traveling considerably slower than Talos' speed of about 20 mph along the straight road. Figure 5-47a shows a view of this situation. When Talos was about to pass the DARPA car, a row of cones along Talos's lane required merging into the lane on the right. Talos then slowed down and queued behind Skynet's chase car. It turned out that the location of the goal did not match reality well, being located further forward than the stop line indicated, as shown in Figure 5-47b. As a result, there was enough space between Skynet and its chase car for Talos to fit in, and the motion planner directed Talos to the goal. After stopping at the stop sign, the goal switched to the next waypoint in front of (the then stationary) Skynet.

After the goal switched, the motion planner readily found several trajectories that reached the goal by going around Skynet, which was stopped and therefore regarded as a static obstacle. The rich tree can be seen in Figure 5-48a, where it can be noticed that the most cost-efficient trajectories, shown in yellow, are the ones furthest to the left, and have the greatest clearance with respect to Skynet. In fact, although hard to notice, if we look at the predicted trajectory of Talos in green, we can see that it is one of the farthest to the left among those in yellow. This behavior shows the ability of the Robust RRT algorithm to keep the vehicle at a safe distance from obstacles. Figure 5-48b illustrates Talos further down the intersection. The most interesting thing to notice, however, is that Skynet has started to move, albeit very slowly. Similar to previous incidents, Skynet's speed was below Talos' tolerance of 3 m/s to treat it as a moving obstacle, and was therefore being regarded as static, without any projection about future position.

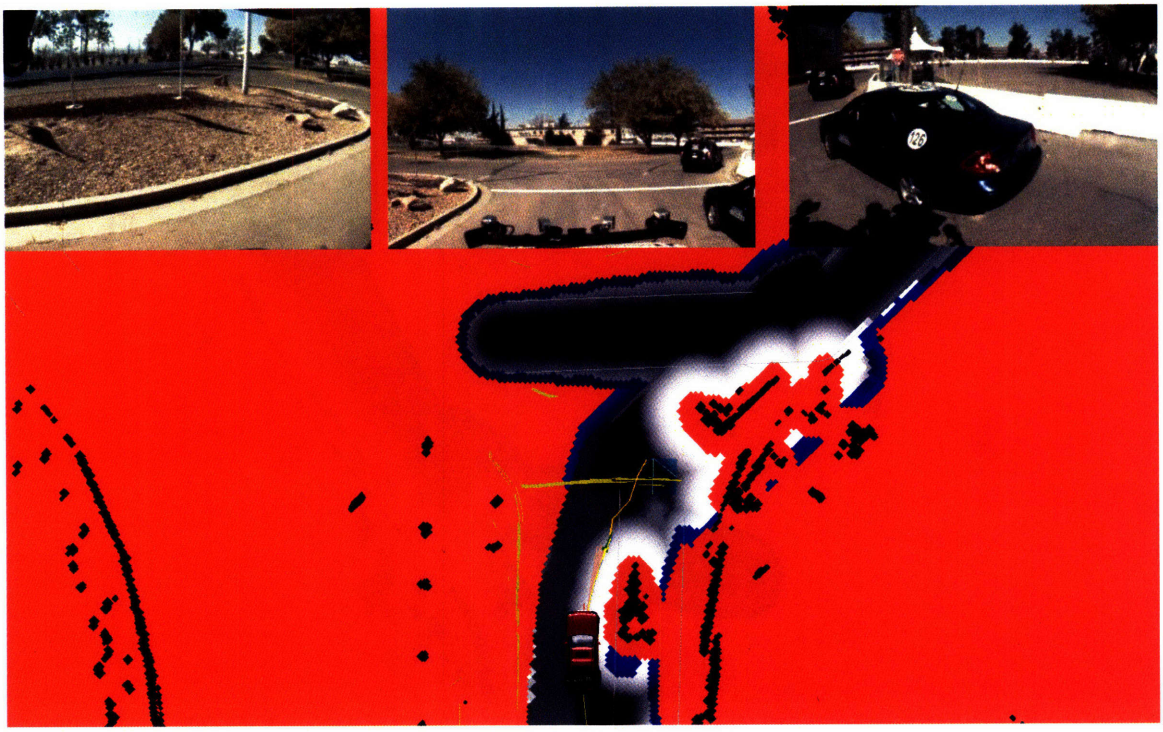
Skynet being considered as a static obstacle but moving forward in every iteration caused Talos to get choked against the left lane boundary, as can be seen in Figure 5-

