RICE UNIVERSITY

Informed Planning and Safe Distributed Replanning under Physical Constraints

by

Konstantinos E. Bekris

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Approved, Thesis Committee:

Lydia E. Kavraki, Professor, Chair
Computer Science

Joe Warren, Professor
Computer Science

Edward Knightly, Professor
Electrical and Computer Engineering

Houston, Texas

JULY, 2008

Informed Planning and Safe Distributed Replanning under Physical Constraints

Konstantinos E. Bekris

Abstract

Motion planning is a fundamental algorithmic problem that attracts attention because of its importance in many exciting applications, such as controlling robots or virtual agents in simulations and computer games. While there has been great progress over the last decades in solving high-dimensional geometric problems there are still many challenges that limit the capabilities of existing solutions. In particular, it is important to effectively model and plan for systems with complex dynamics and significant drift (kinodynamic planning). An additional requirement is that realistic systems and agents must safely operate in a real-time fashion (replanning), with partial knowledge of their surroundings (partial observability) and despite the presence or in collaboration with other moving agents (distributed planning).

This thesis describes techniques that address challenges related to real-time motion planning while focusing on systems with non-trivial dynamics. The first contribution is a new kinodynamic planner, termed Informed Subdivision Tree (`IST`), that incorporates heuristics to solve motion planning queries more effectively while achieving the theoretical guarantee of probabilistic completeness. The thesis proposes also a general methodology to construct heuristics for kinodynamic planning based on configuration space knowledge through a roadmap-based approach. Then this thesis investigates replanning problems, where a planner is called periodically given a predefined amount of time. In this scenario, safety

concerns arise by the presence of both dynamic motion constraints and time limitations. The thesis proposes the framework of Short-Term Safety Replanning (STSR), which achieves safety guarantees in this context while minimizing computational overhead. The final contribution corresponds to an extension of the STSR framework in distributed planning, where multiple agents communicate to safely avoid collisions despite their dynamic constraints.

The proposed algorithms are tested on simulated systems with interesting dynamics, including physically simulated systems. Such experiments correspond to the state-of-the-art in terms of system modeling for motion planning. The experiments show that the proposed techniques outperform existing alternatives, where available, and emphasize their computational advantages.

Acknowledgments

I would like to thank my adviser Lydia Kavraki for being an inspiring mentor and supporter throughout my graduate studies. I am also thankful to my thesis committee members, Edward Knightly and Joe Warren, as well as to my recommenders, Nancy Amato, Antonis Argyros, Thierry Fraichard, James Kuffner and Manuela Veloso, for supporting my work and for their insightful comments.

I feel especially fortunate for being a member of the Physical and Biological Computing Group at Rice University. As a member of the group I had the opportunity to interact and collaborate with a great set of researchers and people.

My deepest gratitude and love go to my family and friends.

Work on this thesis has been partially supported by NSF IIS 0713623 and NSF IIS 0308237. The computational experiments were carried on equipment obtained by the above grants and by CNS 0454333 and CNS 0421109 in partnership with Rice University, AMD and Cray.

Contents

Illustrations

Chapter 1

Introduction

Motion planning was born by the need to understand the complexity of designing the motion for mobile or movable objects, as well to have algorithms that automatically compute motion [CLH+05, LaV06b]. Initially, the problem was abstracted as a fully geometric one [Lat91, Rei79]. The focus was on computing motions that avoided collisions with obstacles, where the difficulty rose from scene complexity and problem dimensionality. Even in this form, the problem is computationally hard [Lat91, Rei79]. Nonetheless, there has been progress over the years in the design of practical algorithms that allow the solution of problems that involve complex geometric constraints and many degrees of freedom. While algorithms still cannot conclude intractability or guarantee a solution in finite time, in practice they perform exceptionally well [KSLO96, HLM99, LK01a, SL03a, PBC+05].

Nevertheless, the problem is more complex for systems with differential constraints. Such constraints arise in non-holonomic vehicles that are under-actuated or in general when the control influence of a system is small compared to momentum. The challenge is that not every collision-free path is acceptable; it must also be feasible given the constraints. The dynamic model may not be even available, but simulated by a software package, like a physics engine [Smi06]. Additionally, agents often have to move and act autonomously with only partial knowledge of their surroundings, either because they depend on their sensing capabilities or due to dynamic changes in the environment. Motion planning problems can be further complicated by the presence of multiple moving systems in the same workspace. Interesting challenges arise when the objective is to coordinate the operation of such multi-agent systems so as to achieve a common objective.

Gradually the motion planning field is moving from just dealing with collision-free motions to addressing incrementally more challenging and realistic problems. This thesis is both an indicator and a contribution to this transition. The starting point and the foundation for this work remain the algorithms that have been traditionally the focus of the motion planning community. The direction, however, is to address those challenges that arise when motion planning has to be employed in realistic applications.

1.1 Motivating Applications

Applications that require solutions to interesting and challenging motion planning problems relate both to real systems, as in the case of robotic applications, as well as to simulated moving agents, as in computer games and realistic simulations.

Autonomous Vehicles and Robots

Autonomous vehicles used to belong in the realm of science fiction. However, the importance of this technology cannot be overlooked. It has the potential to considerably improve the current standard of living, save thousands of lives lost every year in automobile accidents and reduce the financial burden imposed to the health care system. This is why there has been increased attention recently to the scientific areas related to this technology.

Some of the first problems that have to be solved in order to achieve autonomous driving relate to sensing and estimation. An autonomous system must be aware of its surroundings and its position in the world before it is able to act. The recent Grand Challenge competitions organized by DARPA have shown the significant steps that have been made in these areas. Fig. 1.1 shows two automobiles that participated and won the last two events. The figure also shows the sophisticated sensing infrastructure that can be employed nowadays to assist autonomous driving.

Figure 1.1 : Wining cars in the DARPA Grand Challenges of 2005 and 2007.

Despite the successful completion of the last two competitions by multiple teams, the planning aspect of the events did not involve all the complexities of autonomous driving. For example, highway driving, which involves interesting physical constraints (i.e., increased momentum), was not part of the tasks that the autonomous vehicles had to execute. So, although there has been great progress in the area of autonomous driving, there are still important roadblocks to overcome before autonomous vehicles are used in everyday life. And these roadblocks correspond to motion planning challenges, especially when interesting dynamics are involved.

Driving naturally leads to flying. Planning algorithms can also help to navigate autonomous flying devices, such as blimps and helicopters through obstacles. They can also compute thrusts for a spacecraft so that collisions are avoided around a complicated structure, such as a space station.

Realistic and Interactive Simulation

Motion planning can be useful in many application as a simulation tool. By considering the effect of dynamics during the design phase of a product, performance and safety evaluations can be performed before actual construction. For example, planning can be used to assess whether a sports utility vehicle tumbles over while turning or when stopping abruptly. Time and costs can be spared by determining design flaws early in the development process.

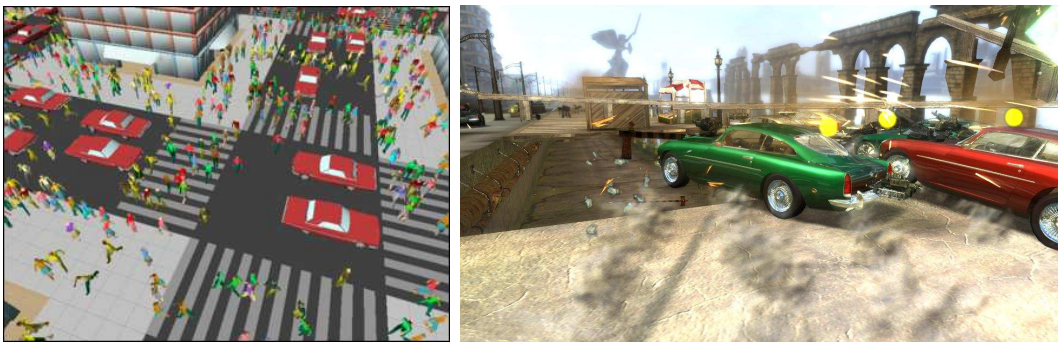

Figure 1.2 : Examples of simulation related applications, like (a) crowd and urban infrastructure simulations and (b) computer games, where motion planning has an important role.

Opportunities for using planning algorithms also abound in the area of computer games. Modern video games involve complex agents that exhibit sophisticated behavior. Planning algorithms can enable game developers to program character behaviors at a higher level, with the expectation that the character can determine on its own how to move in an intelligent way and adapt to different virtual environments.

Crowd simulations or simulations of urban infrastructures depend on the capability to realistically model the motion of agents in the simulated environment. An important characteristic of these applications is that they involve a large number of mobile agents. Thus, they require autonomous planning capabilities that have good scalability properties as the number of agents increases. Fig. 1.2 shows examples from crowd simulations and computer games that require motion planning capabilities.

Relation to Networking

Both robotics and simulation related applications have been greatly influenced over the last decades by advances in networking research and technology. For example, in robotics, there is an increased interest in robotic sensor networks. Such networks involve multiple units that are able to sense their surroundings,

communicate among themselves but also move and act. The hope is that such systems will be cheaper to deploy and more robust than a single very expensive and highly equipped unit in monitoring applications. In order to effectively, however, deploy and control such systems it is necessary to have distributed motion planning algorithms that utilize the communication capabilities that the units have in order to solve common tasks through coordination.

Moreover, networking plays an increasingly important role in the area of computer games. Some of the most successful games over the last decade are massively multi-player role playing games, where thousands of avatars are being controlled remotely by human users and operate in the same virtual world. There are many issues related to the issue of maintaining a consistent physical simulation between the different processors that are actively studied. As the characters become more complex and the gaming experience more realistic, there will also be an increased requirement for autonomous motion coordination between different agents. For example, existing games allow characters to penetrate one another as they move in the virtual world because the collision checking is an expensive operation. An underlying distributed planning module could be used to proactively select controls that will avoid collisions up to a degree that does not conflict with the human user's commands.

1.2 Basic Motion Planning Notions

Before continuing into what techniques are needed so that motion planners can be effectively employed in the above applications, it is important to define a vocabulary of basic motion planning notions.

The focus of a motion planning algorithm is a mobile or movable *system* for which we are planning for, such as a robot or a vehicle. This system operates in a *workspace*, which corresponds to the system's surroundings, especially the obstacles in the environment that the system cannot penetrate.

The set of all parameters and variables that fully describe the system in the workspace correspond to the system's *state*. The state could, for example, represent the position, orientation and velocity of a robot or a vehicle. A *valid state* fully respects all the *physical constraints* that are imposed to the system. For example, non-penetration with obstacles and velocity bounds are examples of such physical constraints that a valid state has to respect,. Planning problems involve a *state space* that encapsulates all possible states that the system could be in. The focus in this thesis is on problems with continuous, uncountably infinite state spaces.

A motion planning algorithm typically answers a *planning query*. A query typically involves an *initial state* where the system must start from and a *goal state* or often a *goal region* of states that the system must arrive in.

The purpose of motion planning is to select the *controls* that are able to manipulate and change the state of the system. An important part of the planning formulation is the specification of how the state changes when controls are applied. For continuous problems this is often expressed through an ordinary differential equation, referred to as the *state update equation*.

A *trajectory* is a sequence of states for the system. A *feasible trajectory* is a sequence of valid states which can be produced by time integrating the state update equation. A *plan* is a sequence of controls that can produce a feasible trajectory. For the type of motion planning problems considered in this thesis, we are interested in computing plans that produce feasible trajectories, which start at the initial state and reach a state in the goal region. Due to the problem's complexity, we are interested in finding a plan that causes arrival at a goal state, regardless of its efficiency.

The traditional geometric model of motion planning did not include parameters that involved dynamics, such as velocity, acceleration, torques and forces. If we ignore dynamics and extract only those parameters from the state that are necessary to fully describe the position and orientation of the system then we can define the system's *configuration*. Given a configuration it is possible to compute

whether a system is in collision or not with obstacles. In the case of a traditional model of a car-like vehicle, the configuration corresponds to its Cartesian coordinates and orientation. But for a manipulator, a configuration also encompasses the joint angles that define where each link is located inside the workspace. Similarly, to the state space, the *configuration space* (or $\mathcal{C}$-space) corresponds to the set of all possible configurations that the system can be found. A geometric *path* is a sequence of configurations for the system. The traditional geometric model of motion planning aimed to compute such geometric, collision-free paths. In that context, time typically was not explicitly referenced in the path.

1.3 Challenges

Despite the importance of motion planning in many exciting applications, autonomous planners must overcome many challenges until they can be used ubiquitely, safely and robustly.

Complex Physical Constraints

Driving an automobile can be used as an example of complicating physical constraints. Many people have difficulty with parallel parking and much greater difficulty parking a truck with a trailer. These problems are challenging because a car is constrained to move in the direction the rear wheels are pointing. Maneuvering the car around obstacles therefore becomes challenging. If all four wheels can turn to any orientation, the problem vanishes. Such constraints are referred to as non-holonomic constraints. Figure 1.3 shows an extremely complicated vehicle for which non-holonomic planning was employed by the industry.

For a car that moves at high speeds, as the speed increases, the effects of momentum become increasingly important. The car is no longer able to instantaneously start and stop, a capability that is actually reasonable to assume for parking problems. Although there exist planning algorithms that address such issues, there are still many unsolved research problems. This type of constraints are

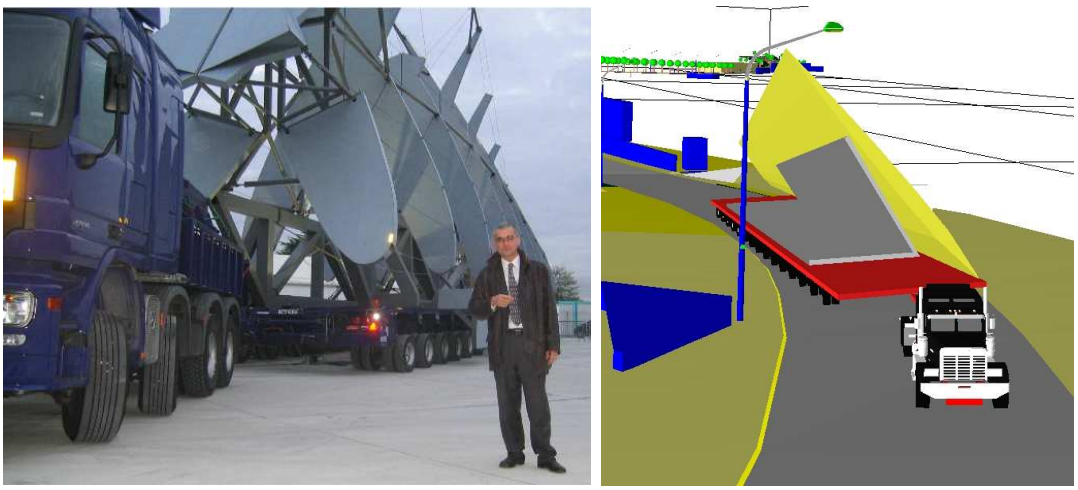

Figure 1.3 : (a) This gigantic truck was designed to transport portions of the Airbus A380 across France. Kineo CAM developed non-holonomic planning software that plans routes through villages that avoid obstacles and satisfy non-holonomic constraints imposed by 20 steering axles. (b) Simulation used for computing the path of the trucks.

referred to as dynamic constraints, since they relate to the dynamic parameters of a system, such as acceleration and momentum.

There is also a plethora of physical parameters that are typically ignored my existing motion planners, including some fundamental parameters of the physical world, such as gravity, friction and the effects of contacts. In this way, the output of a planner cannot be accurately followed by a real system. When the result is simulated it does not seem realistic.

One issue that complicates the motion planning process when additional physical parameters are incorporated is that problem dimensionality increases. For example, a three-dimensional geometric planning problem requires six parameters to model the state of a rigid-body. If dynamics are also stored in the state in the form of linear and angular velocities, then twelve parameters are needed to model each rigid-body. Dimensionality is not the most important parameter in defining how hard a motion planning problem is; it does impose, however, a significant computational overhead since it requires exploring significantly larger state spaces.

Partial Observability and Dynamic Environments

Another important challenge that motion planners have to address is the issue of partial observability. Most existing techniques operate under the assumption that they have a complete, detailed model of the world. Nevertheless, most real autonomous systems depend on sensing information in order to be able to model and represent the world. This implies that they cannot be aware of potential changes in the environment beyond their sensing radius, such as doors closing and opening or movable objects like furniture that have been moved since the construction of the map. In applications such as workspace exploration, there is no model available before the planning process is initiated.

Partial observability can be also an issue in the case of simulated agents, even in computer games, where it is often assumed that all the information about the world is available. One objective for AI agents in computer games is to operate given the same information that is available to a human player. For example, often today higher levels of difficulty for AI agents is achieved by providing to the agents additional global information, such as the entire map of the world and positions of opponents. This information is not available to a human player. In order to make the gaming experience more realistic it is necessary to design planning tools that are able to operate under limited information.

Even in the case that an autonomous agent is able to directly observe the entire environment, real or simulated, the problem might be complicated by the presence of moving obstacles. Since the future trajectory of the moving obstacles can be unknown, a planning algorithm cannot take the current state of the world as granted. Such problems require planners that adapt to dynamic and expected circumstances.

To address the issues of partial observability and dynamic environments, motion planning must be seen as a module in a much larger architecture that is able to execute operations such as prediction and estimation. Thus, it is neces-

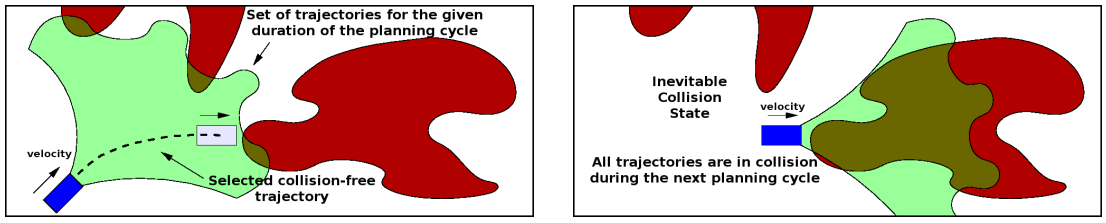

Figure 1.4 : A valid plan that results in a collision-free trajectory may still lead to an ICS during the next planning cycle.

sary to redefine the problem that motion planning algorithms must solve within this context. When real-time operation is necessary, a planner is not called to find a complete solution to a specific motion planning query where everything is known. Instead it is called so as to make progress towards the solution in an ever changing and partially unknown environment given the time limitations available.

Safety Concerns for Systems with Drift

The combination of partial observability and complex physical constraints raises some important safety concerns, especially for systems with significant drift. This safety issue is known as the problem of Inevitable Collision States (ICS), because systems with significant drift can end up in states from which they cannot avoid collisions into the future, due to momentum.

Consider the example of an automobile with very high velocity in front of a wall. The car is in an a collision-free situation but there might be no sequence of controls that it can employ in order to avoid collisions into the future. Such situations do not appear when a planning algorithm runs offline, because additional computation time can be invested into detecting whether there are collision-free trajectories out of such states. But in an online planner care must be taken so that the system does not end up in such situation during the following planning step.

Multi-agent Problems

A special but important case of having multiple systems operating in the same environment involves teams of cooperating agents that coordinate, potentially through communication, in order to achieve a common goal, as in robotic sensor network. There are many questions that must be answered in order to have such teams operate efficiently. How and when should the agents communicate? How can their information, such as a common map of the environment, be integrated? How can collisions between them be avoided? A fundamental concern is whether the coordination has any bottlenecks or whether it is fully distributed and can be scaled to large numbers.

While there have been numerous methods for distributed coordination in the artificial intelligence literature, these schemes do not deal directly with the issues that are involved in motion coordination, especially physical constraints and the safety concerns that rise from real-time limitations.

1.4 Contributions

This thesis proposes new algorithmic tools for designing planners that deal with complex physical constraints, real-time planning and multi-agent problems.

Informed Subdivision Tree

The first contribution is a novel planner, termed Informed Subdivision Tree (`IST`), that incorporates heuristics to solve motion planning queries that involve physical constraints more effectively. The planner has the following important characteristics.

Integrates Physics-based Simulators: There has been great progress over the last years in building software packages that simulate three dimensional objects and how they move according to Newton's laws of physics by using time integration methods. Such software is used in computer games, in computer animation to assist artists in producing realistic motion as well as in industrial design to test

the robustness of products. IST can integrate and has been tested with physics-based simulation so as to be able to model parameters typically ignored in the motion planning process, such as friction, gravity and contacts.

Kinodynamic Planning: In order to be able to deal with dynamic constraints, this thesis builds upon the methodology of kinodynamic planning. This is a search-based approach in the joint state space of kinematic (i.e., collision with obstacles, certain non-holonomic constraints) and dynamic (i.e., velocity and acceleration limits) constraints. This means that the algorithm searches for solution trajectories that directly respect all the physical constraints imposed on the moving system.

Using planning algorithms that consider dynamics and directly produce feasible trajectories has the advantage that they can be easily embedded into robotic systems because they output commands that can be directly interpreted by the moving platform.

Kinodynamic planning is also compatible with physics-based simulation software. It only depends upon the presence of a method that integrates the state of a system forward in time, which is what a physics-based simulator offers.

Informed but complete search: Searching in the joint state space, however, imposes significant computational overhead to the planning algorithm. Existing kinodynamic algorithms focus on guaranteeing that eventually the entire state space will be covered in order to be able to solve specific queries. Through the incorporation of heuristics, however, it is possible to guide the search procedure so as to minimize the amount of the state space covered in order to solve a planning query. IST attempts to maximize the utilization of any possible available heuristic during its operation. At the same time it is able to guarantee that eventually the entire state space will be covered and every query can be solved.

The overall algorithm outperforms uninformed state-space coverage oriented techniques, as well as existing informed variants. At the same time, it achieves the same theoretical guarantees that modern algorithms provide.

General Heuristic for Kinodynamic Planning

Although IST is compatible with a wide variety of alternative heuristics, another objective of this thesis is to provide a general method for constructing effective heuristics for kinodynamic planning. In order for a heuristic to be informative, it must be able to estimate good approximations of the distance between a state and the goal region. Many heuristics used today attempt to solve a simplified version of the overall planning problem and use this solution to bias the search operation of the planner in the entire state space. Typical examples include either the simplification of the moving system involved in the planning problem, such as computing a solution for a point size system in the workspace, or simplifying the workspace, such as computing a solution for the true moving system in an obstacle-free space.

This thesis proposes a heuristic for kinodynamic planning, which takes into account all the geometric properties of the workspace and of the moving system. Computing this heuristic is again a motion planning problem, only one that does not involve dynamics. That is why the proposed approach is based upon the state-of-the-art algorithms for the traditional geometric model of motion planning. These algorithms construct a graph data structure, a roadmap, out of collision free configurations for the moving system. Given the roadmap, a heuristic estimate of the distance from a state to the goal region is computed by mapping the path from the state to the goal to the closest roadmap path.

This thesis describes an algorithm for constructing a roadmap, called Efficient Visibility Roadmap with Useful Cycles (EVRC). The objective of the technique is to output a data structure that approximates a roadmap with the following two properties:

1. Every configuration can be connected to a configuration of the roadmap with a collision-free path

2. Every path in the configuration space can be deformed to a path on the roadmap where the path remains collision-free throughout the deformation

These two properties are important so that the roadmap paths can be good approximations of the true shortest geometric paths and in this way the heuristic to be informative. Another objective is to keep the computational cost of computing the roadmap as low as possible since `EVRC` must be called before `IST` to preprocess the space and provide the necessary information for the online computation of the heuristic. Then, the computation of the heuristic itself must be as inexpensive as possible as well.

It is shown here that while this heuristic is considerably more informative than existing alternative, the cost of computing it is not prohibitive and overall the integration of `IST` with `EVRC` results in an efficient kinodynamic planner. Borrowing the properties of sampling-based planners, `EVRC` has the advantage that is very general and can be applied to a wide range of systems.

Safe Replanning for Systems with Drift

`IST` and `EVRC` provide a way to define a fast search stragegy for hard planning problems with physical constraints. In order to deal with real-time requirements is to necessary to design a strategy that recomputes trajectories online. The focus is on systems with significant drift, which cannot stop instantaneously, and raise the issue of Inevitable Collision States.

This thesis describes a framework for replanning called Short-Term Safety Replanning (`STSR`) to compute trajectories online while providing safety guarantees. The replanning framework calls a planner to compute a partial path towards the goal given a predefined time limit. This path is executed during the consecutive planning step, where the planner in parallel computes future trajectories. In this way there is a pipeline of operations that include sensing, planning and execution.

The safety guarantees provided imply complete avoidance of Inevitable Collision States in replanning applications that involve static obstacles, such as exploration and planning among unexpected obstacles. In problems that involve dynamic obstacles it results in τ-safety, which means the algorithm can compute a solution that is safe for a certain time period into the future τ.

In order to provide these safety guarantees a motion planning algorithm has to execute additional calls to a collision checking routine, which is the most expensive operation a planner can execute. This can significantly reduce the capability of a planner to effectively explore the state space in order to come up with a good quality solution. The STSR framework identifies the minimum set of states that have to be checked for safety during each planning cycle so as to minimize this collision checking overhead. Compared with previous approaches for safe replanning with dynamics the technique is able to achieve the same safety guarantees but at a much smaller computational cost. The STSR framework has been integrated with the IST algorithm to solve replanning problems.

Distributed Motion Planning

Finally, this thesis proposes an extension of the STSR framework for problems that involve multiple coordinating systems with drift, such as a networked team of vehicles, that replan online to solve a common task. The extension is achieved by identifying the amount of information that multiple dynamic systems have to exchange in order to avoid Inevitable Collision States. The objective is to maintain the following invariant: during each planning cycle, every system has at least one available trajectory that is collision free and does not lead to ICS.

Given this information, the thesis describes two communication protocols that achieve distributed collision avoidance and an integration with planning techniques such as IST. The first protocol is a prioritized scheme, where through the use of priorities, the various systems are able to resolve conflicts and maintain the invariant. While the prioritized protocol achieves the desired safety properties, it raises scalability concerns. The second protocol is a fully distributed approach that is based on an asynchronous message-passing optimization technique. This approach is not only able to achieve the safety guarantees of the prioritized scheme but can also scale much better to larger teams of vehicles.

There are no comparable techniques in the related literature that simultaneously address all the challenges involved in the formulation of this problem: systems with drift, replanning, distributed and scalable coordination. Moreover, the proposed technique can also address additional group constraints, such as sensor network type features like radial limitations so as to maintain wireless network connectivity.

1.5 Thesis Roadmap

Chapter 2 describes related work to this work, which is based mostly in the motion planning literature but provides also a short overview of related results from work in control theory and artificial intelligence.

Chapter 3 describes the operation of the `IST` planner, specifically how the algorithm searches the state space guided by a heuristic.

Chapter 4 outlines the `EVRC` method for constructing a configuration space roadmap and how it can be used to provide the heuristic estimate within `IST`.

Chapter 5 proposes the `STSR` scheme for replanning. It points out the safety concerns that arise in kinodynamic replanning and shows how `STSR` can achieve safety and reduce computational overhead.

Chapter 6 describes the extension of `STSR` on multi-agent coordination problems and the two distributed protocols for multi-agent coordination.

Chapter 7 provides the experimental evaluation of the described techniques and comparison with competing methodologies.

Finally, chapter 8 closes this thesis we a summary of the most interesting points from the proposed algorithms, a list of their limitations and directions for future research.

Chapter 2

Background

Motion planning is a term that means different things to different research communities. In particular, (a) robotics, (b) control, (c) artificial intelligence and (d) computational geometry/algorithmic research, are four areas where motion planning problems are being actively studied.

Robotics addresses the automation of mechanical systems that sense, compute and act. A fundamental need in robotics is to have algorithms that convert high-level specifications of tasks from humans into low-level descriptions of how to move in the physical world. Related to motion planning is the problem of mobile robot navigation, which typically ignores dynamics and focuses on the translations and rotations required to move the mobile robot on a plane. Trajectory planning then translates the solution of a robot navigation algorithm and determines how to follow it so as to respect the robot's mechanical limitations. Robot navigation research pays a lot of attention to uncertainty, modeling errors and optimality.

In **control theory**, the term of motion planning refers to the design of inputs to a nonlinear dynamical system that drives it from an initial state to a specified goal state. Obstacles are often not considered in this formulation. Control theory also develops feedback policies, which adaptively respond during online execution, and has focused on stability, which ensures that the dynamics do not cause the system to get out of control. A large emphasis is also placed on optimization to minimize the consumption of resources, such as energy or time.

In **artificial intelligence**, planning is more of a discrete nature. Instead of moving objects through continuous spaces the task might be a puzzle or to achieve a task that is modeled discretely. In this way, planning and problem solving are

almost synonymous in the context of the AI literature. There is also a lot of work on adversarial problems, language representation issues for planning, modeling uncertainty and optimization.

Finally, in **computational geometry** and **algorithmic** research, the interest has been mostly in studying the computational complexity and designing algorithms for geometric problems that involve motion. Also there has been a lot of research in related problems such as collision checking and other applications that relate to motion planning such as computer graphics, animation and games.

This chapter attempts to provide an overview of motion planning research, which includes contributions from the above areas. We will initiate this review with the theoretical results that stem from computational geometry and algorithmic research, which was very active during the late '70s and '80s in motion planning. Then we will describe some important ideas that appeared in control theory in early '90s. In mid '90s there was the appearance of sampling-based algorithms for motion planning, which have become dominant in the field since then and form the basis of the algorithms in this work. Since this thesis focuses on problems that involve dynamic constraints, replanning and multi-agent problems, the last three sections present the related literature to these problems in the context of sampling-based motion planning. A lot of the work in the last two sections on replanning and multi-agent problem is also related to artificial intelligence and mobile robotics.

2.1 Theoretical Foundations

A classical version of motion planning is sometimes referred to as the Piano Mover's Problem. It is posed for polyhedral moving systems in a polyhedral workspace an it has been shown to be `PSPACE`-hard [Rei79]. In two dimensions this problem is also known as the Sofa mover's problem. A series of theoretical results provided polynomial time algorithms for problems with fixed dimensions. For example an algorithm for the Sofa mover's problem has complexity $O(n^5)$

[SS83a], for moving a fixed number of discs the complexity is $O(n^3)$ [SS83b], for moving a 2D star-shaped robot with k arms [SAS84] it is $O(n^{k+4})$ and for moving a 3D rod shaped robot [SS84] the complexity is $O(n^{15})$.

The results for problems with fixed dimension suggested an exponential dependence on the problem dimensionality [SS83a, SS84]. A single exponential algorithm in $\mathcal{C}$-space dimensionality was proposed by Canny and showed that the problem is `PSPACE-complete` [Can88]. Although impractical, the algorithm serves as an upper bound on the general version of the basic motion planning problem. It applies computational algebraic geometry techniques for modeling the $\mathcal{C}$-space to construct a *roadmap*, a 1D subspace that captures the connectivity of $\mathcal{C}_{free}$. Other approaches attempted to approximate the structure of $\mathcal{C}$-space but they were also impractical [BP83, Per83, KD86].

The complexity of the problem motivated work in path planning research. One direction was to study subclasses of the general problem for which polynomial time algorithms exist [HS96]. Even some simpler, special cases of motion planning, however, are at least as challenging. For example, the case of a finite number of translating, axis-aligned rectangles in $\Re^2$ is `PSPACE`-hard as well [HSS84]. Some extensions of motion planning are even harder. For example, a certain form of planning under uncertainty in 3D polyhedral environment is `NEXPTIME`-hard [CR87]. The hardest problems in `NEXPTIME` are believed to require doubly-exponential time to solve.

A different direction was the development of planners that were practical under realistic assumptions. For example, several algorithms exist for constructing roadmaps when $\mathcal{C} = \Re^2$ and $\mathcal{C}_{obs}$ is polygonal. Most of these cannot be directly extended to higher dimensions.

The *maximum clearance roadmap* (or *retraction method*) [OY82] constructs a roadmap that keeps paths as far from the obstacles as possible. The best-known algorithm runs in $O(n \lg n)$ time in which n is the number of roadmap curves [Sha04]. The *shortest-path roadmap* [Nil69] (Figure 2.1) contains paths that actually touch the obstacles, which must be allowed for paths to be optimal.

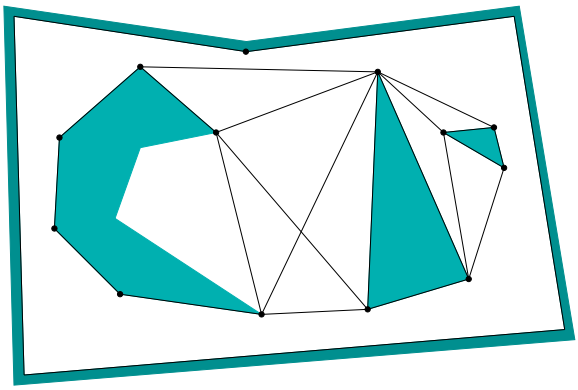

Figure 2.1 : The shortest-path roadmap connects vertices in $\mathcal{C}_{obs}$.

An $O(n^2 \lg n)$-time construction algorithm can be formed by using a radial sweep algorithm from $\mathcal{C}$ vertices. It can theoretically be computed in time $O(n^2 + m)$, in which m is the total number of edges in the roadmap [O'R04]. Figure 2.2 illustrates the *vertical cell decomposition* approach. The idea is to decompose $\mathcal{C}_{free}$ into cells that are trapezoids or triangles. Planning in each cell is trivial because it is convex. A roadmap is made by placing a point in the center of each cell and each boundary between cells. Any graph search algorithm can be used to quickly find a collision-free path. The cell decomposition can be constructed in $O(n \lg n)$ time using the *plane-sweep principle* [Cha87, dBvKOS00].

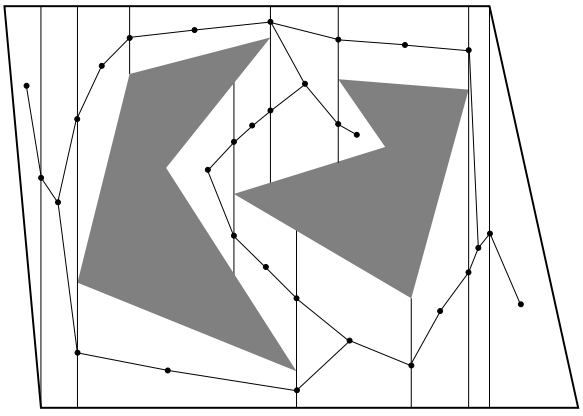

Figure 2.2 : The roadmap derived from the vertical cell decomposition.

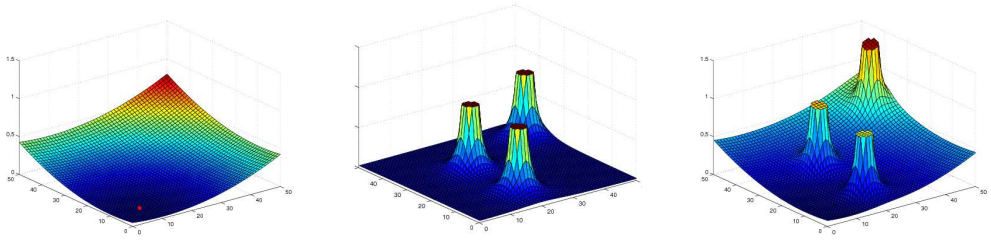

Figure 2.3 : Attractive and repulsive components define a potential function.

2.2 Potential Functions

A different approach for motion planning is inspired from obstacle avoidance techniques [Kha86]. It does not construct a roadmap, instead it builds a differentiable real-valued function $U : \Re^m \rightarrow \Re$, called a potential function [HA88, Kod89, HA92]. The potential function guides the motion of the moving system. The potential consists of an attractive component $U_a(x)$, that pulls the robot towards the goal, and a repulsive component $U_r(x)$, that pushes the robot away from the obstacles as Figure 2.3 shows. The gradient of the potential function is a vector $\nabla U(x) = DU(x)^T = [\frac{\partial U}{\partial x_x}(x), \ldots, \frac{\partial U}{\partial x_m}(x)]^T$, which points in the direction that locally maximally increases U. A path can be computed by starting from x_I and applying "gradient descent":

1 $x(0) = x_I$; $i = 0$;
2 **while** $\nabla U(x(i)) \neq 0$ **do**
3 $\quad x(i+1) = x(i) + \nabla U(x(i))$
4 $\quad i = i + 1$

This gradient descent approach does not guarantee a solution to the problem since it can only reach a local minimum of $U(x)$, which may not correspond to the goal state x_G as Figure 2.4 shows. A planner that makes uses of potential functions and attempts to avoid the issue of local minima is the *randomized potential field* (RPP) [BL91], which followed a stochastic approach and was later proved to be probabilistically complete [LL96]. The algorithm combines gradient descent, random walks and backtracking. However, it requires substantial parameter tuning. Sequential search with backtracking was also explored [GG95].

The gradient of the potential function can be also used to define a *vector field*, which assigns a motion for the robot at any configuration $x \in \mathcal{C}$. This is an important advantage since it does not only result in a single path, but a *feedback control* strategy. This makes the approach more robust against control and sensing errors. Most of the techniques in feedback motion planning are based on the idea of *navigation functions* [RK92], which are potential functions properly constructed so as to have a single minimum. A function $\phi : \mathcal{C}_{free} \rightarrow [0, 1]$ is called a navigation function if it:

- is smooth (or at least C^k for $k \geq 2$)
- has a unique minimum at x_G in the connected component of the free space that contains x_G,
- is uniformly maximal on the free space boundary
- and is Morse, which means that all its critical points, such as saddle points, are isolated and can be avoided with small random perturbations.

Navigation functions can be constructed for sphere boundary spaces centered at x_I that contain only spherical obstacles as Figure 2.5 shows. Then they can be extended to a large family of $\mathcal{C}$-space that are diffeomorphic to sphere-spaces, such as star-shaped spaces as in Figure 2.5.

Putting the issue of local minima aside, another major challenge for such potential function based approaches is constructing and representing the $\mathcal{C}$-space in the first place. This issue makes the applications of these techniques too complicated for high-dimensional problems.

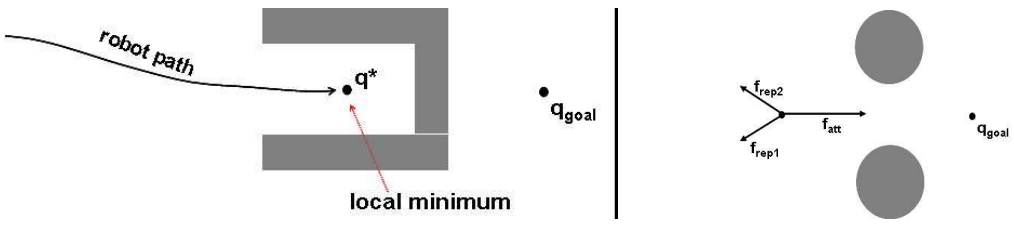

Figure 2.4 : Local minima problems with potential functions.

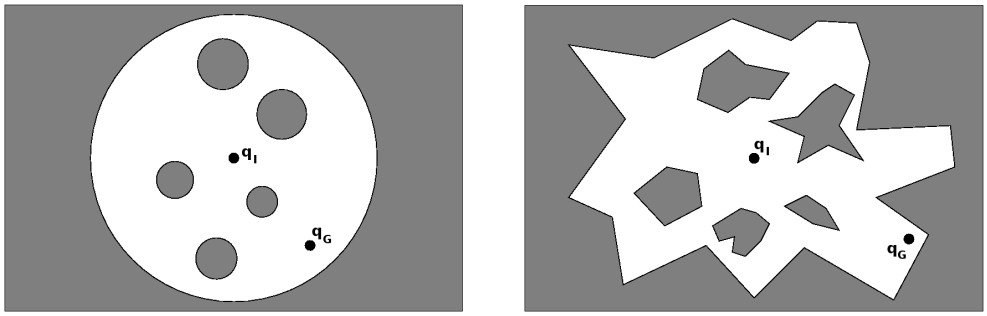

Figure 2.5 : Examples of sphere and star spaces.

2.3 Sampling-based Motion Planning

Sampling-based planning is a general approach that has been proven successful in practice with many challenging problems. It avoids the exact geometric modeling or approximation of the $\mathcal{C}$-space that was the bottleneck of previous approaches. Examples of these planners are the Ariadne's Clew algorithm [ATBM92, BATM94, AGM98] and the Probabilistic Roadmap Method (PRM) [KL94, Ov94, KSLO96, Kav95, Šve97]. Especially the latter, PRM, proved to be very successful in problems with complex geometries. It is also easier to implement than its algebraic counterpart, Canny's algorithm. PRM also benefited enormously from advances in collision checking [LC91, Qui94, LM91, GLM96, KPLM98, EL00]. In retrospect, using collision checking as a black box (as shown in Figure 2.6) was a powerful primitive of PRM that contributed to its performance, ease of implementation and subsequent widespread use.

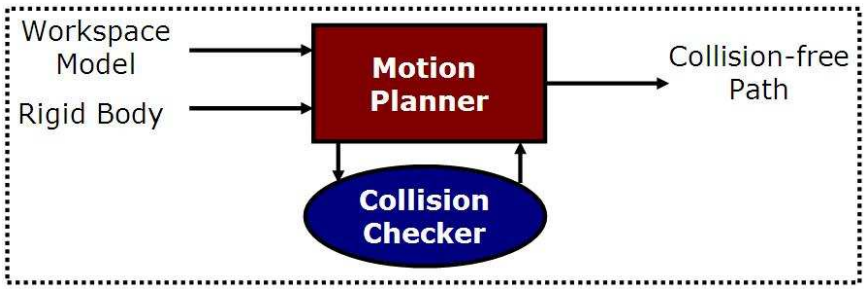

Figure 2.6 : Collision checking is used as a "black box" within the sampling-based motion planning framework.

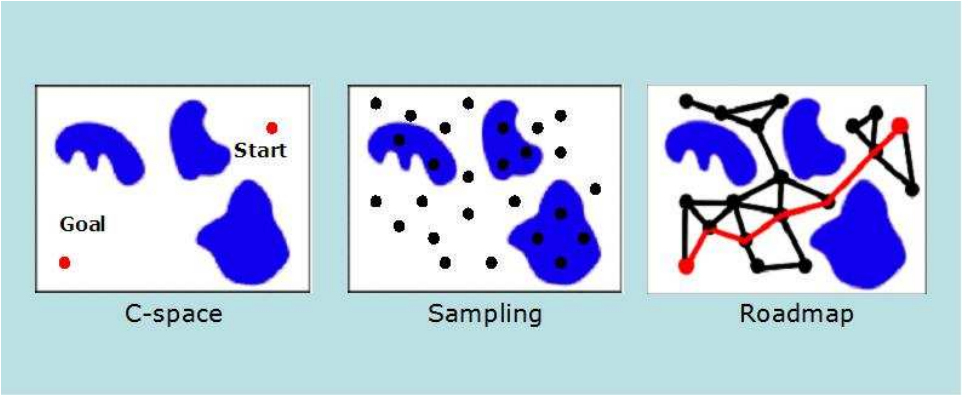

Figure 2.7 : The Probabilistic Roadmap Method (PRM) operates in the configuration space ($\mathcal{C}$) where the robot is modeled as a single point. PRM aims to construct a graph in $\mathcal{C}$ that trully expresses $\mathcal{C}$'s topological properties by sampling collision-free configurations. When the graph is available a planning query can be solved by connecting the start and the goal configuration to the graph.

PRM splits planning into two phases, a learning and a querying phase. During learning, PRM samples collision-free configurations and connects them with simple paths to build a roadmap. The initial implementation used uniform sampling and straight line paths [KSLO96]. Given a roadmap, multiple queries can be answered quickly. First the initial and goal configurations are connected to the roadmap and then planning amounts to solving the corresponding graph search problem. An illustration of a PRM roadmap is given in Figure 2.7.

PRM's efficiency depends on how well the sampling strategy captures the connectivity of the free space. A major factor that affects PRM 's performance is the presence of "narrow passages" [ABD+98, LK02, HKL+98]. In order to solve problems with "narrow passages", PRM must select samples from very small sets in order to connect the roadmap. A plethora of biased sampling techniques were introduced to improve the effectiveness of PRM. Example include quasi-random sampling [LB02], or a variety of techniques that focused sampling on subsets of the configuration space. These subsets could be either parts of the space with low connectivity [KSLO96], or parts of the space close to the obstacle boundaries [ABD+98, BOvdS99] or the medial-axis of the free space [GHK99, WAS99, HK00].

PRM takes advantage of the roadmap constructed during the learning phase to efficiently answer multiple queries. Single query problem can be often solved more efficiently by focusing the exploration only on certain parts of the $\mathcal{C}$-space. This motivated approaches that focus on single queries. One approach for single-query planning is to speed up computation by doing collision checking only when it is necessary. The lazy PRM variant [BK00, SL03a] constructs a roadmap whose nodes and edges are initially assumed to be collision-free. The roadmap can even be computed implicitly on a grid [Boh01, LB02]. Lazy PRM searches the roadmap graph to find a path that connects the query configurations. The computed path is then checked for collisions. The path is discarded if it is in collision and the violating nodes and edges are removed from the roadmap. Lazy PRM continues to search for alternative paths until a collision-free path is found.