49a. We can see in the figure that Talos was as far left as possible from the obstacle as it be could based on the perceived environment. There was a slight mismatch between what was being perceived and reality, as can be appreciated by comparing the location of the left curbs in the view from the center-left camera and their location as rendered in the visualization application. As Skynet kept moving slowly and kept being detected as a static obstacle not rendering the region in front of it as infeasible, Talos continued to get choked. The space between the front of Skynet and the left lane boundary was decreasing, but it was nonetheless regarded as feasible. Finally, both vehicles collided, as shown in Figure 5-49b. As in the collision with Caroline, the motion planner commanded emergency braking before touching Skynet, but both vehicles were too close already for it to be effective.

We can conclude that the root cause of this accident, as that with Caroline, was a failure to anticipate unexpected behavior from a stopped or slowly-moving robot in a zone or intersection. Had Skynet truly been a static obstacle, or had it been perceived correctly as moving, and this accident would not have happened.

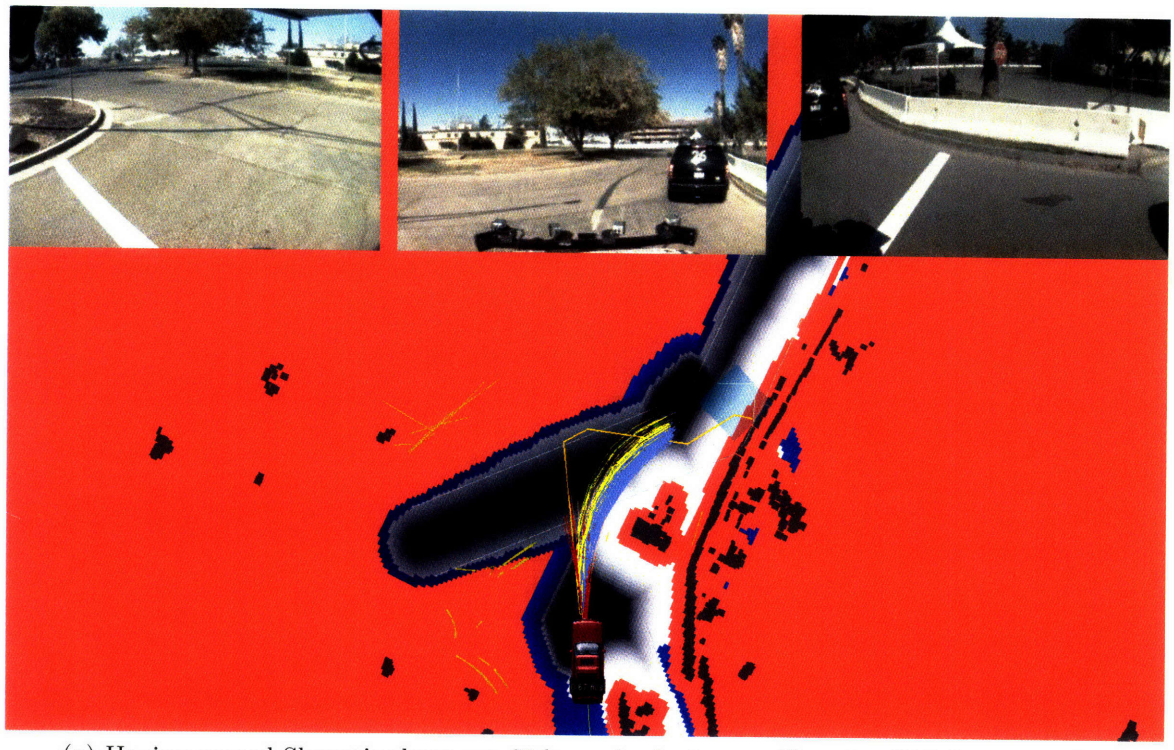


(a) Talos is going to pass Skynet's chase car when cones ahead block and merge the lanes.



(b) Talos tries to pass Skynet's chase car again, as Skynet (and it's chase car) is not moving forward.

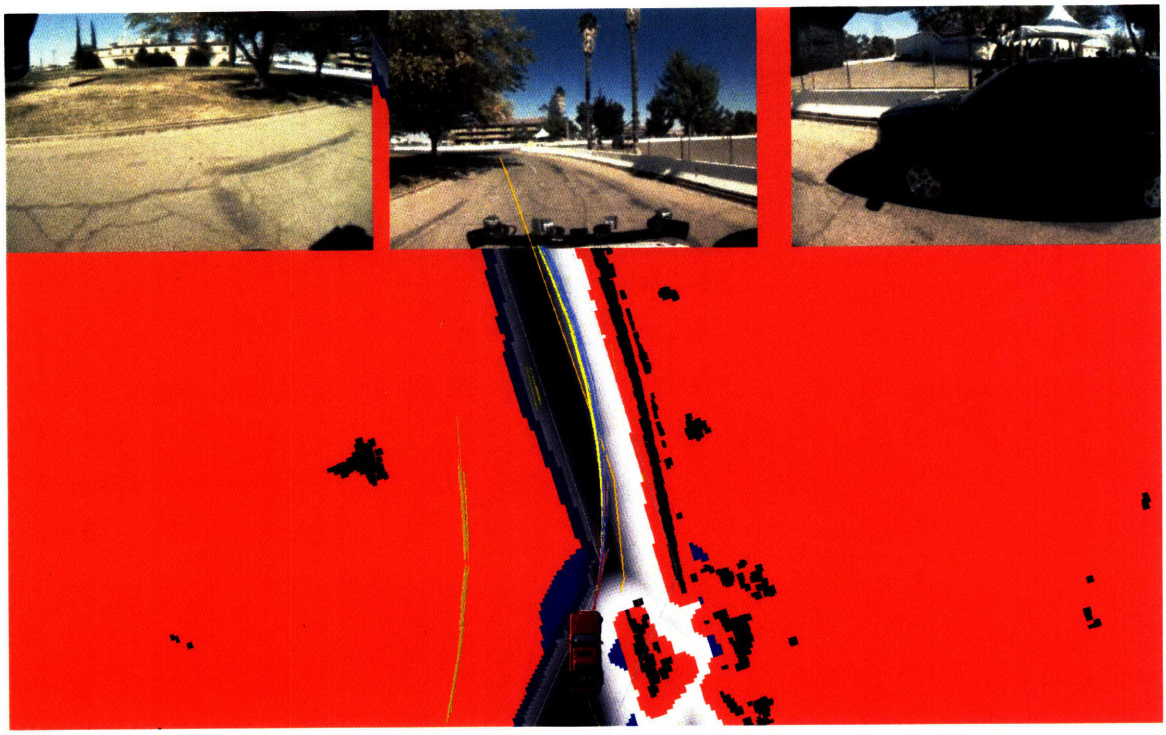
Figure 5-47: Talos tries to pass Skynet's chase car as it was moving slowly.