The benefits of sampling-based planning come at the cost of sacrificing completeness. That is, path non-existence cannot be proven with sampling-based planners. A looser guarantee, termed *probabilistic completeness* is provided instead. If an algorithm is probabilistic complete and a path exists, then the planner will find it eventually [KSLO96, KLMR96, KKL96, Šve97, LK04].

An alternative approach to roadmap-based planners yielded in a family of planners known as tree-based planners [LK01b, HKLR02, LK05a, SL03a, YJSL05, BB07]. Tree-based planners bias start the exploration by rooting a tree at the initial configuration and incrementally explore the relevant parts of the $\mathcal{C}$-space by advancing the tree toward the goal. The success of tree-based planners depends on the strategy employed to expand the exploration tree. Popular approaches include the Rapidly-Exploring Random Tree (RRT) [LK99], the Expansive Spaces Tree (EST) [HLM99], the Single-Query Bidirectional Lazy SBL [SL03a]. Some instances of tree-based planners build two trees, one rooted at the initial configuration and another one rooted at the goal configuration, and grow the trees toward each-other [HLM99, LK01a, SL03a]. The Sampling-based Roadmap of Trees (SRT) leverages the benefits of PRM and tree-based planners. It expands multiple trees from various initial seed points. Neighboring trees are then grown

toward each-other giving rise eventually to a roadmap. SRT has proven effective both for multiple-query and single-query planning while it can also be parallelized very efficiently [BCL+03, PBC+05] The tree-based planners have been successfully used to solve planning problems with kinematic and dynamic constraints, which are discussed in the following section.

2.4 Tree-based Algorithms and Kinodynamic Planning

A kinematic path, returned by a planner such as PRM may not be executable due to bounds on velocity, acceleration and applied forces [DLOS98]. To alleviate this problem motion planners must incorporate a higher level of physical realism. The approach is to plan directly in the state space instead of the $\mathcal{C}$-space by including the parameters related to the motion constraints.

Algebraic solutions for the computation of exact paths under dynamic constraints are known only for 1D and 2D point masses with velocity and acceleration constraints [O'D87, CRR91]. Research in optimal control showed that optimal paths can be achieved with piecewise-extremal ("bang-bang") controls and a finite number of switches [Hol83, BDG85, SS85, Sch87]. Approximation methods that use grids have proved to depend exponentially in the number of grid points or resolution [SH85, SD88]. Donald et al. [DXCR93] provided the first optimal-approximation polynomial-time algorithm for a point mass with dynamics. For first order vehicles [FW88, Wil88] there are ways to characterize minimum wheel-rotation paths [RS90, BM02]. A different methodology, related to potential fields, employs path deformation to adapt online a precomputed path given motion constraints and sensing observations [KJCL97, BK91, QK93, LBL04].

A hierarchical approach exists in techniques that use PRM. The planner can be used to produce a roadmap of kinematic paths, which are later adapted to the dynamic constraints of the robot [SSLO98, SA01]. Such techniques must compute a valid trajectory to connect two given states, a difficult sub-problem in general, known as the steering problem [LFV04] and hence the approach is

not generally applicable. On the other hand, tree-based planners apply forward integration of controls instead of steering. This is a simple and fast primitive, which naturally simulates the underlying propagation model of a system. In this way, tree-based planners [LK01b, HKLR02, LK05a, CSL01, CL03, CL02, KVdP06] have become the norm in kinodynamic planning [CLH$^+$05, LaV06b]. Especially, the Path-Directed Subdivision Tree (`PDST`) planner [LK05a] has shown good performance in such planning instances and allows biasing the search of the algorithm while providing probabilistic completeness. Chapter 3 proposes a new tree-based planner for kinodynamic problems. However, it displays several key differences with the planners above as discussed. More details about the techniques related to the proposed algorithm will be provided in section 3.2.

2.5 Real-time Planning and Replanning

The original sampling-based planners were offline methods and assumed known workspaces. Planning with partial-observability requires interleaving sensing, planning and execution, where a planner is called frequently and has finite time to replan a trajectory. Replanning from scratch is possible [HKLR02] but recent methods use information from previous planning cycles to speed up the performance of replanning [BV06, FKS06, ZKB07, GKX07]. There are also methods that use a roadmap to replan online [BFK06, KM04].

When replanning with a sampling-based planner for a system with second-order dynamics, safety issues arise: a collision-free but partial plan may lead a vehicle to a state from which collisions cannot be avoided due to the dynamics (Inevitable Collision States (`ICS`) [FA04, PF05, FDF02, BK07]). This problem is particularly acute when multiple second-order vehicles operate in close proximity in the same environment. Similarly, a partial plan could also lead to states from which network connectivity will be inevitably lost. A framework that deals with `ICS` and real-time planning for a single vehicle has been recently developed in the sampling-based planning literature [FA04, BK07].

2.6 Planning for Multiple Systems

The use of sampling-based planners in multi-agent problems is limited and typically follows a centralized approach [CRL03, CBR02]. *Centralized* planning is reliable [SL03b] but computationally expensive due to the exponential dependency on problem dimensionality. *Decentralized* methods, such as prioritized schemes [BBT02], plan separately for each robot and then coordinate robots' interactions. Planning is faster but leads to collisions [SL03b, CRL03]. An important challenge is how to make decentralized planning more reliable [SI06]. We show in this thesis that it is possible to achieve safety in a decentralized scheme that employs priorities even with second-order dynamic constraints [BTK07a].

There are multiple techniques for decentralized motion planning in control theory [Mur07]. In formation control agents move while maintaining preassigned relative positions, which can be achieved with potential-fields [OFL04, OS06, PDKC03], leader-follower approaches [TPK04, EHS01] or local control laws [PSFB06]. Decentralized navigation functions [DKT06, LDK04] provide a feedback solution and can be used for vehicles with independent goals. Most methods focus on providing elegant stability proofs but are difficult to apply in general state spaces (e.g. complex obstacle, robot shapes and dynamics) [LaV06a].

An alternative to priority-based schemes for decentralized solutions can be found in the AI literature. There are many techniques for distributed constraint satisfaction [YH00] and optimization [MSTY05] for teams of cooperating agents, such as factored Markov Decision Processes [GKP02] and auction mechanisms [GM02, DZKS06]. The work in this thesis of distributed motion coordination employs message-passing, asynchronous algorithms for coordination related to loopy belief propagation, a method for distributed optimization in constraint networks [Pea88, KV06]. These message passing algorithms have been successfully applied to solve distributed inference problems in wireless sensor networks [PK04]. Employing such message-passing algorithms for coordinating vehicular networks results in better scalability than priority-based schemes [BTK07b].

Chapter 3

Informed Kinodynamic Planning

Search-based techniques for kinodynamic planning explore the entire state space for trajectories that respect the differential constraints. A major advantage is that they are applicable to a variety of different systems. They construct a "reachability tree" in the state space through a sampling operation [LK01b, HLM99, LK05a]. Although effective in eventually solving hard planning instances, they have high computational requirements as search methods, especially in kinodynamic problems that are typically higher-dimensional compared to geometric ones.

This chapter focuses on reducing the solution time of search-based techniques for kinodynamic planning by incorporating heuristics based on workspace and query knowledge. We detail a new method, the Informed Subdivision Tree (`IST`), that balances greedy and methodical search while providing guarantees that eventually every problem can be solved. In simple parts of the state space the exploration is greedily guided by the heuristic. In constrained parts, such as narrow passages, the heuristic may not be beneficial and the algorithm automatically explores alternative routes for a solution. This methodical behavior is a result of an adaptive state-space subdivision scheme that estimates online the algorithm's performance in exploring the entire state space. Experimental comparisons on physically simulated systems between `IST` and uninformed planners [LK01b, LK05a] as well as with informed versions [LK01b, US03] show that `IST` outperforms the alternatives. `IST` also reports better quality paths in certain complicated workspaces, as in maze-like environments.

3.1 Problem Definition

Consider a moving system, such as a robot, whose motion is governed by differential equations of the form:

$$\dot{x}(t) = f(x(t), u), \qquad g(x(t), \dot{x}(t)) \leq 0 \tag{3.1}$$

where f, g are smooth; $x(t)$ is a state and fully describes the system at time t. The set of all states is the state space $\mathcal{X}$. The set of states for which the moving object is not in collision is the free state space $\mathcal{X}_{free}$. The set of all controls u define the control space $\mathcal{U}$. For a given $x(t)$, u is *valid* if it respects Eq. 3.1.

We are particularly interested in non-holonomic systems with second-order constraints, such as a car controlled by bounded acceleration and bounded steering velocity. The following notation will be useful in our description:

- A *plan* $p(dt)$ is a time sequence of controls: $p(dt) = \{(u_1, dt_1), \ldots, (u_n, dt_n)\}$, where $dt = \sum_i dt_i$.
- When $p(dt)$ is executed at $x(t)$, a vehicle follows the *trajectory*: $\pi(x(t), p(dt))$.
- If a plan has a single control u applied for duration dt, then $\pi(x(t), (u, dt))$ is called a *primitive* trajectory.
- A *feasible* trajectory is collision-free and respects Eq. 3.1.
- A state on $\pi(x(t), p(dt))$ at time $t' \in [t : t + dt]$ is denoted as $x^{\pi}(\ t'\ )$.
- For *stable* states there exists a control, which retains the system in the same state:

$$x(t) \text{ is stable iff } \exists\ u\ s.t.\ \int_{dt} f(x(t), u)\ dt = 0$$

Given a state $x_0(t_0)$ and a goal region $\mathcal{X}_G \subset \mathcal{X}$, compute a plan $p(dt)$ so that the resulting trajectory $\pi(x_0(t_0), p(dt))$ is feasible and the end state $x^{\pi}(\ t_0 + dt\ ) \in \mathcal{X}_G$ is a stable state within the goal region.

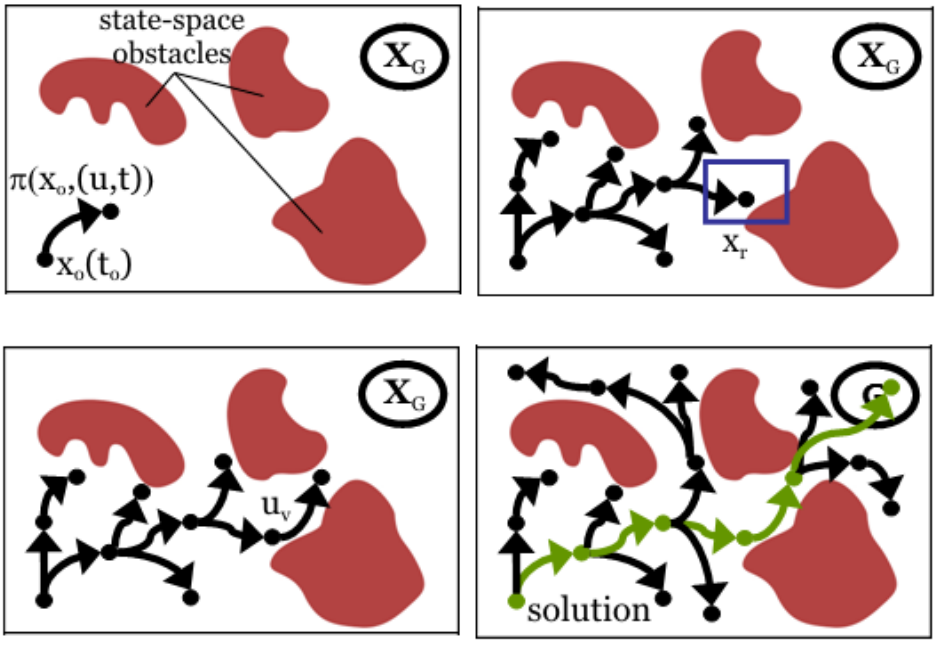

Figure 3.1 : The basic scheme for tree-based planning with sampling. Each iteration of the algorithm uses a selection/propagation step, where a reachable state along the tree is selected first and then a valid control is applied to produce a feasible trajectory.

3.2 Sampling-based Kinodynamic Planning

We first describe an abstract sampling-based approach for kinodynamic planning. The search operation is initiated at $x_0(t_0)$ and explores $\mathcal{X}_{free}$ by propagating feasible primitive trajectories. These trajectories are stored on a tree data structure $\mathcal{T}$ as edges. An edge that corresponds to trajectory $\pi(x(t), p(dt))$ is rooted at a node, which corresponds to the state $x(t)$. Fig. 3.1 and Algorithm 1 illustrate the operation of an abstract sampling-based kinodynamic tree planner.

Given the abstraction, we have the following three choices to construct a concrete algorithm. How to select: (i) the state x_r, (ii) the control u_v and (iii) the duration dt. Different algorithms follow different mechanisms for these choices but share the goal of covering the state space quickly and avoiding regression [KVdP06], which means propagating paths in already explored space.

Algorithm 1 BASIC SAMPLING-BASED TREE PLANNER

Set the root of $\mathcal{T}$ to the initial state $x_0(t_0)$
while $\nexists\, x \in \mathcal{T}$ *s.t.* ($x \in \mathcal{X}_G$ and x stable) **do**
  Select a reachable state $x_r \in \mathcal{T}$
  Select a valid control u_v for the state x_r
  Propagate the primitive trajectory $\pi(x_r, (u_v, dt))$ for
    a duration dt and for as long as it is feasible
  Add all the states along $\pi(x_r, (u_v, dt))$ in $\mathcal{T}$

The RRT algorithm [LK01b] samples a state in $\mathcal{X}$ and then selects the state $x_r \in \mathcal{T}$ closer to the sampled state given a metric. In Euclidean spaces, RRT has higher probability of extending a path inside the largest unexplored Voronoi cell. In kinodynamic planning, however, a good metric may not be available and the Voronoi-bias is not well defined. Similarly, the Expansive Spaces technique [HLM99] depends on an ideal sampler to guarantee coverage. PDST [LK05a, LK05b] avoids the use of a metric by using an adaptive subdivision scheme and provides probabilistic completeness for a general class of problems: if a path exists it will be eventually found [KKL96, LK04]. PDST biases the exploration towards larger cells of the subdivision, which correspond to relatively unexplored parts of the space. Less attention has been given in the literature to the choices regarding the control to be applied and the duration of propagation for kinodynamic problems.

The above planners can be viewed as continuous-space analogs of traditional uninformed search. Some informed variations in the literature exhibit a switching behavior between coverage planning and best-first search. The RRT-"goal bias" variant selects with certain probability to expand from the state which minimizes a distance metric to the goal and the rest of the time uses the RRT Voronoi bias for coverage [LK01b]. An older algorithm, the Randomized Potential Field (RPP) [BL91], also has multiple modes. It constructs a potential function to execute gradient-descent and then employs random walks and backtracking to exit local minima.

More recent informed variants introduce ideas from traditional AI search and study the effects of heuristics. The RRT* [US03] merges RRT with the A* algorithm, by using a heuristic in the state selection step. An issue, however, when using heuristics in continuous problems is the scale of the heuristic versus the true path cost. The randomized A* approach [DK07] uses learning in order to solve this scaling problem. The Anytime RRT algorithm [FS06] successively constructs RRTs that result in higher quality paths by employing heuristic search. The Exploring/Exploiting Tree (EET) [RBK07] emphasizes the need to balance the greedy and methodical aspects of search. EET uses potential functions to bias tree-based planners.

Critique

Although effective in eventually solving hard problems, sampling-based kinodynamic planners have high computational requirements as search methods, especially kinodynamic problems that are typically higher-dimensional compared to geometric ones. Moreover, they use metric information to guide the search and cover the entire state space. However, there are no good metrics for state spaces that involve kinodynamic constraints and as the dimensionality of the state space increases, the quality of the metric information deteriorates. This often results in regression, the algorithm re-explores already explored parts of the state space. Finally, as coverage-oriented algorithms, these planners exhibit an exhaustive flood-fill behavior and explore parts of the state space which are not necessarily helpful in solving specific queries. In this way, they can be viewed as sampling-based, continuous-space analogs of uninformed search.

Some of the approaches that improve upon the basic techniques, such as bi-directional or multi-directional tree expansion [PBC+05], are not general enough for kinodynamic planning, while others offer small improvements by simply switching between coverage orientated search and best-first search [LK01b] or depend on the proper setting of many parameters [BL91].

3.3 Informed Subdivision Tree (IST)

The IST algorithm follows the basic selection-propagation scheme of a sampling-based kinodynamic planners, as specified above, but attempts to provide a balance between methodical search and greedy search while not depending on many parameters. The overall architecture is shown in Fig. 3.2. This section describes the state selection and control propagation modules of the algorithm.

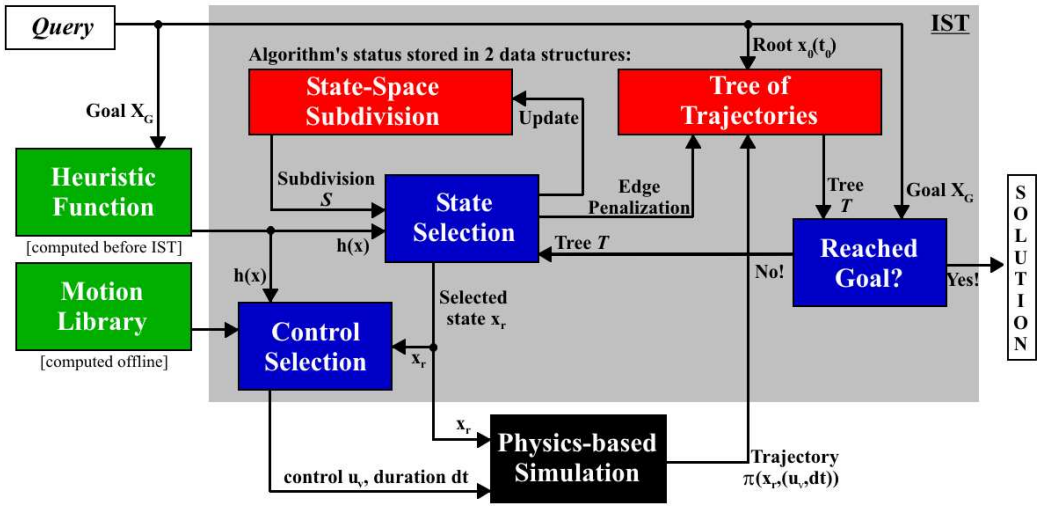

Figure 3.2 : IST expands a tree of trajectories, by selecting a state along the tree and a control. Physical simulation is used as a black box, where given these selections, the resulting trajectory is stored in the tree.

3.3.1 State Selection

The state selection step of IST is guided by the following, partially contradictory, objectives:

1. It aims to promote the selection of states that are closer to the goal (depth-first search behavior)

2. It aims to avoid reexplorating the same part of the state space (regression avoidance)

3. It aims to promote the selection of states that are connected with good quality paths with the root of the tree (path quality)

The first objective depends upon the existence of a heuristic function $h(q)$ that maps any state to a value $h : \mathcal{X} \rightarrow \mathbb{R}$ that is an estimate of how far away the state is from the goal region. To satisfy the second objective the `IST` algorithm employs an adaptive subdivision scheme of the state space. This scheme is responsible to locate which parts of the state space have been overly explored and vice verca so that `IST` can automatically adapt its selection process. The operation of the subdivision is explained in detail in the following paragraphs. Finally, for the third objective the algorithm maintains the cost from the root of the tree to the reachable states so as to promote the selection of states that are connected with shorter trajectories.

Overall the state selection step is a hierarchical procedure. The state is not selected directly. Instead, the algorithm selects first a cell of the state space subdivision, then an edge in this cell and finally a state along the edge. This is done for two reasons. Primarily because through a hierarchical selection procedure it is possible to take advantage of the subdivision's properties, which is able to adapt the size of its cells depending on how explored a subset of the state space is. The second reason is more practical. It is very costly to maintain all the bookkeeping information at the state level, so instead we store parameters that guide the exploration procedure at the cell and the edge level. In the following discussion, we will first provide details about the algorithmic tools that `IST` employs and then summarize the selection procedure.

Adaptive Subdivision for State Space Coverage

The subdivision data structure $\mathcal{S}$ corresponds to a set of cells: $\mathcal{S} = \{c_1, \ldots, c_K\}$. Each $c_i \subset \mathcal{X}$ corresponds to a subset of the state space. Initially this set contains one cell that encompasses the entire $\mathcal{X}$. The subdivision is refined in each iteration of the algorithm as shown in Fig. 3.3.

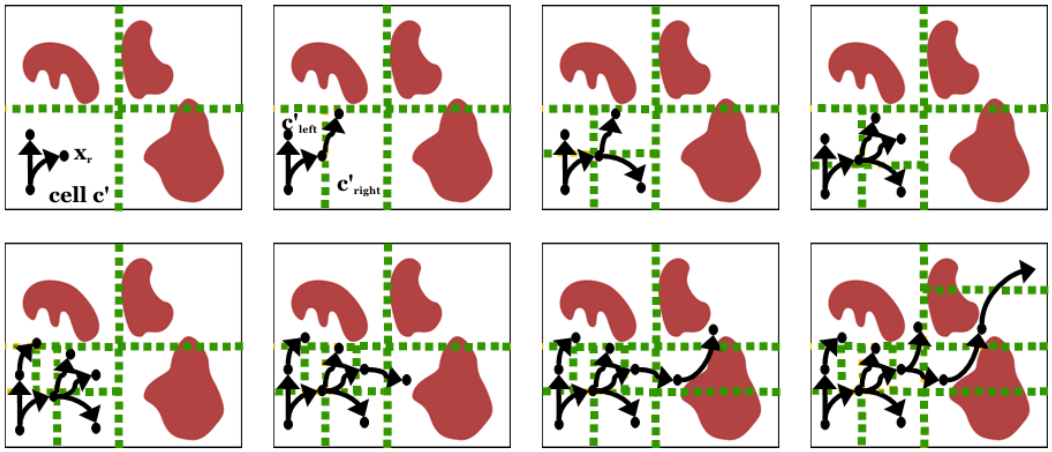

Figure 3.3 : The interaction between state selection and the subdivision data structure.

Whenever a reachable state x_r is selected, the cell $c' \in \mathcal{S}$ is found so that $x_r \in c'$. The algorithm guarantees that only one cell contains each state. Then c' is removed from $\mathcal{S}$ and two new cells, c'_{left} and c'_{right}, are introduced so that:

$$c'_{left} \cup c'_{right} = c' \quad \text{and} \quad c'_{left} \cap c'_{right} = 0 \tag{3.2}$$

Given that the initial cell is the entire $\mathcal{X}$, Eq. 3.2 implies that:

$$\cup_{\mathcal{S}} c_i = \mathcal{X} \quad \text{and} \quad \forall c_i, c_j \in \mathcal{S} : \; c_i \cap c_j = 0 \tag{3.3}$$

An implementation of the cell partition can be obtained with a Binary Space Partition Tree. Although not necessary, cell c' is typically split into two equal size cells along a dimension that maintains the subdivision balanced.

The purpose of $\mathcal{S}$ is to provide an online estimate of the coverage performance of the state space exploration. Large cells in $\mathcal{S}$ represent parts of the space that have not been explored as much as smaller cells, in the sense that the algorithm has not expanded often new trajectories from states within them. Consequently, the subdivision level of a cell $c \in \mathcal{S}$ (the number of subdivisions that occurred to create c) is used as an estimator $\mu(c)$ of the exploration value of the cell c.

Cells that do not contain any reachable states in $\mathcal{T}$ have by definition infinite estimator value: $\mu(c) = \infty$ if $\forall x \in \mathcal{T} : \nexists\, x \in c$. IST promotes the expansion of new trajectories from states that belong to cells with small $\mu(c)$ (large explored cells). This behavior promotes the exploration of unexplored parts of the space.

Although we have defined the subdivision to operate in the entire state space, this is not necessary for completeness purposes. It is sufficient to define a projection of the state space into a lower dimensional space and attempt to cover the projected space. Defining this projection depends on the application, allowing us to define the important projection of the state space that the algorithm must cover. For example, in the case of a second-order control car that has a five dimensional space: $\mathcal{X}_{car} = (x, y, \theta, V, s)$, ($(x, y)$: the planar coordinates, θ: orientation, V: velocity and s: steering direction) we typically subdivide only along the first three parameters of the state space.

Utilization of Heuristics

IST is able to incorporate a wide variety of heuristic information. The only requirement is that the heuristic is upper and lower bounded by finite positive values. For computational efficiency, it is also necessary that the computation of the heuristic to be fast.

Two typical examples of heuristics that can be used include the following:

- Distance-based goal bias: Ignore obstacles and use the distance between the state and the goal region given a metric for an obstacle-free version of the state space.

- Simplifying the system's geometry: It is possible to approximate the moving system by a simpler geometry (i.e., as a single point, or by using a bounding sphere), for which it is easy to compute a potential function given the obstacles in the environment (either in discretized or continuous form).

Morever, there is the case of having additional information about a planning problem in the form of a cost-map or an energy-map. A cost map scores different

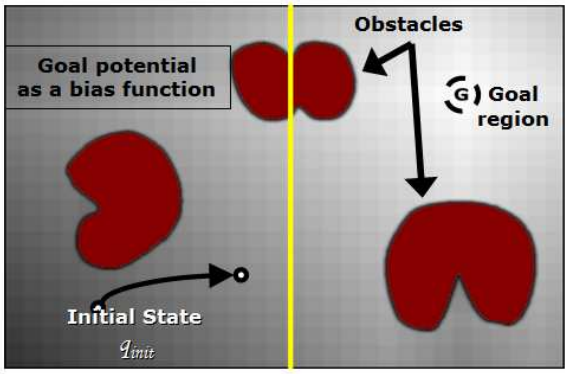

Figure 3.4 : An illustration of an abstract potential function.

regions of the environment depending on their traversability is often available for mobile robots in outdoor environments. In computational biology problems, as in protein-ligand interactions, there might be an energy function defined in the space. If such information is available it can be incorporated as a heuristic by integrating the cost over traversible paths.

There are two points of interaction between the heuristic function and the core `IST` algorithm, a preprocessing step and the online calls to compute the heuristic cost of particular states. During the preprocessing phase, a method is called to collect all the necessary information that will accelerate the online computation of the heuristic. For example, in the case of the second heuristic from the above list, a potential function can be computed for a point approximation of the moving system before the call to `IST`, as shown in Fig. 3.4. Then this potential can be used during the online computation of the heuristic.

There are two problems with the above mentioned heuristics. The quality of the first heuristic degrades as the environment becomes more occupied and complicated. On the other hand, the more complicated the system, the worse the quality of the second heuristic. In the next chapter, we will describe an approach for constructing a heuristic that is applicable to a wide set of problems, takes obstacles into account and has a higher level approximation for the system than the simplistic point-based approximation.

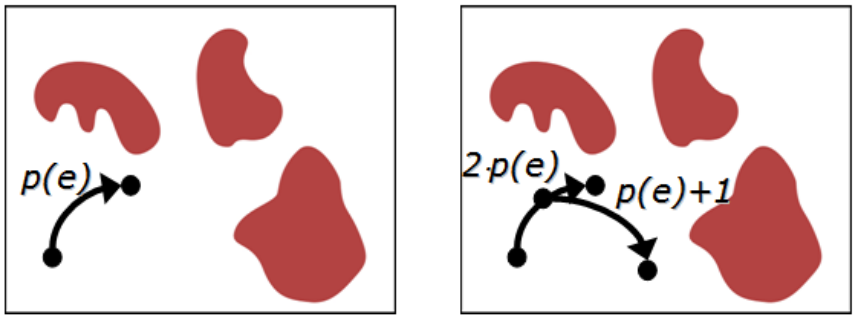

Figure 3.5 : The edge penalization scheme.

Edge Penalization Scheme

By promoting the selection of larger cells of the subdivision we avoid regression at the cell selection level. However, the regression problem manifests itself at the edge selection level as well. We must avoid selecting the same edge within in a particular subset of the state space because it has the best heuristic value compared to all its neighbors or the best path cost to the root of the tree.

To avoid regression at the edge level, we penalize edges of $\mathcal{T}$ when a state along the edge is selected as x_r. `IST` maintains a penalty factor $p(e)$ along each edge $e \in \mathcal{T}$. The first edge in the tree starts with a penalty of 1. Fig. 3.5 shows how the penalty factors are updated by the algorithm. When a state along e is selected then the penalty of the edge increases exponentially: $p(e) = 2 \cdot p(e)$. The penalty for the new edge being created will be: $p(e') = p(e) + 1$. `IST` promotes the selection of edges that have low penalty value. The objective of the penalization is to avoid selecting the same edges for expansion, which would result in regression.

Overall State Selection Step

Given the subdivision, the heuristic and the penalty scheme we can describe `IST`'s state selection step. Figure 3.6 illustrates two consecutive applications of the state selection step.

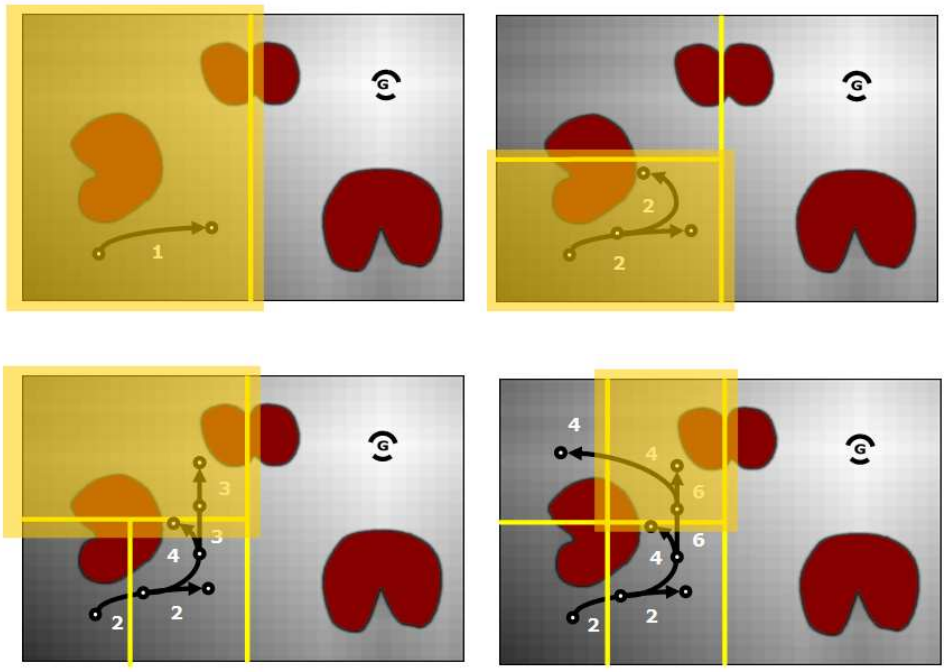

Figure 3.6 : Four consecutive steps from the operation of the proposed state selection technique. The highlighted cell corresponds to the selected cell c_{min} at each iteration. The numbers corresponds to the edge penalty values $p(e)$.

1. Select the cell $c_{min} \in \mathcal{S}$ that minimizes a score function:

$$score(c) = \mu(c) \ \cdot \ h(c), \tag{3.4}$$

   where $\mu(c)$ is the cell's subdivision level and the cell heuristic is: $h(c) = min\ _{\forall x \in \mathcal{T} \ s.t. \ x \in c}\ h(x)$.

2. Select an edge $e_{min} \in c_{min}$ that minimizes:

$$\texttt{score}(e) = p(e) \cdot cost(e), \tag{3.5}$$

   where $p(e)$ is the edge penalty and $cost(e)$ is the duration of the trajectory from x_0 until the last state on e.

3. Along the edge e_{min} select a random state $x_r \in e_{min}$.

4. Execute bookkeeping operations:

   (a) Split c_{min} according to the subdivision rules.

   (b) Split all the edges that belonged to c_{min} so that the invariant that every edge belongs to only one cell holds.

The important characteristic of the state selection is that it separates the heuristic $h(c)$ from the true path cost $cost(e)$. The heuristic is used to select a cell in the subdivision in a depth-first manner. The true path cost is used to select the edge within a cell in a breadth-first manner. In this way, we do not need to scale $h(c)$ vs. $cost(e)$. At the same time, the approach promotes the exploration of alternative parts of the state space by promoting the selection of large unexplored cells. The penalization scheme allows the eventual selection of every edge on the tree. Moreover, this algorithm still allows for an efficient implementation. The cell selection can be achieved with a binary heap and most of the information necessary to make decisions can be updated in constant time at each iteration of the algorithm.

3.3.2 Control Propagation

Given a selected x_r, the control selection step's objective is to estimate the best possible control to apply from x_r while allowing eventually the selection of any possible controls.

`IST` employs an offline procedure to create a database of trajectories for various dynamic parameters of the system's state. The offline procedure samples a discrete set of states $\hat{X}$ for the system. All these states are set to zero Cartesian coordinates and zero orientation. For each state $\hat{x} \in \hat{X}$ we sample various controls $u_{\hat{x}}$ and propagate a trajectory in an obstacle-free environment. We then store a discretized version of the resulting primitive trajectory $\pi(\hat{x}, (u_{\hat{x}}, dt_{max}))$ for a maximum duration dt_{max}.

During the online operation, if an edge is selected for the first time as e_{min} we make use of the database of trajectories with the objective of selecting a control

that will bring the system to a state with a smaller heuristic cost. We find the pre-stored state $\hat{x}$ that is closer to the state x_r in terms of the dynamics parameters. Then for each control $u_{\hat{x}}$ we transform the stored trajectory $\pi(\hat{x}, (u_{\hat{x}}, dt_{max}))$ so that the initial state is x_r without doing any collision checking. Out of all trajectories we find the one with the best heuristic value $h(x)$ and choose the corresponding control $u_{\hat{x}}$ as the u_v control selection for this iteration.

Every other time that an edge e is selected for expansion, we attempt to increase the variety of controls that have been expanded from e. `IST` stores on each edge the controls that have been expanded in the past from states along this edge and builds a discretized probability distribution. The probability distribution is used so as to bias towards the selection of controls that have not been selected in the past. It retains a non-zero probability, however, for every possible control.

The last point related to the propagation scheme is the duration dt of the resulting trajectory $\pi(x_r, (u_v, dt))$. By default, the duration of the trajectory has to be such so that the trajectory remains feasible throughout its execution. Nevertheless, `IST` imposes an additional constraint that aims towards improving the quality of the resulting paths. If the propagated trajectory $\pi(x_r, (u_v, dt))$ enters a new cell of the subdivision, other than the cell that x_r belongs to, where there is already another path with a better cost, then the propagation of $\pi(x_r, (u_v, dt))$ is stopped. The idea is that there is already a better path that reaches this part of the space, so this new trajectory might be unnecessary. Nevertheless, for probabilistic completeness purposes we do propagate a single state from the trajectory into this cell so that it can be potentially revisited by the algorithm in subsequent iterations.

Algorithm 2 summarizes the overall operation of `IST` as described in the previous sections. The algorithm assumes that the database of trajectories is available.

Algorithm 2 INFORMED SUBDIVISION TREE

`(Initialization)`
Set the root of $\mathcal{T}$ to the initial state $x_0(t_0)$
Create a cell c_0 that corresponds to the entire state space $\mathcal{X}$
Initiate the set of subdivision cells $S = \{c_0\}$
Execute a `PRM` for a kinematic version of the system

while $\nexists\ x \in \mathcal{T}$ so that $x \in \mathcal{X}_G$ (goal region) **do**
  `(State Selection)`
  Select the cell $c_{min} = argmin\ _{\forall c \in \mathcal{S}}\ \{score(c)\}$
  Select the edge $e_{min} = argmin\ _{\forall e \in c_{min}}\ \{score(e)\}$
  Update the penalty value: $p(e_{min}) = 2 \cdot p(e_{min})$
  Select a random state $x_r \in e_{min}$
  Split the cell c_{min} according to the subdivision rules
  Split the edges in c_{min} to the corresponding cells

  `(Control Selection)`
  if first time that e_{min} is selected **then**
    Find u_v that brings to a state with good heuristic value $h(\cdot)$ based on the trajectory database (no collision-checking)
  else
    Compute the valid control u_v based on the probability distribution of e_{min}
  Given u_v, update probability distribution of e_{min}

  `(Duration Selection)`
  while The resulting trajectory is feasible and
  there is no better path from x_0 to the current cell **do**
    Propagating for time dt

  Add the primitive trajectory $e_{new} = \pi(x_r, (u_v, dt))$ in $\mathcal{T}$
  Set the penalty parameter of $p(e_{new}) = p(e_{min}) + 1$

Discussion

The Informed Subdivision Algorithm described in this chapter focuses on solving kinodynamic planning problems faster by utilizing heuristic information, while providing a methodology that is also able to cover the entire state space and solve hard problems in case the heuristic is not able to properly guide the search. Chapter 7 provides experimental comparison between `IST` and other sampling-based kinodynamic planners. It is, however, obvious that the performance of

the algorithm depends heavily on the quality of the heuristic information. The following chapter proposes a general methodology for computing heuristics that can be employed by `IST`, which is more informative than many of the existing ways to bias kinodynamic planners.

Chapter 4

General Configuration Space Heuristic

A heuristic function $h(q)$ maps any state x to a real value: $h : \mathcal{X} \rightarrow \mathbb{R}$. This value can represent an estimate of the distance from x to the goal region X_G or more generally an estimate of how promising x is as the next state to be selected for propagation. `IST` is able to incorporate any heuristic information and provide probabilistic completeness as long as the heuristic is upper and lower bounded by finite positive values. For computational efficiency, it is also desirable that the heuristic satisfies the following objectives:

- It can be computed fast.
- Does not have high memory requirements.
- It is as informative as possible, so as to appropriately guide the exploration in the state space.

4.1 Roadmap Approach

For many planning problems it is possible to construct appropriate heuristics specifically designed for the task. For example, cost maps can be used to construct such heuristics for autonomous vehicles that operate in outdoor terrains []. In this section we are interested in designing an approach that is very general and can be applied to a variety of motion planning problems that involve both complex kinematic and dynamic constraints. The idea is to construct a heuristic given $\mathcal{C}$-space knowledge. In this way, the heuristic is more informative than many of the existing approaches that simplify the geometry of the moving system, often by approximating as a single point or by using a bounding volume like a sphere.

Given such $\mathcal{C}$-space heuristic, the `IST` algorithm has then deal with the dynamic aspects of the motion planning problem.

To achieve the above objectives we propose the construction of a roadmap $\mathcal{R}$ in $\mathcal{C}_{free}$, which is a graph $\mathcal{R}(V, E)$ whose nodes V are collision-free configurations. The edges of the graph depend on the definition of a symmetric local method that computes an admissible path $\mathcal{L}(q_i, q_j)$ connecting configurations q_i, q_j in the absence of obstacles. Consequently, $\mathcal{R}(V, E)$ has the typical roadmap properties [KSLO96]:

a. $q_i \in V$ iff $q_i \in \mathcal{C}_{free}$

b. $\forall\ q_i, q_j \in V :\ (q_i, q_j) \in E$ iff $\mathcal{L}(q_i, q_j) \in \mathcal{C}_{free}$

In order to be possible to use the roadmap as a heuristic, it is desirable for $\mathcal{R}$ to satisfy the following two properties:

1. **Visibility property:** $\forall\ q' \in \mathcal{C}_{free} :\ \exists\ q \in V$ so that $\mathcal{L}(q', q) \in \mathcal{C}_{free}$

2. **Path Deformation/Homotopy property:** For all collision free paths $\tau' \in \mathcal{C}_{free}$, there is also:

   - a path $\tau \in \mathcal{R}$ and
   - a continuous map $m : [0, 1] \times [0, 1] \rightarrow \mathcal{C}$ that deforms τ' to τ through collision-free space: $m(s, 0) = \tau'(s)$, $m(s, 1) = \tau(s)$ and $m(s, t) \in \mathcal{C}_{free}$ for all $s, t \in [0, 1]$.

These two properties express the requirements that the roadmap covers the entire $\mathcal{C}$ and is also able to truly represent the topological properties of $\mathcal{C}$.

Roadmaps that satisfy the first property are called "visibility roadmaps" [SLN00]. If we require the number of nodes in the roadmap to be optimum, then this problem is related to the well known and challenging art-gallery problem [GO97]. Even guaranteeing that a set of guards truly covers the entire $\mathcal{C}$ given a local method $\mathcal{L}$ is challenging, and a finite coverage may not always exist.

Roadmaps that satisfy the second property are able to capture all the homotopy classes of paths in the $\mathcal{C}$. A roadmap capturing the homotopic classes means that every valid path can be continuously deformed to a path of the roadmap. However, as it has been noted in the literature [NO04], capturing the homotopy classes in higher dimensions might not be sufficient to encode the set of representative paths since homotopic paths may be too hard to deform into each other. A roadmap is a good representation of the varieties of free paths if any path can be "easily" deformed into a path of the roadmap. The notion of simple path deformation varies in the literature [SBD$^+$02, JSss].

If we have a roadmap $\mathcal{R}(V, E)$ that satisfies the two properties then the computation of the heuristic value $h(x')$ of state x' is straightforward:

i. Extract the configuration q' that corresponds to x'

ii. Extract a configuration q_{goal} in the goal region X_G

iii. Compute nodes $q, q_g \in V$ so that $\mathcal{L}(q, q')$ and $\mathcal{L}(q_g, q_{goal}) \in \mathcal{C}_{free}$. Nodes q, q_g are guaranteed by the visibility property to exist.

iv. Find the roadmap path $\tau \in \mathcal{R}$ that connects q, q_g. This path is guaranteed by the second property to exist if there exists a path τ' that connects q' and q_{goal}. The second property also guarantees that τ and τ' are homotopic, so the first is a good approximation of the second, which allows the heuristic to be informative.

v. Compute the heuristic value $h(x') = d(q', q)+d(\tau)+d(q_g, q_{goal})$, where $d(\cdot, \cdot)$ is a distance value between configurations and $d(\cdot)$ is the length of a path.

Due to the complexity of constructing a roadmap that truly satisfies the visibility and deformation properties we will follow the dominant methodology in the literature and apply a probabilistic approach to compute a roadmap that approximates these properties. The focus will be on designing a technique that also satisfies the other objectives for a heuristic methodology, fast computation

and small memory requirements. In the next section, we will briefly outline two related works [SLN00, NO04] on which we built upon.

4.2 Foundations

The Visibility-based Probabilistic Roadmap Method (`Visib-PRM`) has been proposed as a general probabilistic approach for constructing a roadmap that aims to cover the entire $\mathcal{C}$-space with a small number of nodes [SLN00]. The roadmap is constructed incrementally by randomly sampling $\mathcal{C}$ and attempting to connect some pairs of collision-free samples with $\mathcal{L}$. When collision free configurations are found they are added to the roadmap: (i) if they cannot be connected to any existing node in the roadmap (i.e., they are visibility guards) or (ii) if they can be connected to at least two existing nodes that belong to two different connected components (i.e., they connect existing guards).

The `Visib-PRM` algorithm has many important advantages. It has a termination condition that depends upon the volume of free space currently covered by the roadmap. The algorithm is guaranteed to terminate for any parameter value M. When it stops, a probabilistic estimation of the percentage of free space not covered by the guards is $\frac{1}{M}$. This also means that path planning queries may succeed to connect configurations to the roadmap with a probability of: $1 - \frac{1}{M}$. A second advantage is that although the resulting roadmap tends to cover well $\mathcal{C}$, it is also very small and its size remains intrinsic to the complexity of $\mathcal{C}_{free}$. It is bounded by the maximal number of guards that cover the free space without mutual visibility.

Nevertheless, the technique has been criticized that it underperforms experimentally when compared against other probabilistic approaches for constructing roadmaps, which do not necessarily focus on covering the entire $\mathcal{C}$ [GO02]. It can be argued that the technique, in order to retain a small roadmap, unnecessarily ignores many candidate nodes for which it has invested computation time to check whether they are connected to existing nodes of $\mathcal{R}$.

Algorithm 3 `Visib-PRM`(M)

$V \leftarrow 0$; $E \leftarrow 0$; initialize $\mathcal{R}(V, E)$
$ntry \leftarrow 0$
while $ntry < M$ **do**
{
  `(Configuration sampling)`
  Select a random free configuration q
  $Visib\_set \leftarrow$ empty;

  `(Computing` q`'s visibility)`
  for all nodes $g \in V$ **do**
    if $Visib\_set$ does not contain nodes from g's connected component and $\mathcal{L}(q, g) \in \mathcal{C}_{free}$ **then**
      Add $\{g\}$ to $Visib\_set$

  `(Node addition)`
  if $Visib\_set$ is empty `(`q `is a guard)`
  or $Visib\_set$ has nodes from different components **then**
  {
    Add $\{q\}$ to V

    `(Edge addition)`
    for all $g \in Visib\_set$ **do**
      Add (q, g) to E and merge connected components
  }
  else
    ntry $\leftarrow$ ntry + 1
}
Return $\mathcal{R}(V, E)$

It must be noted that the `Visib-PRM` does not address the path deformation property. It can be used, however, as the starting point to construct such a roadmap, where additional nodes are added when it is detected that homotopy classes of paths are not already represented by $\mathcal{R}$ [JSss]. Obviously, the overall technique is computationally more expensive than the `Visib-PRM` algorithm since it first requires a call to the algorithm.