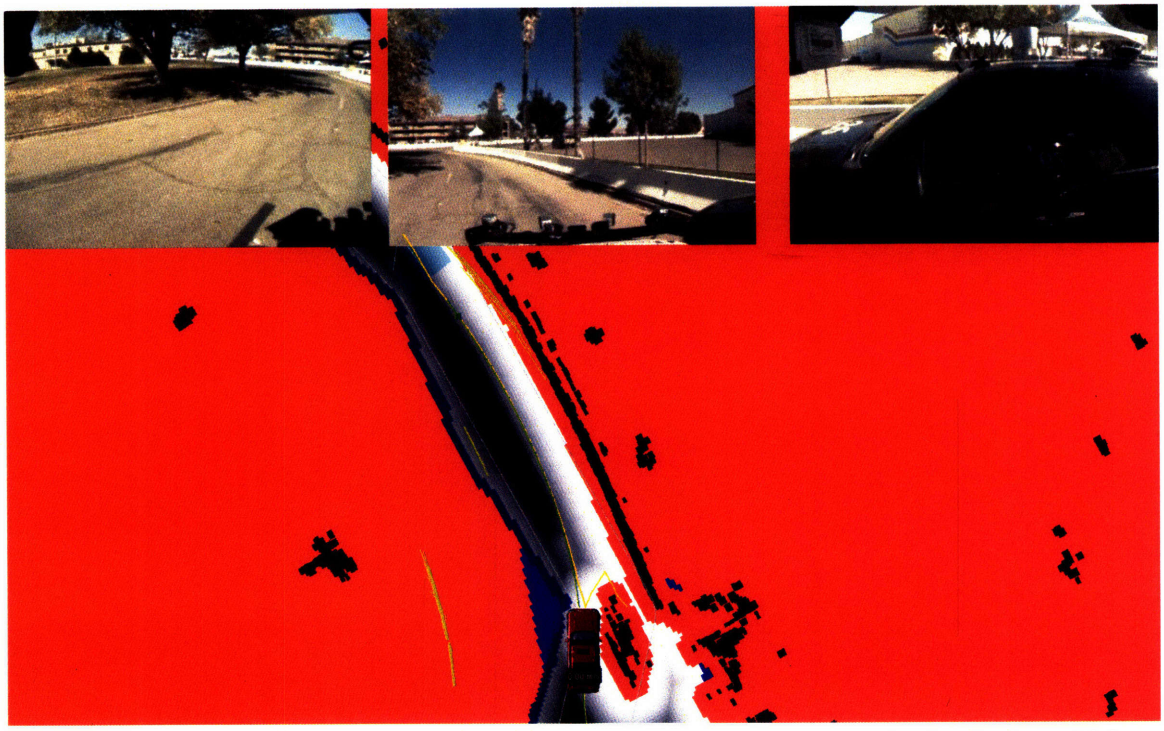
(a) Having passed Skynet's chase car, Talos embarks to pass Skynet, which is not moving.

(b) When Talos is passing Skynet, it suddenly starts moving forward.

Figure 5-48: Having passed the chase car, Talos embarks to also pass a stopped Skynet when it starts moving.



(a) Skynet keeps moving forward even as Talos merges ahead of it.



(b) Talos is choked against the lane boundary and brakes hard, but still both vehicles colllide.

Figure 5-49: Skynet is regarded as a static obstacle and Talos tries to merge back but both cars collide.

Chapter 6

Conclusions and Future Work

This thesis presented the Robust RRT algorithm, a sampling-based motion planning algorithm specifically developed for large robotic vehicles with complex dynamics and operating in uncertain and dynamic environments such as urban areas. This algorithm was implemented as the motion planner for Talos, the autonomous Land Rover LR3 that was MIT's entry in the 2007 DARPA Urban Challenge.

The Robust RRT algorithm extends the original RRT concept in five main ways to achieve better performance. The five main strengths of the algorithm are as follows.

1. Use of the closed-loop system for state propagation, i.e., sampling of the input to the closed-loop system, which enables easy handling of complex, nonlinear, and unstable dynamics.

2. Exploitation of the environment structure in sampling, increasing the probability that a given sample will be reachable and yield a feasible trajectory.

3. Use of efficient heuristics in the expansion of the tree and incorporation of a risk penalty to capture uncertainty in the environment and keep the vehicle a safe distance away from hazards.

4. Guaranteed safety of the vehicle assuming no unexpected changes in the environment, which is achieved by requiring that every trajectory sent for execution ends in a state corresponding to the vehicle stopped.

5. Risk evaluation following a lazy evaluation strategy, which enables the planning algorithm to spend most of the computation time in the expansion step even with a constantly changing environment.

The advantages of the Robust RRT algorithm were demonstrated through several simulation and actual log data obtained during the 2007 DARPA Urban Challenge. The use of a single planning algorithm for all the different scenarios encountered during the DUC (such as roads, big and open zones, blockages requiring U-turns) also demonstrates the flexibility of the Robust RRT algorithm for planning in varied settings.