There are alternative ways to enhance the path diversity of the roadmap, which however relax the probabilistic guarantees that can be provided from path

deformation roadmaps. For example, the easiest way is to connect every configuration to all of the configurations that it can be connected with. While this method does enhance the path diversity of the roadmap, most connections are redundant. In the context of the basic PRM algorithm it also increases the running time of the algorithm. A method, however, has been proposed that adds connections between nodes that are considered useful according to the following definition [NO04]:

Useful edge: Let q' be a configuration that is a candidate for adding to $\mathcal{R}$, and q an existing node of $\mathcal{R}$. The distance $d(q', q)$ is the $\mathcal{C}$ distance between q' and q. The graph distance between q' and q is $G(q', q)$ (the length of the shortest path in $\mathcal{R}$ from q' to q). If there is no path from q' to q, then $G(q', q) = \infty$. The edge (q', q) is K-useful edge if:

$$K \cdot d(q', q) < G(q', q) \tag{4.1}$$

A small value of K adds more cycles (if $K < 1$, all edges are allowed), a large value of K adds less edges (if $K = \infty$, no cycles are allowed and the graph is a forest). The A^* algorithm can be used to compute the graph distance $G(q', q)$. It can also be shown that when using this technique to define which edges should be connected, the path length found by the algorithm converges to K times the optimal path length. In the next section, we will describe the technique which we propose, which builds upon the existing Visib-PRM algorithm and the definition of a useful edge and aims to provide considerable computational advantages.

4.3 Efficient Visibility Roadmap with Cycles (EVRC)

We propose an approach that attempts to provide a tradeoff between increasing the size of the visibility roadmap and reducing the time it takes to construct it. The algorithm attempts to directly include edges that enhance the path diversity of the roadmap by using the useful edge criterion as the roadmap is being

constructed. Furthermore, it also aims to minimize wasting computations by ignoring configurations for which many collision checking calls have been made. We will refer to this method as Efficient Visibility Roadmap with Cycles (`EVRC`). We describe here the basic differences of the proposed approach from the `Visib-PRM` algorithm and how the idea of useful cycles is incorporated in the overall technique.

Reduced Visibility Region: The original `Visib-PRM` allowed the visibility of a node to extend to infinity. Thus, when a configuration q is considered for addition, all local paths $\mathcal{L}(q, g)$ for all existing nodes $g \in V$ are checked. Although `Visib-PRM` retains a small set of nodes, this is still an expensive operation given that the probability of a successful connection decreases as the distance between two configurations increases. That is why we bound the visibility region of a node by a distance threshold vis_{max}. Only the local paths $\mathcal{L}(q, g)$ for which $d(q, g) < vis_{max}$ and $g \in V$ are checked when configuration q is considered for addition.

The benefit of this change stems from the fact that for each candidate configuration we are checking now only a small set of possible connections. For example, in completely obstacle free two-dimensional spaces, the maximum number of possible connections is 6 and this number increases as the complexity of the $\mathcal{C}$-space increases. The disadvantage is that there is no straightforward way of computing an optimum value for vis_{max}. The lower the value of vis_{max}, the smaller the percentage of the existing nodes we will check for possible connections but the denser the resulting roadmap. On the other hand, larger values for vis_{max} result in small size roadmaps. The computational advantages are typically maximized somewhere in between and this value is a function of the complexity of the environment.

Node addition criterion: The `Visib-PRM` allows the addition of a configuration q as a node in the roadmap under two circumstances, if q is a guard or if q

connected two separate connected componets of the roadmap. In EVRC, we allow the addition of the node in one additional case:

- If the configuration q can be connected to two nodes q_i and q_j from the same component and:

$$K \cdot [\ d(q_i, q) + d(q, q_j)\ ] < G(q_i, q_j) \tag{4.2}$$

This case is related to the useful edge criterion. The addition of q as a node allows the creation of a useful cycle.

Edge addition criterion: Visib-PRM added one edge between the candidate configuration q and each connected component that it could be connected with. This ensures that the graph retains a forest structure. Instead, in EVRC, the useful edge criterion is applied. All possible connections with nodes within the threshold vis_{max} are considered and an edge is added if it satisfies Eq. 4.1. In this way, we do not aim to first construct a visibility roadmap that is a forest and then enhance it with useful paths to represent all the homotopy classes. As we build the roadmap we attempt to directly identify the useful cycles.

Configuration sampling: In EVRC, the sampling of the candidate node is not completely random as in Visib-PRM. Instead, the configuration sampling procedure is broken into two phases. During the first phase, the algorithm focuses on producing guards and covering the space. Thus, the algorithm promotes the selection of configurations that are beyond the distance threshold vis_{max} from their nearest neighbor in the roadmap. During the second phase, the algorithm reverts to random $\mathcal{C}$-sampling. Switching between the two phases depends on a stopping criterion that is similar to the stopping criterion for the overall algorithm. If we can no longer add guards after N attempts ($N < M$) we move from the first phase to the second.

Minimize wasted computations: Finally, `EVRC` does not completely ignore the nodes that are not added to the roadmap. In fact, it retains a secondary data structure $\mathcal{R}'(V', E')$, where the nodes are all the configurations that failed the addition test. The edges of $\mathcal{R}'$ correspond to connections between nodes q' in V' and nodes $q \in V$ of the basic roadmap $\mathcal{R}$. In this way, `EVRC` actually creates a hierarchy, where the guard and bridge nodes in $\mathcal{R}(V, E)$ define a skeleton data structure of the $\mathcal{C}$-space and the rest of the configurations are stored in $\mathcal{R}'$ are linked to this skeleton.

One advantage of retaining $\mathcal{R}'$ is that we can use these secondary nodes during the query phase. We will explain more about this advantage in the next section. Most importantly, however, we can also reconsider the nodes in $\mathcal{R}'$ for addition in the basic roadmap $\mathcal{R}$ even after they have failed the node addition criterion the first time. Although these nodes where not deemed beneficial in the past, through the addition of a new node they might become useful by allowing the connection of two disconnected components.

Thus, when a new configuration q is added to the basic roadmap, we revisit all the nodes q' in $\mathcal{R}'$ for which: $d(q, q') < vis_{max}$ and check whether they satisfy now one of the node addition criteria. Since the nodes are assumed to have limited visibility (vis_{max}), the number of secondary nodes checked remains low. The advantage of revisiting the secondary configurations over just considering new randomly produced nodes is the fact that we have already computed the connection properties of these configurations with the existing nodes on the roadmap. Before actually executing any collision checking for connecting q with q', we first check whether the addition of q' would pass the node addition criterion assuming that the edge (q, q') is collision free. The number of local paths checked for collision during this step is bounded by the number of secondary nodes within the visibility limit from configuration q. As a heuristic we first check those node q' with maximum distance from q (always bounded by vis_{max}) for addition to the roadmap.

Algorithm 4 `EVRC`(M, N, vis_{max})

$V \leftarrow 0$; $E \leftarrow 0$; initialize $\mathcal{R}(V, E)$
$V' \leftarrow 0$; $E' \leftarrow 0$; initialize $\mathcal{R}'(V', E')$
$ntry \leftarrow 0$; $sample\_ntry \leftarrow 0$
$phase \leftarrow$ GUARD_PHASE;
while $ntry < M$ **do**
{
  `(Configuration sampling)`
  $q \leftarrow \mathcal{C}$-SAMPLING($\mathcal{R}, phase, sample\_ntry$)
  $Visib\_set \leftarrow$ empty;

  `(Computing` q`'s visibility)`
  for all nodes $g \in V$ so that $d(q, g) < vis_{max}$ **do**
    if $\mathcal{L}(q, g) \in \mathcal{C}_{free}$ **then**
      Add $\{g\}$ to $Visib\_set$

  `(Node and edge addition in basic` $\mathcal{R}$`)`
  NODE-ADDITION($q, Visib\_set, \mathcal{R}, \mathcal{R}'$)

  `(Add to secondary structure` $\mathcal{R}'$`)`
  if q not added in V **then**
  {
    Add $\{q\}$ to V'
    for all $g \in Visib\_set$ **do**
      Add (q, g) to E'
    ntry $\leftarrow$ ntry + 1
  }
}
Return $\mathcal{R}(V, E)$, $\mathcal{R}'(V', E')$

Overall algorithm: `EVRC`'s operation is described in Alg. 4. The algorithms first initializes the basic and secondary graph data structures, sets the configuration sampling phase to GUARD_PHASE and the counters $ntry = sample\_ntry = 0$. The first counter ($ntry$) expresses how many failed attempts have been made to add a configuration in the roadmap. If the number of failed attempts passes the parameter M, then the algorithm terminates. The second counter ($sample\_ntry$) is used only in the GUARD_PHASE of the $\mathcal{C}$-SAMPLING procedure to measure

the failed attempts to produce a candidate guard node given only distance information. If the value *sample_ntry* exceeds the parameter N, then the sampling function progresses into the second phase: RANDOM_PHASE and produces random nodes.

Algorithm 5 $\mathcal{C}$-SAMPLING($\mathcal{R}, phase, sample\_ntry, N$)

```
if phase == GUARD_PHASE then
{
  q ← GUARD-SAMPLING( R, N, &found_guard )
  if found_guard == true then
    sample_ntry = 0
  else
    sample_ntry ← sample_ntry + 1

  if sample_ntry >= N then
    phase = RANDOM_PHASE
}
if phase == RANDOM_PHASE then
  q ← sample C_free
Return q
```

Algorithm 6 GUARD-SAMPLING($\mathcal{R}, N, found\_guard$)

```
found_guard ← false
ntry ← 0
while found_guard == false and ntry < N do
{
  q ← sample C_free
  g ← closest node in R to q
  if d(q, g) < vis_max then
    ntry ← ntry + 1
  else
    found_guard = true
}
```

With the production of a candidate configuration q by the $\mathcal{C}$-SAMPLING procedure, the set $Visib\_set$ is initialized to an empty set. All the nodes in the existing roadmap $\mathcal{R}$ that are within distance vis_{max} from q, and which are connected

Algorithm 7 NODE-ADDITION($q, set, \mathcal{R}, \mathcal{R}'$)

```
(Node addition criterion)
if  set is empty (q is a guard)
or set has nodes from different components
or exist nodes in set so as to satisfy Eq. 4.2 then
   Add {q} to V

if  q is added in V  then
{
   (Edge addition)
   for all g ∈ set do
      if (q, g) is a useful edge according to Eq. 4.1 then
         Add (q, g) to E

   (Revisit secondary nodes)
   for all g' ∈ V' so that d(q, g') < vis_max do
      if L(q, g') ∈ C_free then
      {
         Add (q, g') to E'
         NODE-ADDITION( g, E'(g'), R, R' )
         if g is added to V then
            Remove g from V' and remove E'(g) from E'
      }
}
```

with a collision free path are added to the set $Visib\_set$. Then the procedure we NODE-ADDITION is called. This method first enforces the node addition criterion. If configuration q is a guard or connects two separate connected components or adds a useful cycle then it is added in $\mathcal{R}$. After the addition of the configuration as a node, we have to check which edges to be added. We use the useful edge criterion to determine which edges between q and the nodes in the $Visib_set$ to add. Since q has been added to the roadmap, we now move on to check whether this addition triggers also the addition of a configuration from the secondary data structure $\mathcal{R}'$. For all the nodes q' in $\mathcal{R}'$ that are within the threshold distance vis_{max} and which are connected to q, we call recursively the NODE-ADDITION function to check whether their addition enhances the

roadmap. This time NODE-ADDITION is called with the set $E'(q')$ as the set of candidate edges.

When the control returns to the top level EVRC method and in the case that the configuration q was not deemed useful to be added in the basic roadmap $\mathcal{R}$, it is added in the secondary structure $\mathcal{R}'$. Also for all the nodes in $Visib_set$, we add the edges that connect these nodes to q in the set E'.

4.4 Fast Metrics using the EVRC Roadmaps

We have already outlined in section 4.1, how to compute a heuristic estimate $h(x')$ for a state x' given a roadmap. There is, however, an additional challenge in actually applying the approach outlined in that section. Step (iii) specifies that given configurations q' and q_{goal}, the approach computes nodes $q, q_g \in V$ so that $\mathcal{L}(q, q')$ and $\mathcal{L}(q_g, q_{goal}) \in \mathcal{C}_{free}$. This step requires to execute additional collision checking steps during the online computation of the metric function and after the computation of the roadmap. This considerably increases the cost of computing such a metric function and undermines the utility of the overall approach.

A simple alternative is to avoid executing any collision checking and return the configurations $q, q_g \in V$ that are closer to the q' and q_{goal} given a $\mathcal{C}$-metric. This obviously decreases the computational overhead of computing the estimate $h(x')$ but it often results in grossly incorrect estimates. This problem is further aggravated by the fact that the roadmap $\mathcal{R}(V, E)$ is a relatively sparse roadmap.

This is a point where the secondary roadmap $\mathcal{R}'(V', E')$ can prove useful. It contains all those collision free configurations that where tested during the roadmap construction but where not added to the roadmap and we know their connectivity properties with the existing properties. So, we can also use $\mathcal{R}'(V', E')$ when checking for the closest nodes to configurations q' and q_{goal}. The combination of $\mathcal{R}'(V', E')$ and $\mathcal{R}'(V', E')$ correspond to a much more dense sampling of the $\mathcal{C}$-space.

Overall the technique for computing $h(x')$ given a roadmap computed by EVRC

is described in Alg. 8. Here we assume that we have already found node $q_g \in V$ so that $\mathcal{L}(q_g, q_{goal}) \in \mathcal{C}_{free}$. Finding this roadmap node has to be executed only once for all metric computations.

If the collision checking package is able to return distance information between the moving body and the closest obstacle, then this information can be also part of the heuristic cost by penalizing states that are closer to obstacles in a potential function like manner.

Algorithm 8 ROADMAP-METRIC $(x', \mathcal{R}, \mathcal{R}', q_g)$

$q' \leftarrow$ configuration of state x'
$q \leftarrow argmin_{q \in V \text{ or } V'}(d(q, q'))$
$\tau \leftarrow$ path in $\mathcal{R}$ and $\mathcal{R}'$ that connects q and q_g
Return $h(x') = d(q', q) + d(\tau)$

Discussion

The important advantages of the roadmap-based heuristic over simpler distance metrics is that it considers both workspace obstacles and the complete $\mathcal{C}$-space representation of the moving systems. In this way the approach computes paths that are truly collision-free in the $\mathcal{C}$.

Its drawback is that it is more expensive to compute. In our experiments we have seen improvements even if we use very simple heuristics that consider workspace obstacles (i.e. a wavefront function on a grid for point approximations of the true system). Moreover, on all of our experiments, the computation of the roadmap, which occurs before the construction of $\mathcal{T}$, is orders of magnitude faster than the computation of $\mathcal{T}$ itself, since the kinematic reduction of a kinodynamic problem is considerably less constrained.

Chapter 5

Safe Replanning for Systems with Drift

Realistic autonomous systems and agents have only partial information about their environment. Partial observability requires interleaving sensing, planning and execution, where a planner is called frequently and has finite time to replan a trajectory [BV06, FKS06, BFK06]. Moreover, systems like vehicles exhibit kinodynamic constraints that restrict their motion, which must be accounted for at the motion planning stage. In this chapter, a replanning framework is presented, called Short-Term Safety Replanning (`STSR`), that respects such constraints and generates safe paths under finite computation times. Safety means that the vehicle does not collide with obstacles at least for applications with static obstacles even when the system's workspace model is being updated dynamically. This is an important consideration when replanning for real vehicles as a collision-free trajectory may bring the system close to an obstacle with high-velocity and no maneuver to avoid collisions [FA04, FDF02].

For replanning applications, the computational performance of planning is also of primary importance. A slow approach delays taking into account new sensing information and the vehicle does not react on time to changes in the workspace. One way to speed up performance in replanning is to reuse computations between consecutive planning cycles. Nevertheless, a computational concern, given the presence of dynamics, is the increased cost of collision checking in order to provide the safety guarantees. `STSR` identifies the minimum set of states that have to be checked for safety; reducing in this way the overhead of providing safety guarantees. The proposed framework has also been integrated with the `IST` algorithm from chapter 3.

This chapter finally describes the application of the planner to a task that

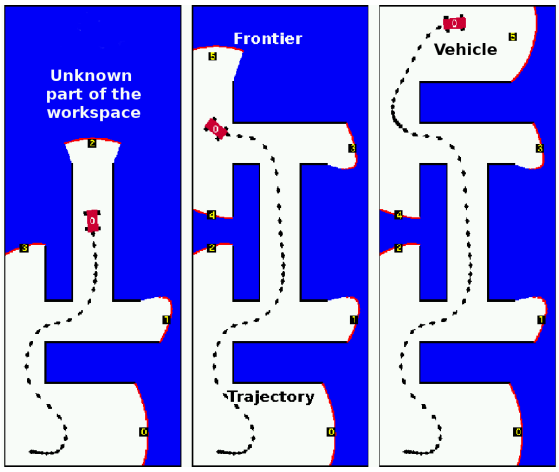

Figure 5.1 : Mapping an unknown space with an acceleration controlled car.

requires replanning: mapping of unknown environments. Initially the vehicle knows only a small part of the space but it must eventually cover the entire workspace. Fig. 5.1 shows an example from our simulations with a car-like robot with bounded acceleration.

5.1 Replanning Formulation

We assume in this section, similar to chapter 3, a a moving system, such as a non-holonomic vehicle, whose motion is governed by state update equation: $\dot{x} = f(x, u)$ and $g(x, \dot{x}) \leq 0$, where $x \in \mathcal{X}$ is a state, u is a control and f, g are smooth. The metric $\rho(q_1, q_2)$ is defined in the state space $\mathcal{X}$, where $q_1, q_2 \in \mathcal{X}$. The robot is in a workspace $\mathcal{W}$, equipped with a sensor of limited range. The robot uses its sensors to update a dynamic workspace representation $s(\mathcal{W}, t)$. A state $x \in \mathcal{X}_{free}$ is considered collision-free at time t if it places the robot chassis in the known collision-free part of $s(\mathcal{W}, t)$.

Replanning is necessary for multiple tasks where the workspace changes dynamically. We focus on the case the system does not know anything about the workspace and must cover it in order to build a map (workspace exploration). A similar dynamic task not covered here due to space limitations is planning

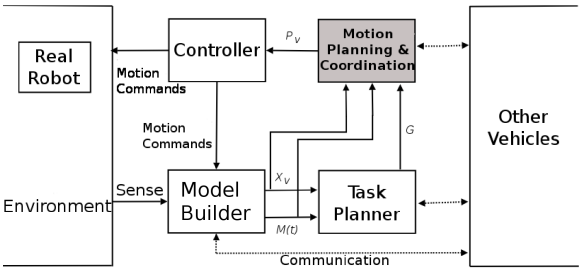

Figure 5.2 : The closed loop architecture and modules on a single vehicle.

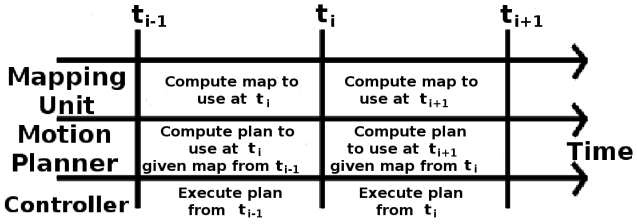

Figure 5.3 : The robot's synchronization scheme.

among dynamic obstacles. The high-level approach for such replanning tasks is to break them into a sequence of smaller planning problems. To achieve this, we assume that the mapping and estimation unit, the planner, as well as the low level motion controller are synchronized, as shown in Fig. 5.2 and 5.3. Then, for the planning cycle $(t_{i-1} : t_i)$, the following steps are executed:

- A representation $s(\mathcal{W}, t_{i-1})$ built in the previous cycle.
- A goal region $\mathcal{X}_{goal} \subset \mathcal{X}$ is defined for the current cycle.
- A motion planner computes a plan.
- The plan will be executed during cycle $(t_i : t_{i+1})$.

The problem that must be solved then is the following:

Replanning under Kinodynamic Constraints:

Compute plan $p(t)$ to execute during $(t_i : t_{i+1})$ that produces $\pi(x, p(t))$ which is: (a) collision-free, (b) leads to $\mathcal{X}_{goal}(t_{i-1})$, (c) minimizes the distance traveled by the robot.

5.2 Inevitable Collision States

The fact that $\pi(x, p(t))$ is collision-free does not guarantee safety into the future, since $\pi(x, p(t))$ may lead to an Inevitable Collision State (`ICS`) [FA04].

Definition: A state x' is an `ICS` if there is no trajectory $\pi(x', p(\infty))$ which is collision-free.

From the above definition it is obvious that it is computationally infeasible to compute whether a state x' is `ICS` or not, since:

- It requires checking all possible trajectories out of x', which is an infinite number for the problems we are considering
- It requires integrating the trajectories into infinity, which in most cases is impossible
- In the case of dynamic scenes it is impossible to know the motions of moving of moving obstacles, especially further into the future

In order to deal with the problem we are making the following concessions. We are going to treat differently replanning tasks with static obstacles from the problems that involve moving ones. The first category includes tasks such as: (i) planning in a workspace with unexpected static obstacles and changes (i.e., doors being closed or open), (ii) workspace exploration and mapping, which is the application that we focus on in this chapter and (iii) monitoring a static scene for events that do not effect the safety of the autonomous observer. In the second category problems include: (i) planning among moving obstacles, (ii) pursuit-evasion and target tracking.

For problems with static obstacles, we do retain the definition of an `ICS`, as it is given above. For problems with moving obstacles, the definition is altered so that it is sufficient if trajectories out of state x' to be collision free for a predefined period τ: $\pi(x', p(\tau))$ is collision-free. For this predefined amount of time τ it is necessary to make assumptions about the movement of moving obstacles. For

example, we can make worse case assumptions and consider all the space that could be potentially occupied by a moving obstacle within the τ period as an obstacle. Or we can use some sort of prediction in order to guess where the obstacles will be located. Overall, however, we have to integrate paths only for a finite predefined amount of time τ into the future but we cannot guarantee the overall safety of the system. The safety in the case of moving obstacles depends: (i) on the behavior of the moving obstacles (i.e., are they passive obstacles or do they attempt to approach the moving system?), (ii) the relative velocities and velocity bounds between the moving system and moving obstacles, (iii) the effectiveness of the prediction module in correctly estimating the future motion of dynamic obstacles and (iv) the effectiveness of the planning algorithm that runs on the moving system in searching for a diverse set of paths out of state x'.

Still, in the case of static obstacles we have to deal with the issues of infinite trajectory integration and infinite number of trajectories. It is much faster, however, to check whether state x' belongs to a superset of the `ICS` set by taking a conservative approach. The conservative approach requires using only a small set of "contingency" plans $\Gamma(x')$ and define a state x' to be **unsafe** iff:

$$\text{For a predefined set of plans } \Gamma(x'):$$

$$\nexists \text{ plan } \gamma \in \Gamma(x') \text{ s.t.: } \pi(x', \gamma(\infty)) \text{ is collision free.} \tag{5.1}$$

As the above notation shows, the set of contingency plans depends on the state x'.

To address the complication of infinite trajectory integration, we can take advantage of the properties of a large subcategory of systems with drift, systems that exhibit "diminishing drift". This type of moving systems have the capability to apply controls that "diminish" their drift, that bring their acceleration and velocity parameters to zero. For example, real automobiles and vehicles, as most real practical systems, belong in this category because it is possible to break until the system comes to a complete stop. When a system comes to a complete stop in an environment with static obstacles there is no reason to further integrate the

path into the future, because there is nothing that can possibly effect the safety of the system. Instead we have to integrate the path only up to the point that the velocity of the system becomes zero. This means that in the case of systems with "diminishing drift", such as automobiles, the set $\Gamma(x')$ of "contingency plans" can correspond to breaking maneuvers that bring the system to a complete stop. The time it takes to execute the contingency plan is unrelated to the duration of the planning cycle, instead it depends on the state x'.

For systems with non-diminishing drift, such as airplanes that have to remain on the air, we have to resort to a different set of contingency plans. In this case, we can apply controls that will force the system to visit the same part of the space again and again. For example, a maneuver that makes the system to repeat the same circular motion is also sufficient to address the problem of infinite trajectory integration. Once the system executes the circle safely in a static environment then it can execute the same maneuver safely into infinity. We refer to such maneuvers as "looping maneuvers". Actually, a breaking maneuver that brings a system to a complete stop can be seen as the trivial case of a looping maneuver, where the loop corresponds to a single state.

Regardless of whether we are applying state revisiting maneuvers or breaking maneuvers, however, we have at this point a way of inferring the safety of a state. We depend on an oracle that uses a set of predefined maneuvers to find whether the system can be safely brought to a safe state or a safe sequence of states. The oracle makes heavy use of collision checking. This means that it can be orders of magnitude more expensive to check whether a state is safe compared to checking whether it is collision-free or not. Thus, it is very important in a replanning framework to minimize the number of states that have to be checked for safety in order to guarantee the safety of the system. This is the objective of the `STSR` framework that we describe in the next section.

5.3 Short-Term Safety Replanning

In order to guarantee overall safety for a system it is necessary to guarantee that the replanning approach is able to find a collision-free trajectory on every replanning cycle. Given the discussion on `ICS`, this implies that the replanning approach must provide the following invariant:

Invariant: For each replanning cycle $(t_i : t_{i+1})$ the system selects a plan $p(t_i : t_{i+1})$ which when executed at state $x(t_i)$:

a. The resulting trajectory $\pi(x(t_i), p(t_i : t_{i+1}))$ is collision-free.

c. It leads to state $x^{\pi}(t_{i+1})$ that is safe according to Eq. 5.1.

Our objective is now to design a technique that takes advantage of the efficiency of sampling-based kinodynamic planners, such as `ICS`, in the context of replanning and provide the above safety invariant. The problem is that the planner might fail to compute a safe trajectory given the time limitations imposed. Consequently, we are interested in building a tree $\mathcal{T}$ during each cycle and somehow always finding a trajectory that leads to a safe state, either by extracting it from $\mathcal{T}$ or through another way. We can achieve this objective through the following strategy:

1. Use a sampling-based kinodynamic tree $\mathcal{T}$ to compute candidate trajectories

2. Check which trajectories are safe on $\mathcal{T}$

3. Out of the safe trajectories select one that best promotes the execution of the task

4. If there is no safe trajectory found by $\mathcal{T}$ use the contingency plan $\gamma \in \Gamma(x(t_i))$ that was used to prove that state $x(t_i))$ was safe during the previous cycle

The last step in the above strategy provides safety in a recursive manner. Note that the above strategy does not specify how trajectories along $\mathcal{T}$ are checked for safety. This can considerably influence the computational cost of providing safety. Note, however, the following property of safe states:

Property: All the states along a trajectory leading to a safe state are also safe.

We can take advantage of the above property in order to check only a subset of states along the tree for safety. One idea is that it is sufficient to check the leaf and branch nodes of a tree. Fraichard et al. [PF05] and Frazolli et al. [FDF02] used this idea to reduce the overhead of safety checking.

Nevertheless, we can do better than that. The Short-Term Safety Replanning framework that we propose in this work has the following objectives:

- To check states that effect the safety of the system only during the next cycle $(t_i : t_{i+1})$.
- Not to explore parts of the space into the future (past time t_{i+1})) that do not extend from safe trajectories during the next planning cycle $(t_i : t_{i+1})$.

The framework takes advantage of the fact that it is possible to know or control the duration of the next planning cycle $(t_i : t_{i+1})$. This fact together with the property above means that in order for the trajectory followed during the next cycle: $\pi(x(t_i), p(t_i : t_{i+1}))$ to be safe, it is suffiecient to check only one state: $x^{\pi}(t_{i+1}$, the state at the end of the planning cycle.

Theorem: Assume a non-holonomic system with drift executing a replanning task in a static environment. The duration of the next planning cycle is T (from time t_i to time t_{i+1}. In order to guarantee safety it is sufficient to produce trajectories that are safe only for duration T, given contingency plans that are breaking or looping maneuvers.

Proof Sketch: Assume the vehicle is safe at state $x(t_i)$ at time t_i. This means that there is a breaking or looping maneuver $\gamma \in \Gamma(x(t_i))$ so that $\pi(x(t_i), \gamma(\infty)$ is

safe. The question is whether reaching an ICS can be avoided while replanning. There are two cases: (a) The planner will either succeed in producing a new safe plan for duration T and at time t_{i+1} the robot will end up in a situation when the state $x(t_{i+1})$ is safe. (b) The planner fails to compute a plan for the next period. However, we can execute the safe plan $\gamma \in \Gamma(x(t_i))$ for at least the next cycle of duration T (actually for infinite time, but we need only this part of the plan). The resulting state $x^{\gamma}(t_{i+1})$. So, in every case there is a safe plan. $\square$

Consequently, the strategy specifies that a planner should check for safety only those states that occur at time t_{i+1}. If a trajectory $\pi_r(x_r, pi(t))$ that is being propagated in a single step of a sampling-based kinodynamic planner intersects time t_{i+1}, then state $x^{\pi_r}(t_{i+1})$ is checked for safety. If $x^{\pi_r}(t_{i+1})$ is safe according to the safety oracle (using the breaking or looping maneuvers), then the corresponding edge is added to the tree. Otherwise, the trajectory is not propagated further than the state $x^{\pi_r}(t_{i+1})$, so that all the trajectories stored in $\mathcal{T}$ are safe at least for time T. Moreover, the state $x^{\pi_r}(t_{i+1})$ is never considered as a candidate state for selection to propagate new trajectories into the future.

We can summarize the STSR framework with the following algorithmic description. This description builds upon the abstract framework for sampling-based kinodynamic planners. In this presentation of STSR, we assumed that the duration of the planning cycle is provided as input to the algorithm and it remains the same between different planning cycles. Thus the time that is available to STSR to plan for is the same as the duration of the cycle that is planning for. The algorithm also receives as an input the maximum duration for which we propagate a trajectory dt_{max} and the minimum propagation step dt_{step}. We assume that the duration of a planning cycle $T = t_{i+1} - t_i)$ is a multiple of dt_{step}

5.4 Integration with IST

Given the overall STSR framework for replanning, it is now possible to utilize the IST planner for tasks that require recomputing trajectories on the fly. There

Algorithm 9 SHORT-TERM SAFETY($T(= t_{i+1} - t_i), dt_{max}, dt_{step}$)

$Timer \leftarrow 0$
Set the root of $\mathcal{T}$ to the initial state $x_0(t_i)$

while $Timer < T$ **do**
{
  Select a reachable state $x_r(t) \in \mathcal{T}$
  Select a valid control u_v for the state $x_r(t)$

  $dt = dt_{step}$
  $keep\_propagating = true$
  while $keep\_propagating == true$ **do**
  {
    if $dt < dt_{max}$ and $\pi_r(x_r, (u_v, dt))$ is feasible **then**
      if $t + dt == t_{i+1}$ and $x^{\pi_r}(t_{i+1})$ is not safe **then**
        $keep\_propagating = false$
    else
      $keep\_propagating = false$
    $dt = dt + dt_{step}$
  }

  if $\pi_r(x_r, (u_v, dt))$ does not intersect time t_{i+1} or
  state $x^{\pi_r}(t_{i+1})$ is safe **then**
    Add all the states along $\pi_r(x_r, (u_v, dt))$ in $\mathcal{T}$
  else
    Add all but the last state along $\pi_r(x_r, (u_v, dt))$ in $\mathcal{T}$

  Update $Timer$ with time passed since calling the function
}

are three important points relating to the use of a planning algorithm within a replanning framework: (i) what is the planning horizon of the planner, (ii) is any computation from previous planning steps reused in consecutive cycles and (ii) what is the effect of an informed algorithm and of a heuristic (as `IST` is) in replanning.

Planning Horizon

Each time we are calling the planner, we are actually interested in a trajectory

of duration only equal to the next planning cycle, e.g., $(t_i : t_{i+1})$. The question that arises is what planning horizon should the planner employ. For example, it would be sufficient if `IST`was searching only for safe trajectories of duration $(t_i : t_{i+1})$. In our integration of `IST` with `STSR`, however, we have allowed for an unbounded planning horizon. The planner keeps searching for trajectories into the future that will not be needed during the next planning step. This approach is selected for the following two reasons:

(i) The decision that is actually being made in each replanning cycle is the following: which state is the system going to be in at the end of the next planning cycle. Due to `STSR`, all of the candidates states are safe, so there is no concern about safety. By allowing to expand trajectories past these states, it is possible to make a more informed decision about the utility of those states in terms of how they assist in solving the task. For example, if the `IST` algorithm is able to find a trajectory that leads to the goal, then the decision is trivial. If we do not allow the planner to look into the future we cannot have this information.

(ii) Although the resulting future trajectories stored on the tree $\mathcal{T}$ are not immediately necessary, they can be potentially used later.

The second point raises the issue of how we can reuse computations from previous planning cycles through an operation we refer to as tree retainment.

Tree Retainment

Since we have a a longer planning horizon than planning cycle, a large part of the tree constructed during the previous step may still be valid in the beginning of the planning procedure and it can be used to accelerate the search for a new path given new sensor data (as in the Dynamic `RRT` approach [FKS06]).

Note, however, that kinodynamic constraints add an additional limitation, since it is not possible to go backwards along the tree. Only the sub-tree of

the initial robot state x_{i+1} for the new planning cycle is valid for planning and everything above it is unreachable and must be discarded. Trimming other parts of the tree that are invalid due to unexpected collisions can similarly be executed. Moreover, if a path to the desired target exists in the remaining tree, every path that does not lead to the target can also be pruned. In this way, when a path has been found, the technique focuses on obtaining plans of increasing quality.

Another issue with tree retainment, however, has to do with safety. Because we are following the `STSR` framework, the retained part of the tree has not actually been checked for safety. Consequently, when we extract the part of the previous tree that can be used as a starting point for the new search we have to make some additional calls to the safety checking oracle. We will have to check all those states along retained trajectories that occur at the end of the next cycle, as we do for newly constructed trajectories.

This implies that there is a trade-off between the computational benefits that we receive from `STSR` and the procedure of tree retainment. `STSR` reduces the number of states that we have to check for safety when we are producing new trajectories along the tree data structure $\mathcal{T}$ but requires that we have to make additional safety checks when we retain part of the previous tree. If we were following the approaches of Fraichard et al. [PF05] and Frazolli et al. [FDF02] there would be no need for additional safety checking during tree retainment. However, overall, `STSR` is able to reduce the number of states that are checked for safety. This is due to the fact, then when we retain $\mathcal{T}$, a big part of the data structure is actually pruned, because it corresponds to unreachable states. In `STSR` these parts of the tree are never checked for safety.

Resetting Mechanisms of `IST`

We need to specify certain details about the mechanics of the `IST` algorithm in the context of replanning. For example, each time that the algorithm is being reset what happens with all the data structures and algorithmic tools that are being employed: the adaptive subdivision and the edge penalization scheme.

In our integration of IST with STSRwe have selected to drop the previous subdivision data structure and create a new one from scratch its time the algorithm is being reset. We also set the edge penalties of edges that are children of the root state to 1, and we linearly increase the penalties for children edges. Since we are using a greedy algorithm that tends to promote the selection of edges further away from the root, this resetting procedure balances the bias and gives an opportunity to edges closer to goal to be selected as well.

Using an Informed Approach

The final point has to do with the effects of using an informed algorithm as the planning module within a replanning framework. The advantage arises when the algorithm has not managed during the previous planning cycle to find a trajectory that reaches the goal but has only returned a partial solution, which is the most often case. An uninformed planner would continue searching towards all possible direction given the retained tree $\mathcal{T}$. An informed planner, however, will promote the quick expansion of $\mathcal{T}$ from the point that is closer to the goal according to the heuristic, even more than in single shot planning. This is a result of the reset and pruning operation that take place in replanning. By cutting a big part of the tree and committing to the previously selected trajectory, the planner becomes even greedier. In environments where the heuristic correctly biases the expansion of the tree, this gives additional computational advantages to informed planners.

Moreover, this results in finding quicker trajectories that do reach the goal. Then these trajectories are the only part of the tree that is being retained and the planner can spend more time in smoothing the resulting trajectory than actually attempting to find one. In this way, the informed approach is also able to improve path quality.

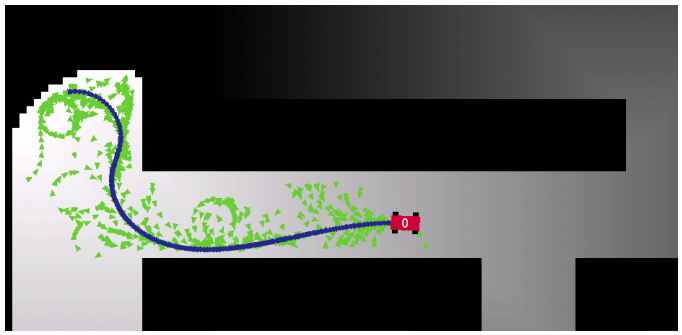

Figure 5.4 : Tree expansion during a single planning cycle that managed to reach the goal. The planner is similar to ISTbut it is biased by a discrete wavefront function that is shown in the background. The light colored triangles correspond to vehicle configurations along the tree data structure. The darker trajectory is the selected path.

5.5 Application to Workspace Exploration

This section describes how the integration of STSR with IST can be applied to solve an interesting replanning task, that of safely mapping an unknown workspace with a system that has significant drift.

Workspace representation and collisions

An important difference between single shot planning and replanning is that in the second we do not have a static, perfect model of the world available, instead we depend upon sensor data and maps of the environment. Most often, maps come in the form of an occupancy grid. In our implementation, we employed an occupancy grid that has 3 values: explored free space, obstacle and unexplored space. Given a state, the robot is positioned on the map and if the chassis intersects an obstacle or an unexplored space cell, it is considered to be collision. Figure 5.1 shows an example of mapping in a two-dimensional simulator for a car-like vehicle.

Selection of target F for the planner

In the case of workspace exploration, the goal region X_G does not remain

the same throughout the execution of the task, because the map of the environment changes. We follow a frontier-based approach to select a goal region for the motion planner at each cycle. A frontier is the boundary between the free explored space and the unexplored one. Neighboring frontier cells are grouped by applying a flooding operation on the grid and the search is biased towards a particular frontier during each planning period. There are many alternative heuristic approaches for selecting target frontiers [BMSS05]. In our implementation, we select frontiers that are: (a) close to the initial robot state given the $\mathcal{C}$ distance metric and (b) small in size, to avoid returning to small unexplored regions after covering large distances. Figure 5.4 shows the A* distance on the grid map from a frontier and a tree expanded towards the frontier.

Trajectory Selection

After we have constructed *tree* with `IST`, then we must select which trajectory to follow. The objective in trajectory selection is to maximize visibility of the unexplored space and minimize the length of the trajectory. Candidate trajectories are all those that initiate from the root to a node of the constructed tree. A weight $w(\pi)$ for every candidate trajectory π is defined as:

$$w(tr(\pi) = e^{-(d(\pi)+\lambda \cdot cost(\pi))} \tag{5.2}$$

where $d(tr)$ describes how close the trajectory is to the selected frontier and $cost(tr)$ expresses the trajectory length. Parameter λ expresses the importance of the path length over the distance to the frontier. There are two different ways to compute the distance parameter $d(tr)$ depending on whether the algorithm has managed to produce states that can sense the frontier or not. In the first case, for a trajectory π and a point f along the goal frontier we define the distance as:

$$d(\pi, f) = \begin{cases} d(x, c) & \text{if } \exists\ x \in \pi \text{ s.t. } f \text{ is visible from } x \\ d_{max} & \text{otherwise} \end{cases}$$

where d_{max} is the sensing radius of the robot. Then $d(tr) = \sum_{\forall f} d(\pi, f)$. In this way, trajectories that see many points along the frontier and which are closer to them have a smaller distance parameter. If there is no state along the tree able to sense the frontier, we define $d(\pi)$ as the minimum distance between the end state of the trajectory and a frontier cell according to a $\mathcal{C}$ metric. Both $l(\pi)$ and $d(\pi)$ can be computed recursively during the tree construction. The safe trajectory of maximum weight that has duration at least T is finally returned. If no such trajectory exists, a collision-free contingency plan is guaranteed to exist by this chapter's theorem.

Discussion

The chpater proposes an efficient framework for replanning under kinodynamic constraints. It builds upon previous work that identifies the issues of `ICS` but achieves to minimize the computational cost of providing safety guarantees. In particular the characteristics of the approach are the following:

(a) `STSR`follow an incremental approach similar to methods that repair `RRTs` [BV06, FKS06], reuses computations from previous planning cycles but deals explicitly with kinodynamic constraints and safety issues.

(b) The planner uses the `ICS` formalization [FA04] and provides safety guarantees similar to τ-safety [FDF02]. It reduces, however, the cost of achieving safety by controlling the duration of the planning cycle and employing the short-term safety principle. This principle results in a reduced number of states that must be checked for safety, leading to considerable speedups.

(c) The `STSR` framework is compatible with the `IST`algorithm and takes advantage of its informed nature for replanning.

(d) It has been tested on an application involving the mapping of unknown workspaces with a car-like vehicle that has significant drift. The experimental results are available in Chapter 7.

Chapter 6

Distributed Safe Replanning

The progress in wireless networking allows to consider groups of vehicles that operate in the same environment and use communication to coordinate their motion. Moreover, it gives rise to the idea of networks of vehicles that jointly solve a task while retaining connectivity. Using such teams of multiple, coordinating vehicles offers redundancy and robustness in the execution of many tasks (e.g. space exploration, autonomous demining). Nevertheless, the control of such systems involves multiple research challenges.

This chapter focuses on motion coordination challenges. Given procedures for updating a vehicle's map, state and goal, the objective is to design feasible, collision-free trajectories for multiple vehicles operating in the same, partially-known environment (see Fig. 6.1). In particular, we deal with the safety concerns that arise due to the kinodynamic motion constraints (e.g., bounded velocity and acceleration, smooth steering) that real vehicles exhibit. We study at a theoretical level how inter-vehicle communication [YLVZ04, Hom] can be utilized in this context to achieve safe motion coordination.

We are interested in a solution with the following characteristics:

(i) A general and abstract algorithm that is not limited to specific system dynamics or to specific types of workspaces and obstacles.

(ii) A scalable, distributed solution that respects the physical limitations in sensing and communication and avoids centralized computation.

(iii) A real-time algorithm, since vehicles do not typically have global knowledge of their workspace. This means that sensing, planning and execution are

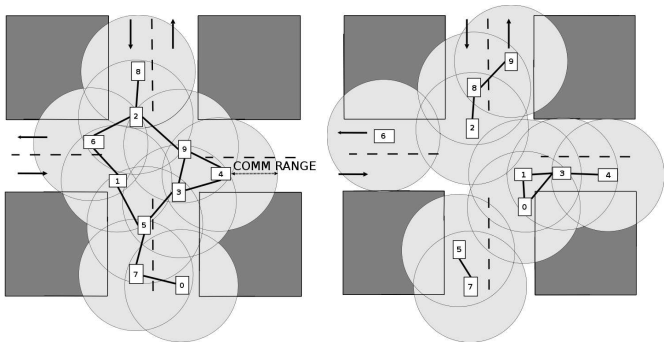

Figure 6.1 : Vehicles form a communication network while they move. On the left, there is one connected component while on the right vehicles have moved and multiple components have been created. Planning for such dynamic networks with centralized approaches has been studied for first-order systems [CRL03, CBR02]. This thesis extends these ideas by considering second order dynamics (we guarantee avoidance of Inevitable Collision States) and describing a decentralized solution using only local information.

interleaved and there is limited amount of time to compute a partial plan towards the goal.

(iv) A safe solution for systems with second-order constraints. The algorithm must provide guarantees for collision-avoidance and the retainment of a communication network if desired by the team.

The first point implies that the sampling-based kinodynamic framework is appropriate for designing such a distributed coordination technique. The last two points suggests that we can also make use of the work in the previous chapter that showed how to achieve safe replanning for systems with second-order constraints, such as systems with significant drift. In some sense, this chapter will extend the `STSR` framework into the case of multiple coordinating agents.

The proposed method in this chapter first identifies the information that must be exchanged between the vehicles so as to plan safe trajectories. These information requirements dictate our approach. Each vehicle uses an `IST`-like approach to generate feasible trajectories that allow the existence of safe alternatives to other vehicles. For coordinating the selection of compatible trajectories between

vehicles, we initially present a priority-based scheme. This allows us to provide a proof that the vehicles always has safe trajectories to follow. The priority scheme is then replaced by an asynchronous, message-passing protocol [KV06, PK04], which still guarantees safety. Among the safe solutions and given the available time, the asynchronous protocol optimizes a joint payoff function.

The proposed method has been implemented on a multi-processor simulator. Each processor models a vehicle and communicates asynchronously with other processors. The experimental results confirm the theoretical guarantees of collision avoidance and network retainment for second-order vehicles jointly exploring an unknown workspace. They are presented in the following chapter.. The distributed protocol has computational and scaling advantages when compared against the prioritized scheme [BTK07a, BTK07b].