A possible future direction for improving Robust RRT could be in better accounting for unexpected changes in the environment. The two incidents where Talos was involved during the DUC were caused by a failure to anticipate unexpected behavior from a seemingly static or a slowly-moving obstacle. The perception capabilities of the system will never be perfect, and therefore a method to better account for unexpected circumstances in case of a perception flaw at the motion planning level seems necessary.

Another area where Robust RRT could be improved is in escaping bug traps. The algorithm exploits the known environment structure to bias the sampling and guide the search. However, in the case where no feedback about the envrironment structure is available, finding a solution might prove very difficult for certain scenarios like bug traps. One possible method that might help in this situation is bidirectional search, and grow two RRTs, one from the vehicle location and another one from the goal. A big complication in this case, however, is to perform the backwards dynamic propagation for complex dynamical systems such as cars.

One very interesting area to which Robust RRT could be extended is in planning for multiple cooperative vehicles. In multiple vehicle settings, the robots could communicate among each other and share any information that his learned about the environment so that it could be used by other vehicles. How to integrate the information from the many vehicles poses many challenges. Furthermore, issues related to centralized or distributed coordination also arise. Multiple vehicle cooperation,

however, might bring some insights for helping reduce the uncertainty in the environment.

With greater concern about car safety from manufacturers, there will be an increasing need for motion planning approaches to play a role in *active safety* systems. Motion planning algorithms could be used to predict the future path to be followed by the vehicle and warn drivers when an imminent collision will take place. When the level of practically perfect reliability of the algorithm is achieved, the motion planner could even take control of the vehicle moementarily to avoid a collision if the driver is not reacting in properly or in time. Therefore, greater emphasis should be put on algorithm reliability and verification techniques.

Motion planning algorithms could also enter the niche of vehicle design, where they could be used to conduct performance and safety evaluations before constructing a vehicle. Planning algorithms could help in the *verification* of the design of vehicles, in which the vehicle is throughly tested to ensure that it performs as expected even under major failures that could result during its use. Using planning in software simulations to determine design flaws early in the vehicle development process would result in *time and cost reductions* for manufacturers. For example, a simulation would determine whether a sport utility vehicle (SUV) tumbles sideways during a high-speed turn without the need to build a real test vehicle. Similarly, a simulation could be used to violently crash a vehicle against a solid wall in order to determine its safety level before resorting to real crash testing.

Bibliography

[1] Jürgen Ackermann. *Robust Control: The Parameter Space Approach.* Springer, 2002.

[2] S. Arya and D. M. Mount. Algorithms for fast vector quantization. In *IEEE Data Compression Conference*, pages 381–390, March 1993.

[3] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings ACM-SIAM Symposium on Discrete Algorithms*, pages 271–280, 1993.

[4] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998.

[5] A. Atramentov and S. M. LaValle. Efficient nearest neighbor searching for motion planning. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 632–637, 2002.

[6] K. E. Bekris, B. Y. Chen, A. Ladd, E. Plaku, and L. E. Kavraki. Multiple query probabilistic roadmap planning using single query primitives. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[7] A. Beygelzimer, S. M. Kakade, and J. Langford. Cover trees for nearest neighbor. University of Pennsylvania, Available from http://www.cis.upenn.edu/~skakade/papers/ml/cover_tree.pdf, 2005.

[8] R. Bohlin and L. Kavraki. Path planning using Lazy PRM. In *Proceedings IEEE International Conference on Robotics & Automation*, 2000.

[9] M. S. Branicky, M. M. Curtiss, J. Levine, and S. Morgan. RRTs for nonlinear, discrete, and hybrid planning and control. In *Proceedings IEEE Conference Decision & Control*, 2003.

[10] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[11] Car-Accidents.com.

[12] P. Cheng and S. M. LaValle. Resolution complete rapidly-exploring random trees. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 267–272, 2002.

[13] P. Choudhury and K. Lynch. Trajectory planning for second-order underactuated mechanical systems in presence of obstacles. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2002.

[14] K. L. Clarkson. Nearest neighbor searching in metric spaces: Experimental results for sb(s). Bell Labs. Available from http://cm.bell-labs.com/who/clarkson/Msb/readme.html, 2003.