6.1 Problem Definition

Consider multiple vehicles $V = \{V_1, \ldots, V_v\}$ operating in a world with obstacles. Each vehicle is able to sense a local region around it and can communicate with other vehicles within a limited range. Each vehicle V_i is a dynamic system whose motion is governed by differential equations of the form presented in Eq. 3.1 in Chapter 3. The exact dynamics of the systems we experimented with can be found in Chapter 7.

Given the communication limitations and states $\{x_1(t), \ldots, x_v(t)\}$, the vehicles form dynamic communication links represented by a graph $G(t) = \{V(t), E(t)\}$, where $e_{ij} \in E(t)$ as long as V_i, V_j are within range. The neighbors of V_i in the graph $G(t)$ are denoted as $N_i(t)$. Vehicle V_i can only communicate with vehicles in the set $N_i(t)$.

The vehicles execute tasks which require motion. While moving, the vehicles must avoid collisions both with obstacles and with other vehicles. The vehicles must update their world model and state estimate given new sensory information. In parallel, they must compute in real-time trajectories towards their goals. To

achieve this objective, a vehicle's function is broken down into a sequence of consecutive operational cycles, as in Section 5.1.

Here we focus on how to utilize communication so as to provide safety guarantees when multiple vehicles operate in close proximity. Each vehicle can only communicate with neighboring vehicles given the communication constraints and exchange information. We will specify what kind of information has to be exchanged to guarantee collision avoidance. The following assumptions are being made:

- Communication is reliable, offers sufficient bandwidth and is not affected by line of sight constraints. The vehicles synchronize their operation.
- As in previous chapters, we do not deal with issues related to uncertainty. We assume that motion commands selected and communicated by a vehicle are executed fairly accurately and there are no sensing errors.

We will have to provide additional notation on top of the definitions of a plan and a trajectory from Section 3.1. When a vehicle executes a plan $p(dt)$ from state $x(t)$ and consecutively executes plan $p'(dt')$, then the resulting **trajectory concatenation** will be denoted as:

$$\pi'(\ \pi(x(t), p(dt)),\ p'(dt')\ ). \tag{6.1}$$

If two vehicles V_i, V_j at time t are not in collision with each other or with obstacles, then their corresponding states $x_i(t), x_j(t)$ are **compatible states**: $x_i(t) \asymp x_j(t)$.

Two trajectories $\pi_i(x_i(t_i), p_i(dt_i))$ and $\pi_j(x_j(t_j), p_j(dt_j))$ are **compatible trajectories** ($\pi_i \asymp \pi_j$) if the two trajectories do not cause collisions with workspace obstacles and:

$$\forall\ t' \in [\max\{t_i, t_j\} : \min\{t_i + dt_i, t_j + dt_j\}] :$$
$$x^{\pi_i}(t') \asymp x^{\pi_j}(t').$$

Compatible trajectories between vehicles may still lead to an ICS from which a collision cannot be avoided in the future due to second-order constraints [FA04]. The definition of ICS in this multi-agent setup has to be extended. A state $x_i(t)$ is an **Multi-agent Inevitable Collision State (MICS)** given the states $\{x_1(t), \ldots, x_v(t)\}$ if $\forall\ \pi_i(x_i(t), p_i(\infty))$:

$$\exists\ (dt \wedge j \neq i) \text{ so that } \forall\ \pi_j(x_j(t), p_j(\infty))$$
$$\text{states } x^{\pi_i}(dt) \text{ and } x^{\pi_j}(dt) \text{ are not compatible.}$$

Safe Motion Coordination:
Given the map of the world and a state estimate $x_i(t+dt)$, the motion planning module of each vehicle V_i must compute before time $(t+dt)$ a plan $p_i(dt')$ so that given the trajectories of all other vehicles $\pi_j(x_j(t+dt), p_j(dt'))$ $(\forall j \neq i)$:

- $\pi_i(x_i(t+dt), p_i(dt')) \asymp \pi_j(x_j(t+dt), p_j(dt'))$
- *State $x_i^{\pi_i}(dt')$ is not* MICS.

A secondary objective for the planner is to refine the quality of the selected trajectory given a measure of path quality and a goal $X_G^i(t)$.

6.2 Multi-Agent ICS Avoidance

The proposed approach has two characteristics. Motion coordination is achieved in a decoupled manner. This feature distinguishes our approach when compared against related work on planning for communicating vehicles [CRL03], where the vehicles forming a temporary dynamic network solve a centralized problem. Moreover, the motion planning and coordination operation of each vehicle are split into two separate steps as shown in Fig. 6.2:

1. **Generate Candidate Plans:** During the first step, the algorithm searches the state space of each vehicle V_i so as to generate a set of candidate plans $\mathcal{P}_i$ by employing a single-vehicle planner.

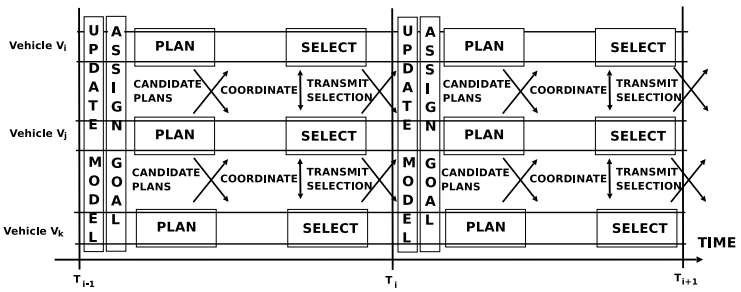

Figure 6.2 : The operation that a single vehicle executes in two consecutive planning cycles.

2. **Select Compatible Plans:** During the second step, neighboring vehicles communicate by exchanging sets $\mathcal{P}_i$ and evaluating their performance in terms of collision avoidance and task execution.

For the plan generation step, sampling-based, kinodynamic planners, such as `IST`, are particularly appropriate to search the state space and produce multiple candidate valid plans $\mathcal{P}_i$ that are at least collision-free with the workspace obstacles. By employing a technique such as `STSR`, it is also possible to use them to execute a replanning operation and avoid `ICS`with static obstacles in the environment.

Two plans of different vehicles are acceptable solutions if the corresponding trajectories are compatible and they must be selected appropriately through coordination in the second step. The current and the following section will describe how such coordination can be achieved by employing a priority-based scheme and how safety can be guaranteed assuming the preassigned priorities and no communication limitations. These assumptions will be later waived. In particular, in Section **??**, the exchanged plans $\mathcal{P}_i$ are viewed as actions in a discrete action space. Then the problem of distributedly selecting compatible trajectories is reduced to a distributed constraint optimization problem, for which message-passing algorithms without priorities exist [KV06, PK04].

For the remaining of this section we will assume vehicles that have unlimited communication range and preassigned priorities.

We will start from a simple extension of the single-system short-term safe replanning framework to a coordinated approach. As Fig. 6.1 shows, communication links between vehicles define a graph, where the vehicles are nodes and two vehicles share an edge if the two vehicles can exchange messages. In the case of unlimited communication range this graph is complete. Suppose every vehicle has a unique global priority. We define the set N^h to represent the neighbors of vehicle V on the communication graph with higher priorities than V and the set N^l to be the set with lower priorities. Then the simple prioritized scheme executed on each vehicle V during a single planning cycle $(t_n : t_{n+1})$ employs the following steps:

1. Compute a set of candidate plans $\mathcal{P}$ of duration $(t_n : t_{n+1})$ with a single-vehicle planner.
2. Receive the selected plans $\mathcal{P}^h$ from neighbors in N^h.
3. Select plan $p(t_n : t_{n+1}) \in \mathcal{P}$ that does not collide with plans in $\mathcal{P}^h$ and best serves the goal of vehicle V.
4. Transmit the plan p to all neighbors N^l.

The simple extension, however, fails to produce safe trajectories for multiple reasons:

- If a cycle is completed before all higher priority plans are received, no plan p can be safely selected.
- Even if $p \in \mathcal{P}$ and $\mathcal{P}^h$ are available on time, it may happen that no plan p is collision-free with all plans in $\mathcal{P}^h$ due to the decentralized nature of the approach.
- Suppose p is collision-free with set $\mathcal{P}^h$. It may still lead to `MICS` given $\mathcal{P}^h$ due to the dynamics.

The definition of a safe state from Eq. 5.1 is inadequate in the multi-vehicle case, where the safety of a vehicle's state depends on the states and the choices of the other vehicles. We extend the definition of safety as follows:

Safe State - Multi-vehicle case: Consider vehicles $V_1, \ldots, V_v$ that have states $x_1(t), \ldots, x_v(t)$ and all vehicles $V_j, j \neq i$ execute plans $p_j(dt)$. Then state $x_i(t)$ is safe iff $\exists\ \gamma_i(\infty)$ so that:

$$\begin{aligned} &\pi_i(x_i(t), \gamma_i(\infty)) \text{ is collision free} \quad \wedge \quad \forall\ j \neq i : \\ &\pi_i(x_i(t), \gamma_i(\infty)) \asymp \pi'_j(\ \pi_j(x_j(t), p_j(dt)),\ \gamma_j(\infty)\ ) \end{aligned} \tag{6.2}$$

Note that the trajectory $\pi_i(x_i(t), \gamma_i(\infty))$ must be compatible with the concatenation of other vehicles' plans and contingencies. Given this new definition of a safe state, we set an objective for the coordination algorithm we described earlier. It must satisfy the following.

Invariant: For each replanning cycle $(t_n : t_{n+1})$ every vehicle V_i selects a plan $p_i(t_n : t_{n+1})$ which when executed at state $x_i(t_n)$:

a. The resulting trajectory $\pi_i(x_i(t_n), p_i(t_n : t_{n+1}))$ is collision-free.

b. During the current cycle $(t_n : t_{n+1})$, it is compatible with all other vehicles, $\forall j \neq i$:

$$\pi_i(x_i(t_n), p_i(t_n : t_{n+1})) \asymp \pi_j(x_j(t_n), p_j(t_n : t_{n+1})).$$

c. It leads to state $x^{\pi_i}(t_{n+1})$ that is safe according to Eq. 6.2 for every choice of plans $p_j(t_{n+1} : t_{n+2})$ that the other vehicles may make during the next planning cycle.

If the Invariant holds then the algorithm will produce safe trajectories. Points a. and b. imply that there is no collision during the current cycle $(t_n : t_{n+1})$, either with static geometry or between vehicles. Point c. implies that all vehicles at the next cycle $(t_{n+1} : t_{n+2})$ have contingency plans which can be followed regardless of the other vehicles' choices. Consequently, the prioritized algorithm

in the beginning of this section can be altered so that step 3 is:

3) Select plan $p(t_n : t_{n+1}) \in \mathcal{P}$ that satisfies the Invariant given the set P^h. If no such plan exists or time is running out, execute contingency $\gamma(t_n : t_{n+1})$, which is precomputed from the previous planning cycle and collision-free due to the Invariant.

Consequently, now we need to answer the question of: *how to produce and select plans $p(t_n : t_{n+1})$ that satisfy the Invariant.* We will show that any selected plan at step 3 of the algorithm must satisfy:

Requirement 1: As in the single-vehicle case, the concatenation of plan $p_i(dt)$ with a contingency plan $\gamma_i(\infty)$ must be collision-free:

$$\pi'(\ \pi(x(t_n), p(t_n : t_{n+1})),\ \gamma(\infty)\ ) \text{ is collision-free.} \tag{6.3}$$

Requirement 2: The concatenation of plan $p_i(dt)$ with a contingency plan $\gamma_i(\infty)$ must be compatible with the contingency plans $\gamma_j(\infty)$ of other vehicles:

$$\begin{aligned} \forall j \neq i: \quad & \pi_i'(\ \pi_i(x_i(t_n), p_i(dt)), \gamma_i(\infty)) \\ & \asymp\ \pi_j(x_j(t_n), \gamma_j(\infty)) \end{aligned} \tag{6.4}$$

Requirement 3: The concatenation of plan $p_i(dt)$ with a contingency plan $\gamma_i(\infty)$ must be compatible with the concatenations of plans $p_j(dt)$ of other vehicles with their contingency plans $\gamma_j(\infty)$:

$$\begin{aligned} \forall j \neq i: \quad & \pi_i'(\ \pi_i(x_i(t_n), p_i(dt)), \gamma_i(\infty)) \\ & \asymp \pi_j'(\ \pi_j(x_j(t_n), p_j(dt)), \gamma_j(\infty)) \end{aligned} \tag{6.5}$$

Theorem: Assume the Invariant is satisfied during planning cycle $(t_{n-1} : t_n)$ for all vehicles. Then if each vehicle V_i selects a plan $p_i(t_n : t_{n+1})$ that satisfies

Eq. 6.3, 6.4 and 6.5 or selects an available contingency plan, then the Invariant will also hold during the next planning cycle $(t_n : t_{n+1})$.

Proof: We will have to show that the three points of the Invariant are satisfied during the next planning cycle $(t_n : t_{n+1})$. There are two cases. Either the algorithm manages to produce and select a plan $p_i(t_n : t_{n+1})$ that satisfies Eq. 6.3, 6.4 and 6.5 or selects a contingency plan. We will treat these two cases separately:

1) Assume such plan $p_i(t_n : t_{n+1})$ has been found. Because the plan satisfies Eq. 6.3 and Eq. 6.5, points a. and b. of the Invariant are satisfied, respectively. Point c. is more complicated. The state $x(t_{n+1})$ that the vehicle will reach after executing $p_i(t_n : t_{n+1})$ must have the property that it is safe according to Eq. 6.2. The application of the contingency plan γ_i at state $x(t_{n+1})$ will result in a collision-free path according to Eq. 6.3, so one of the two specifications of Eq. 6.2 is satisfied. State $x(t_{n+1})$ has to be safe, however, for any choice of plans $p_j(t_{n+1} : t_{n+2})$ that the other vehicles will make during the next planning cycle. There are again two possible cases for the nature of plans another vehicle V_j can follow during cycle $(t_{n+1} : t_{n+2})$:

a. Assume vehicle V_j computes a plan $p_j(t_{n+1} : t_{n+2})$ that satisfies the requirements. Then due to Eq. 6.4, this plan is compatible with the contingency of V_i during that cycle:

$$\pi_i(x^{\pi_i}(t_{n+1}), \gamma_i(\infty)) \asymp$$
$$\pi'_j(\ \pi_j(x^{\pi_j}(t_{n+1}), p_j(t_{n+1} : t_{n+2})),\ \gamma_j(\infty)\ )$$

b. Assume vehicle V_j resorts to a contingency during cycle $(t_{n+1} : t_{n+2})$. Due to Eq. 6.5, however, the contingency of V_j is by construction compatible with the contingency of V_i:

$$\pi_i(x^{\pi_i}(t_{n+1}), \gamma_i(\infty)) \asymp \pi_j(x^{\pi_j}(t_{n+1}), \gamma_j(\infty))$$

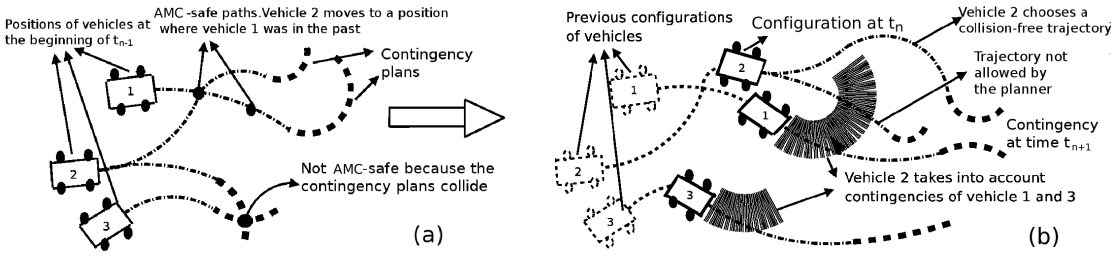

Figure 6.3 : (a) The lower plan for V_2 is not safe since the contingency attached to it collides with the contingency extending from the plan of V_3. The top plan of V_2 is safe. (b) The planner of V_2 will not produce the lower trajectory because it collides with the current contingency of V_1. The top plan is again safe.

In any case, Eq. 6.2 is satisfied for state $x^{\pi_i}(t_{n+1})$, which means that the third point of the Invariant is also satisfied for the next planning cycle.

2) Assume that vehicle V_i has to resort to a contingency. The inductive hypothesis is that the Invariant holds during the current cycle, so the state $x(t_n)$ is safe according to Eq. 6.2 for every choice of plans of other vehicles. From Eq. 6.2 the points a. and b. of the Invariant trivially hold for the trajectory that follows the contingency plan. In order to show that the state $x(t_{n+1})$ reached after the application of the contingency plan $\gamma_i(t_n : t_{n+1})$ is safe according to Eq. 6.2 we can follow exactly the same reasoning as above. From Eq. 6.3 the trajectory $\pi_i(x(t_{n+1}), \gamma_i(\infty))$ will be collision-free and will also be compatible given any choice the other vehicles will make due to Eq. 6.4 and 6.5. □

6.3 Simple Prioritized Protocol

We describe here how we can algorithmically satisfy the specified requirements within a prioritized scheme. Fig. 6.3 provides an illustration of how a vehicle (V_2) uses information from its neighbors (V_1, V_3) to respect Req. 2 and 3.

To satisfy the second requirement, each vehicle V_i must be aware of the contingency plans of other vehicles V_j at state $x(t_n)$ during planning cycle $(t_{n-1} : t_n)$. These contingency plans have already been computed by each V_j during the previous step. This information can be communicated at the beginning of each cycle.

Algorithm 10 PRIORITY-BASED MOTION COORDINATION for V_i

Identify set of neighbors $N = N^h \cup N^l$

`(Exchange contingencies)`
for all $j \in N$ **do**
{
  Send contingency $\gamma_i(\infty)$ to V_j
  Receive contingency $\gamma_j(\infty)$ from V_j
}

`(Planning: satisfies requirements 1,2)`
$HN \leftarrow N^h$ (high priority neighbor set)
Select $PlanningBudget$ according to priority
$Tree \leftarrow$ Retain valid subset of $Tree$ from previous cycle
while $(time < PlanningBudget) \wedge HN \neq \emptyset$ **do**
  MAIN_PLANNING_LOOP_PRIORITY_BASED()

`(Path Selection: satisfies req. 3)`
$p_i^* \leftarrow \gamma_i(\infty)$ (safe from previous round)
$P' \leftarrow$ Extract all plans $p_i'(t_n : t_{n+1})$ from $Tree$
for all $p_i' \in P'$ and while $(time < PlanningCycle)$ **do**
{
  for all $j \in N^h$ **do**
  {
    if (Eq. 6.5 does not hold for $p_i', p_j^*(t_n : t_{n+1}), \gamma_j^*(\infty)$) **then**
      Reject p_i'
  }
  if p_i' is not rejected and p_i' better than p_i^* **then**
    $p_i^* \leftarrow p_i'(t_n : t_{n+1})$
}

`(Transmit selected plan)`
for all $j \in N^l$ **do**
{
  Send selected plan p_i^* to V_j
  Send contingency $\gamma_i^*(\infty)$ at $x_i(t_{n+1})$ to V_j
}

After exchanging contingency plans, the sampling-based, kinodynamic planner is invoked. It generates a tree data structure of feasible trajectories in the state-space that are collision-free and avoids `ICS` with obstacles in the beginning of the

Algorithm 11 MAIN_PLANNING_LOOP_PRIORITY_BASED

```
(Sampling-Based Kinodynamic Planning)
Select an existing trajectory sample s from Tree
Select plan p(dt) and state x(t) on s
Propagate trajectory π(x(t), p(dt))

(Req. 1: Avoid ICS with obstacles)
if (π(x(t), p(dt)) is not collision-free) then
  Reject π
else
{
  if (t < t_{n+1}) ∧ (t + dt > t_{n+1}) then
    (path intersects next cycle t_{n+1})
    if (π(π(x(t), p(dt), γ(∞)) not collision-free) then
      (Leads to ICS with obstacles)
      Reject π
}

(Req. 2: Compatibility with γ_j(∞))
for all j ∈ N and while π is not rejected do
  if ( π(x(t), p(dt)) ≯ π_j(x_j(t_n), γ_j(∞)) ) then
  {
    (Does not respect Eq. 6.4)
    Reject π
  }

(Receive high priority plans)
if (message arrived from j ∈ HN) then
{
  Receive selected plan p*_j(t_n : t_{n+1})
  Receive contingency γ*_j(∞) at x_j(t_{n+1})
  Remove j from HN
}
```

consecutive planning cycle (Eq. 6.3). The planner considers also in collision all the trajectories that intersect the contingencies of other vehicles to satisfy Eq. 6.4.

The third requirement specifies that when a vehicle makes a decision it must inform the other vehicles so that pairs of plans satisfy Eq. 6.5. This exchange of information can follow the vehicles' priorities. Vehicle V_i with the highest

priority computes a solution plan $p_i(t_n : t_{n+1})$ from the motion planner and the accompanying contingency plan that could be executed at state $x^{\pi_i}(t_{n+1})$. V_i transmits its solution to lower priorities vehicles, which must now come up with a plan that respects Eq. 6.5 given V_i's choice. Every vehicle waits to receive the choices of all vehicles with higher priorities before selecting its own plan. If a plan that respects Eq. 6.5 is available from the tree data structure computed by the motion planner, then it is selected and transmitted to the lower priority vehicles. If no such plan is found, the available contingency plan is selected and transmitted. If time is running out (variable *PlanningCycle* in Algorithm 1) and not all higher priority vehicles have send their plans, then a contingency plan is again selected.

Note that the prioritized scheme imposes a total ordering over all the vehicles. When the communication graph is complete this results in the lowest priority vehicle having to wait for all the previous vehicles to select plans. The high priority vehicle has to transmit its selection early enough (variable *PlanningBudget* in Algorithm 1) so that the sequence of selected plans reaches all vehicles within the planning cycle along a chain of priorities. Even if the *PlanningBudget* is not sufficiently long so that all vehicles communicate, the vehicles still do not collide in our setup. The vehicles will end up selecting contingency plans, which correspond to stopping maneuvers and they will stop safely. This undesired effect is less pronounced when vehicles have limited communication as Fig. 6.5 shows, because vehicles that do not communicate do not effect one another. We have, however, addressed this advantage by proposing a fully distributed approach, described in Section 6.4, that guarantees the satisfaction of the three requirements without priorities.