[15] J. Cortés. *Motion Planning Algorithms for General Closed-Chain Mechanisms.* PhD thesis, Institut National Polytechnique de Toulouse, Toulouse, France, 2003.

[16] Lars B. Cremean, Tully B. Foote, Jeremy H. Gillula, George H. Hines, Dmitriy Kogan, Kristopher L. Kriechbaum, Jeffrey C. Lamb, Jeremy Leibs, Laura Lindzey, Christopher E. Rasmussen, Alexander D. Stewart, Joel W. Burdick, and Richard M. Murray. Alice: An information-rich autonomous vehicle for high-speed desert navigation. *Journal of Field Robotics*, 23(9):777–810, September 2006.

[17] DARPA, October 2005. http://www.darpa.mil/grandchallenge05/index.html.

[18] DARPA, November 2007. http://www.darpa.mil/grandchallenge/index.asp.

[19] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer, third edition, April 2008.

[20] M. J. de Smith. *Distance and Path: The Development, Interpretation and Application of Distance Measurement in Mapping and Modelling.* PhD thesis, University College, University of London, London, 2003.

[21] Dmitri Dolgov. Real-time path planning in unknown environments (stanford racing team's parking planner). NIPS Urban Challenge Workshop, December 2007.

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

[23] J. J. Enright, E. Frazzoli, K. Savla, and F. Bullo. On multiple uav routing with stochastic targets: Performance bounds and algorithms. In *Proceedings AIAA Guidance, Navigation, and Control Conference and Exhibit*, August 2005.

[24] Fatality Analysis Reporting System Encyclopedia. National statistics.

[25] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance and Control*, 25(1):116–129, 2002.

[26] J. H. Friedman, J. L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[27] T. N. Gillespie. *Fundamentals of Vehicle Dynamics.* Society of Automotive Engineers, Warrendale, PA, 1992.

[28] J. Go, T. Vu, and J. J. Kuffner. Autonomous behaviors for interactive vehicle animations. In *Proceedings SIGGRAPH Symposium on Computer Animation*, 2004.

[29] Andy Greenberg. Robots meet reality. *Forbes.com*, November 2007.

[30] Lev Grossman. Building the best driverless robot car. *Time Magazine*, November 2007.

[31] Peter Henderson. Suv with mind of its own wins robot car race. *Reuters*, November 2007.

[32] http://www.cat.com/cda/layout?m=37483. Caterpillar's accugrade.

[33] Karl Iagnemma and Martin Buehler, eds. Special issue on the darpa grand challenge, part 1. *Journal of Field Robotics*, 23(8):461–652, August 2006.

[34] Karl Iagnemma and Martin Buehler, eds. Special issue: Special issue on the darpa grand challenge, part 2. *Journal of Field Robotics*, 23(9):655–835, September 2006.

[35] Karl Iagnemma and Martin Buehler, eds. Special issue on the 2007 darpa urban challenge. *Journal of Field Robotics*, 2008. To appear.

[36] P. Indyk. Nearest neighbors in high-dimensional spaces. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 877–892. Chapman and Hall/CRC Press, New York, 2004.

[37] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings ACM Symposium on Theory of Computing*, pages 604–613, 1998.

[38] S. Kagami, J. Kuffner, K. Nishiwaki, and K. Okada M. Inaba. Humanoid arm motion planning using stereo vision and RRT search. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[39] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann. Planning collision-free reaching motions for interactive object manipulation and grasping. *Eurographics*, 22(3), 2003.

[40] M. Kallmann and M. Mataric. Motion planning using dynamic roadmaps. In *Proceedings IEEE International Conference on Robotics & Automation*, 2004.

[41] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics & Automation*, 12(4):566–580, June 1996.

[42] J. Kim and J. P. Ostrowski. Motion planning of aerial robot using rapidly-exploring random trees with dynamic constraints. In *Proceedings IEEE International Conference on Robotics & Automation*, 2003.

[43] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings ACM Symposium on Theory of Computing*, pages 599–608, May 1997.

[44] Yoshiaki Kuwata, Justin Teo, Sertac Karaman, Gaston Fiore, Emilio Frazzoli, and Jonathan P. How. Motion planning in complex environments using closed-loop prediction. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008, to appear.

[45] Jean.-Claude Latombe. *Robot Motion Planning.* Kluwer, Boston, MA, 1991.

[46] Jean.-Paul Laumond, editor. *Robot Motion Planning and Control.* Springer-Verlag, Berlin, 1998. Available online at http://www.laas.fr/ jpl/book.html.

[47] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University, October 1998.

[48] S. M. LaValle. *Planning Algorithms.* Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[49] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 473–479, 1999.

[50] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, May 2001.

[51] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A K Peters, Wellesley, MA, 2001.

[52] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams. A perception driven autonomous urban vehicle. *Journal of Field Robotics*, 2008, to appear.

[53] T.-Y. Li and Y.-C. Shie. An incremental learning approach to motion planning with roadmap management. In *Proceedings IEEE International Conference on Robotics & Automation*, 2002.

[54] S. R. Lindemann and S. M. LaValle. Incrementally reducing dispersion by increasing Voronoi bias in RRTs. In *Proceedings IEEE International Conference on Robotics and Automation*, 2004.

[55] S. R. Lindemann and S. M. LaValle. Steps toward derandomizing RRTs. In *IEEE Fourth International Workshop on Robot Motion and Control*, 2004.

[56] Team LUX, March 2008. http://www.team-lux.com/?id=1.

[57] John Markoff. No drivers, but a lot of drive. *The New York Times*, November 2007.

[58] M. H. Overmars and J. van Leeuwen. Dynamic multidimensional data structures based on Quad- and K-D trees. *Acta Informatica*, 17:267–285, 1982.

[59] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[60] Paul Saffo. Move over, the robot's driving. *ABC News*, November 2007.

[61] T. Schouwenaars, J. How, and E. Feron. Receding horizon path planning with implicit safety guarantees. In *Proceedings American Control Conference*, volume 6, June 2004.

[62] J. T. Schwartz, M. Sharir, and J. Hopcroft. *Planning, Geometry, and Complexity of Robot Motion.* Ablex, Norwood, NJ, 1987.

[63] Andrei M. Shkel and Vladimir Lumelsky. Classification of the dubins set. *Robotics and Autonomous Systems*, 34:179–202, March 2001.

[64] R. L Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991.

[65] M. Strandberg. Augmenting RRT-planners with local trees. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 3258–3262, 2004.

[66] M. Strandberg. *Robot Path Planning: An Object-Oriented Approach.* PhD thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, 2004.

[67] Boris Thomaschewski. Dubins' problem for free terminal direction. *Technische Universitat Ilmenau, Fakultat fur Mathematik und Naturwissenschaften, Institut fur Mathematik*, Preprint No. M 09/2001, August 2001.

[68] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian,

and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23(9):661–692, September 2006.

[69] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[70] Chris Urmson, Charlie Ragusa, David Ray, Joshua Anhalt, Daniel Bartz, Tugrul Galatali, Alexander Gutierrez, Josh Johnston, Sam Harbaugh, Hiroki "Yu" Kato, William Messner, Nick Miller, Kevin Peterson, Bryon Smith, Jarrod Snider, Spencer Spiker, Jason Ziglar, William "Red" Whittaker, Michael Clark, Phillip Koon, Aaron Mosher, and Josh Struble. A robust approach to high-speed navigation for unrehearsed desert terrain. *Journal of Field Robotics*, 23(8):467–508, August 2006.

[71] André Vits, Ivan Hodac, Olivier Mossé, and Juhani Jaaskelainen. Final report of the esafety working group on road safety. Technical report, European Commission, November 2002.

[72] www.ez steer.com/. Trimble's ez-steer.

[73] J. Yakey, S. M. LaValle, and L. E. Kavraki. Randomized path planning for linkages with closed kinematic chains. *IEEE Transactions on Robotics and Automation*, 17(6):951–958, December 2001.

[74] A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *Proceedings IEEE International Conference on Robotics and Automation*, 2005.

[75] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.