6.4 Message-Passing Distributed Protocol

As mentioned before, the priority-based methodology exhibits a major drawback. In the worst case and depending on the connectivity of the network, the vehicle

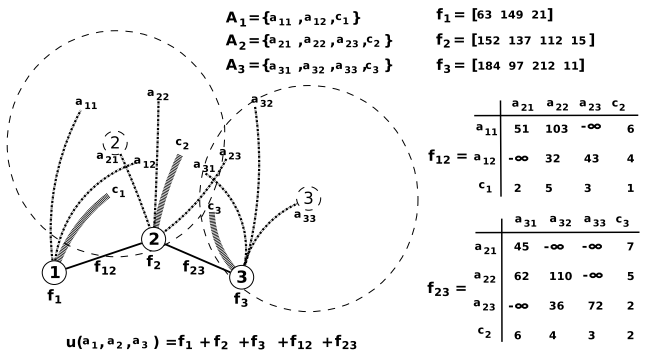

Figure 6.4 : A simple coordination graph, the action sets A_1, A_2, A_3, the atomic and pairwise payoffs $f_1, f_2, f_3, f_{12}, f_{23}$ and the global utility function u

with the lowest priority has to wait for every other vehicle to select a plan before taking its own decision. This implies that vehicles with low priorities often do not receive the choices of their higher priority neighbors within the duration of the planning cycle and they have to resort to contingency plans. The proposed safety requirements, however, do not depend on the prioritized scheme. They allow the implementation of message-passing protocols for distributed constraint optimization. The coordination problem can be modeled with coordination graphs [GKP02] and can be solved with distributed message passing algorithms based on *belief propagation* [Pea88], such as the *max-plus* algorithm [KV06]. In this section, we describe an approach that employs these abstractions.

In the formalization of coordination graphs, we assume we have n agents, and each agent i has to select an action a_i out of a finite action set A_i. Generally the goal is to find the optimal action vector $\overline{a}^* = (a_1^*, \ldots, a_n^*)$ that maximizes a global utility function $u(\overline{a})$. The utility has a structure captured by a coordination graph $CG = (V, E)$. On every node of CG we define a function $f_i(a_i)$ called *atomic payoff*. An atomic payoff describes how well each action serves the goal of the agent corresponding to that node. On the edges e_{ij} of CG we define *pairwise payoff* functions $f_{ij}(a_i, a_j)$ that indicate how good for the team are pairs of actions of interacting agents (see Fig. 6.4 for an example). The global utility is assumed to depend only on the unary and pairwise payoff functions as follows:

$$u(\overline{a}) = \sum_i f_i(a_i) + \sum_{e_{ij} \in E(CG)} f_{ij}(a_i, a_j)$$

Max-plus is a distributed message passing algorithm that attempts to compute an optimal action vector using only local computations and communication for every agent. While the algorithm is running, each agent i chooses a neighboring agent j on CG, collects and adds all incoming messages from other agents in its neighborhood, and sends a new message to j that is computed by the following formula:

$$m_{ij}(a_j) = \max_{a_i} \{ f_i(a_i) + f_{ij}(a_i, a_j) + \sum_{k \in N(i), k \neq j} m_{ki} \} \tag{6.6}$$

At any time during the execution, the agents can compute a *marginal* function $g_i(a_i) = f_i(a_i) + \sum_{k \in N(i)} m_{ki}$. Maximizing g_i provides the best possible action a'_i for agent i with respect to messages from other agents. $u(a'_1, \ldots, a'_n)$ is an approximation to the optimal u.

In order to adapt this formulation in our framework, each vehicle can be viewed as a node in the coordination graph CG. Two nodes share an edge in the graph if the corresponding vehicles can communicate or if they can potentially collide. The discrete set of actions of the max-plus algorithm corresponds to the set of candidate plans $\mathcal{P}_i$. The atomic payoffs $f_i(p_i)$, where $p_i \in \mathcal{P}_i$ can be computed by evaluating how close each trajectory takes vehicle V_i to its goal G_i. In the evaluation of the pairwise payoffs we must also express whether two trajectories are compatible or not:

$$f_i(p_i(dt), p_j(dt)) = -\infty \text{ if}$$
$$\pi_i(x_i(t+dt), p_i(dt)) \not\asymp \pi_j(x_j(t+dt), p_j(dt)) \tag{6.7}$$

where $p_i \in \mathcal{P}_i$ and $p_j \in \mathcal{P}_j$. If the two plans p_i and p_j are compatible then the pairwise payoff can be assigned a positive value that depends on other properties that are required to be optimized. Consequently, it is necessary before the

coordination step for the neighboring vehicles to exchange the sets of candidate plans so as to compute the pairwise payoff functions.

The pairwise payoffs can be computed in a distributed way that balances the burden among vehicles. Each vehicle computes one row for every pairwise payoff matrix that it is involved in, in a cyclic order. Then each vehicle transmits this row to its neighbors. In the computation of f_{ij} if $i \leq j$ then i computes rows $r_1, r_2, \ldots$ and j computes in reverse $r_{max}, r_{max-1}, \ldots$, until the whole array is completed. In this way, a vehicle that has many neighbors is not overwhelmed and its computational overhead is outsourced to neighbors with a smaller number of payoff matrices to compute.

Message-Passing Algorithm

We describe here how the new message-passing approach can satisfy the requirements in a distributed way. The algorithm is provided in pseudo-code in Algorithm 3.

As in the priority-based algorithm, each vehicle V_i must be aware of the contingency plans of neighboring vehicles V_j at state $x(t + dt)$. This information is communicated at the beginning of each cycle between neighbors and the sampling-based, kinodynamic planner is invoked to construct the $Tree$. Next, the set of candidate plans $\mathcal{P}_i$ is constructed from $Tree$. For each plan in this set, the corresponding contingency is attached to it and the unary payoff $f_i(p_i)$ is evaluated. Then, neighboring vehicles exchange their candidate plans and compute pairwise payoffs. Instead of using the simple form of compatibility as in Eq. 6.7, however, we must use Eq. 6.5, which takes into account the concatenation with contingency plans. Given the definition of unary and pairwise payoffs the asynchronous message-passing protocol is initiated and the vehicles start exchanging messages. When the algorithm runs out of time, vehicle transmit to their neighbors their final action selections. Max-plus is incomplete, so if two

Algorithm 12 SAFE AND DISTRIBUTED REPLANNING

Identify set of neighbors N_i

`(Exchange contingencies)`
for all $j \in N_i$ **do**
  Send contingency $\gamma_i(\infty)$ to V_j
  Receive contingency $\gamma_j(\infty)$ from V_j

`(Planning: satisfies requirements 1,2)`
$Tree \leftarrow$ Retain valid subset of $Tree$ from previous cycle
while $(time < PlanningBudget)$ **do**
  MAIN_PLANNING_LOOP_DISTRIBUTED()

`(Evaluate and exchange candidate plans)`
$\mathcal{P}_i \leftarrow$ plans of duration dt from $Tree$
for all $p_i \in \mathcal{P}_i$ **do**
{
  Attach contingency plan $\gamma_i(\infty)$ to $p_i(dt)$
  Evaluate unary payoff $f_i(p_i)$ for $p_i \in \mathcal{P}_i$ given G_i
}
for all $j \in N_i$ **do**
{
  Send set $\mathcal{P}_i$ to V_j
  Receive set $\mathcal{P}_j$ from V_j
  `(Take Req. 3 into account)`
  for all $p_i \in \mathcal{P}_i$ **do**
    for all $p_j \in \mathcal{P}_j$ **do**
      if $(\pi_i'(\pi_i(x_i(t_n), p_i(dt)), \gamma_i(\infty)) \asymp \pi_j'(\ \pi_j(x_j(t_n), p_j(dt)), \gamma_j(\infty))$ **then**
        Compute payoff $f_{ij}(p_i, p_j)$ given the goals G_i, G_j
      else
        $f_{ij}(p_i, p_j) = -\infty$
}

COORDINATION_PROTOCOL()

neighboring vehicles have selected incompatible trajectories then one of the two vehicles switches to the contingency plan. This is a very fast adjustment step, which guarantees that the third requirement is always satisfied. In the experiments section we show that max-plus has to resort to the contingency plan less often than priority-based schemes.

Algorithm 13 MAIN_PLANNING_LOOP_DISTRIBUTED

Select a state $x(t')$ on the existing $Tree$
Select valid plan $p(dt')$
Propagate trajectory $\pi(x(t), p(dt')$
if ($(\pi(\pi(x(t), p(dt'), \gamma(\infty))))$ is not **collision-free**) **then**
  Reject π
else
  for all $j \in N_i$ and while π not rejected **do**
    if ($\pi(x(t), p(dt')) \not\asymp \pi_j(x_j(t_n), \gamma_j(\infty))$) **then**
      Reject π

Algorithm 14 COORDINATION_PROTOCOL()

`(Coordination)`
Enter into asynchronous message-passing to optimize:
$u(\overline{p}) = \sum_i f_i(p_i) + \sum_{e_{ij} \in E(CG)} f_{ij}(p_i, p_j)$
Stop protocol before time $t + dt$
Select plan $\overline{p_i}$ that maximized $u(\overline{p})$
for all $j \in N_i$ **do**
  if ($\overline{p_i}$ incompatible with $\overline{p_j}$) **then**
    Select contingency γ_i as the next action

The secondary goal is to find among the safe solutions one that maximizes the global utility u, within the allocated amount of time. Because the algorithm does not monotonically increase the global utility, it must periodically compute u and keep track of the action vector that produced the maximum value. However, no single agent has all the available information to compute u. In order to achieve an efficient, distributed computation of the utility we use a minimum spanning tree of CG. We use again distributed protocols for computing minimum spanning trees on graphs using local information such as the distance between agents [SB95]. Given the minimum spanning tree structure, an arbitrary vehicle acting as a root of the tree initiates the process:

Down pass The root transmits a signal to compute u. Each node passes the signal down the tree.

Up pass Each node i:

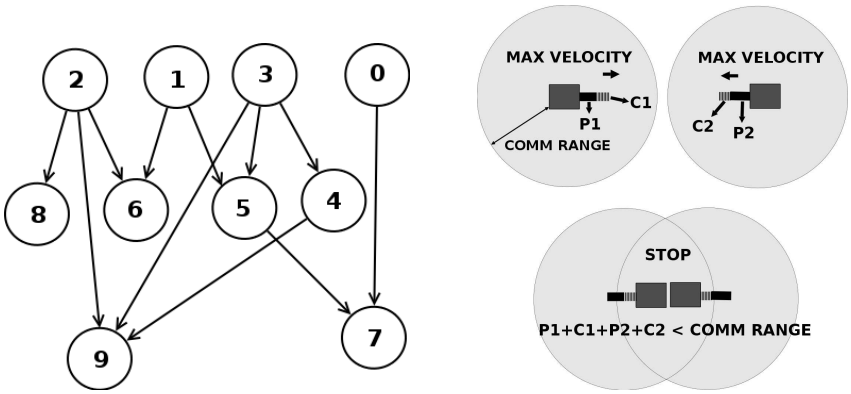

Figure 6.5 : (left) For the dynamic network in Fig. 6.1 the above DAG shows the transmission of selected plans p by high priority vehicles to lower priority vehicles - low number denote high priority. (right) Two vehicles that enter each other's comm range at maximum velocity, cannot collide if after finishing their plans they execute their contingency plans.

1. Collects partial payoff values and actions from children.

2. Maximizes marginal g_i and chooses best action a_i.

3. Adds its contribution to the global payoff u.

4. Sends new partial payoff and actions to its parent.

Down pass The root adds up all partial payoffs so as to compute and maximize u. The optimal value of u^* and actions $\overline{a}^*$ are transmitted down the tree.

This computation is fast since the utility computation messages can be interleaved with normal max-plus messages [KV06].

6.5 Limited Communication

When vehicles have limited communication range, dynamic networks are formed and dissolved as the vehicles move towards their goals (Fig. 6.1). This, however, does not considerably effect the algorithm as long as two vehicles not within range cannot collide until they approach one another and communicate. Given enough space to deaccelerate and come to a complete stop the collision can be avoided.

The Invariant can still be guaranteed by imposing limits on the maximum velocity of the vehicles by taking into account the worst case scenario, shown in Fig. 6.5 (right). Two vehicles are just outside range and they move with maximum velocity towards one another. Then they will keep approaching one another for an entire cycle with maximum velocity. At the end of the cycle, however, they will communicate. If they manage to find compatible plans, they will continue operating normally. Otherwise, they must execute contingencies. The Invariant can still be satisfied as long as the following is true: the distance that a vehicle covers until it comes to a complete stop when it moves at maximum velocity for one planning cycle and then applies a contingency plan must be less than half of the communication range. For some realistic parameters for car-like vehicles (comm. range 100m, breaking deacceleration 10m/sec^2, planning cycle 1sec) the allowable maximum velocity is considerably high (approx. 80Km/h or 50mph).

In the case of limited communication the flow of information is not a chain. The Directed Acyclic Graph (DAG) in Fig. 6.5 (left), shows the flow of information and the partial ordering defined by the priorities of the dynamic vehicular network displayed in Fig. 6.1(left). The DAG structure allows for the planning and selections steps to be executed in parallel on many vehicles even with a prioritized scheme.

6.6 Extension to Vehicular Networks

It is also easy to satisfy the constraint that the vehicles maintain a communication network while moving. Assume the vehicles form a communication graph as in Fig. 6.1(left) and the objective is to move as a vehicular network. To satisfy the network constraint, we need the communication graph to remain connected. For the latter, it is sufficient to retain communication links along a spanning tree of the communication graph. There are efficient algorithms that can compute an approximate spanning tree distributedly given knowledge of the locations of

neighboring vehicles [SB95]. This can be done in the beginning of every planning cycle. The planning algorithm has then to guarantee that the vehicles do not choose trajectories that will break the communication links along the spanning tree.

Assume V_i, V_j share an edge e_{ij} on the spanning tree. We can make sure that e_{ij} will not break if we treat as collision any pair of trajectories that concatenated with the corresponding contingencies bring V_i, V_j out of range. This amounts to just adding an extra check for requirements 2 and 3 for the pairs of vehicles that share edges of the spanning tree. Trajectories that break spanning tree edges, are not considered compatible. Since the vehicles move, the communication graph can change (Fig. 7.12 (right)). Consequently, the spanning tree recomputed in every cycle also changes over time. This allows the network to achieve different topology if it is required. Note that for an edge to be considered as a valid communication link, it must be retainable during a planning cycle given the dynamic motion constraints.

Discussion

This chapter describes a novel integration of sampling-based kinodynamic planners [LaV06a, LK05b] with message-passing protocols [KV06, PK04] to distributedly control the motion of multiple communicating vehicles. It is an extension of work on safe, real-time sampling-based planning [BK07] to the case of multiple networked vehicles with limited communication range. It can guarantee safety in terms of avoiding inevitable collision or loss of connectivity states. Compared to alternative approaches for decentralized motion planning [PSFB06, DKT06] it is easily implementable on general workspaces and to systems with different dynamics. In contrast to existing work on motion planning for dynamic networks, where coordination is centralized [CRL03], the approach is distributed.

Chapter 7

Experiments

This chapter presents the experiments conducted to test the effectiveness of the techniques in the previous four chapters. It is split into two parts: (i) experiments on informed planning, that focuses on the `IST` and `EVRC` algorithms and (ii) experiments on safe replanning, both for the single vehicle and multi-agent case. The first part include experiments conducted with a physics-based simulator, while the second part uses only the more traditional two-dimensional integrable model of a car-like system.

7.1 Informed Planning

Two-dimensional Integrable Models

The first set of experiments on informed planning has been run on the typical two-dimensional integrable models for mobile robots, such as car-like vehicles. We compare the proposed `IST` algorithm against the "Voronoi-bias" selection strategies of `RRT`. In these experiments, the same programming infrastructure and parameters have been used but different selection strategies are tested. Figure 7.1 displays the resulting trees for different selection strategies. Figure 7.2 provides averages over 10 experiments in these two scenes. . A trivial random selection policy fails to produce any path after 100,000 edges have been added to the tree. In order to implement the Voronoi-bias approach, we randomly sample points in the free part of the workspace and use a workspace distance metric to select the closest edge to them. The `RRT` approach offers good coverage of the state space but it is slow in reaching the target configuration. We have experimented with a version of `RRT` that is biased to promote exploration towards the target. In

this version, 20% of the time the state that is used to select the closest edge is a state in the target set. The value 20% gave the best results over different scenes. Although there is an improvement compared to the strictly coverage-oriented version of the RRT algorithm, the approach is still slow in reaching the goal configuration. On the other hand, the IST algorithm manages to aggressively search the state space towards the goal configuration. Considering the poor quality of the metric used, this is a very positive result. This behavior was consistent across all experiments.

Experiments with a physics-based simulator

In order to evaluate the efficiency of the proposed informed planning approach we have also executed experiments using a physically simulated car in a variety of scenes. The simulation environment is based on the Open Dynamics Engine (ODE) [Smi06]. Fig. 7.3 provides details about the simulation. The high-level controls for the car are acceleration and steering velocity, which are appropriately translated into control parameters to the joints that connect the wheels with the chassis.

The workspaces for which we provide comparisons in this chapter are displayed in Fig. 7.4. The first is an approximation of a known benchmark for motion planning, the bug-trap problem. The second problem, referred to as the iso-test problem. It requires the car to swerve between obstacles in a road-like environment, and it has been reported in the literature as a challenging case for sampling-based kinodynamic planners [BL06]. The last problem is the most challenging in terms of the workspace constraints since it is a maze-like environment.

For the nearest neighbor queries we use an efficient technique for approximate nearest neighbor search [YL07, PK06].

The algorithms that we compare against include three uninformed sampling-based kinodynamic planners, RRT [LK01b], Expansive Spaces [HLM99] and PDST [LK05a]. We have also experimented with the informed variant of RRT, called RRT-"goal bias", which 5% of iterations selects for expansion the state along the

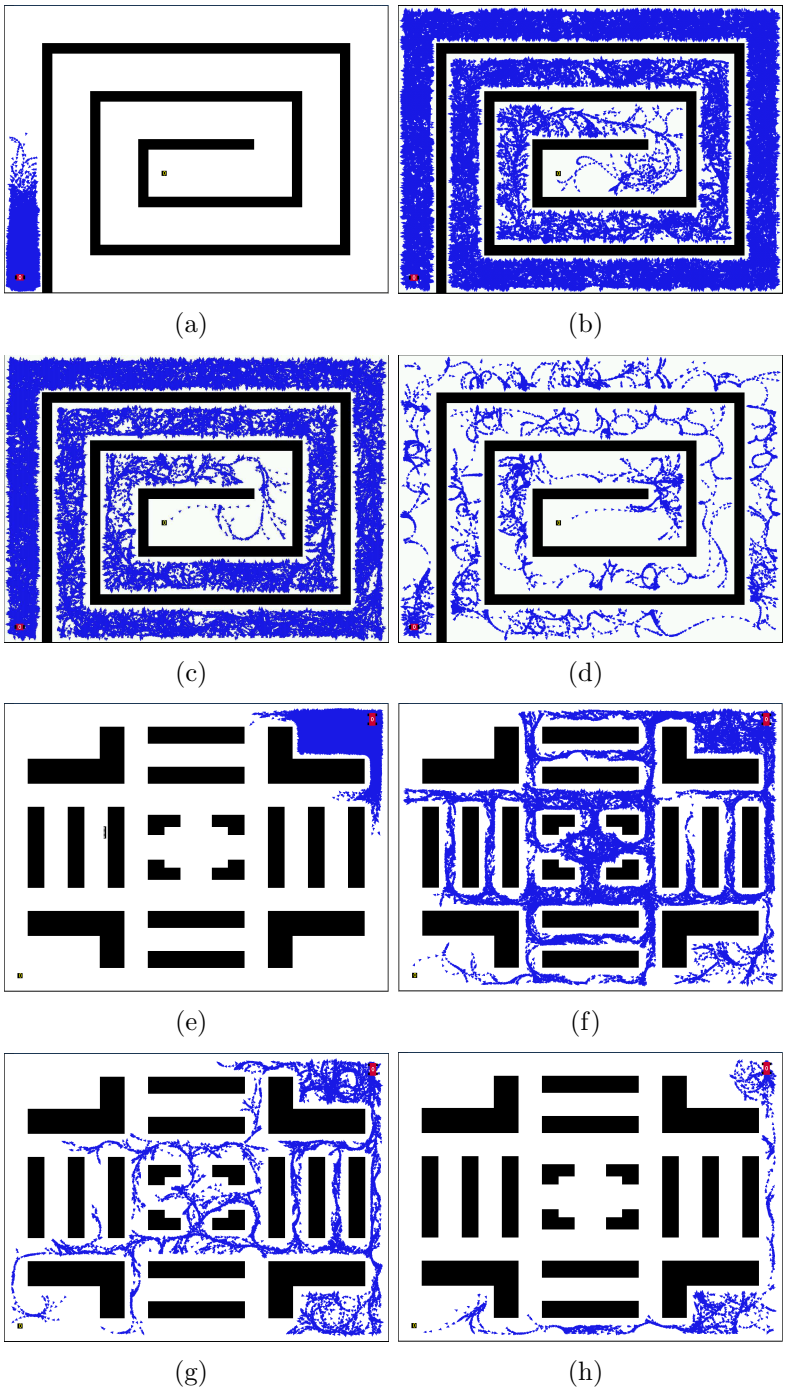

Figure 7.1 : Goal finding in (top row) scene meandros with a 2nd order differential drive-robot and (bottom row) scene labyrinth with an acceleration-bounded car-like robot: (a-e) a trivial random tree does not find the target after 100,000 iterations, (b-f) an `RRT-EXTEND` selection strategy finds the target after (top) 48,410 iterations and (bottom) 51,245 iterations (c-g) `RRT-EXTEND-BIAS`, where 20% of the time the target is the attractor, finds the target after (top) 42,855 iterations and (bottom) 17,212 iterations (d-h) `IST` reaches the target after (top) 13,774 edges and (bottom) 4,363 respectively.

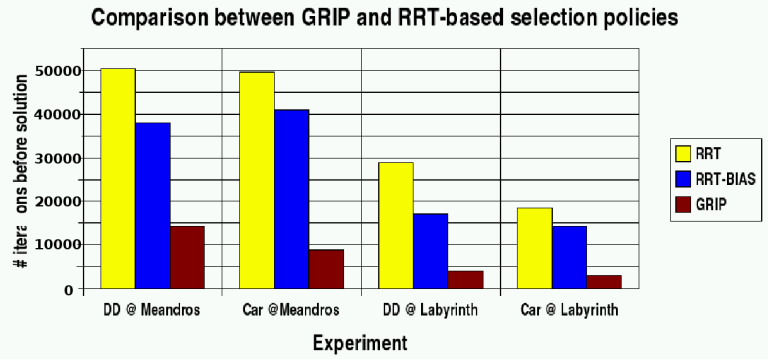

Figure 7.2 : Comparison between the IST selection/propagation scheme and Voronoi biased selections.

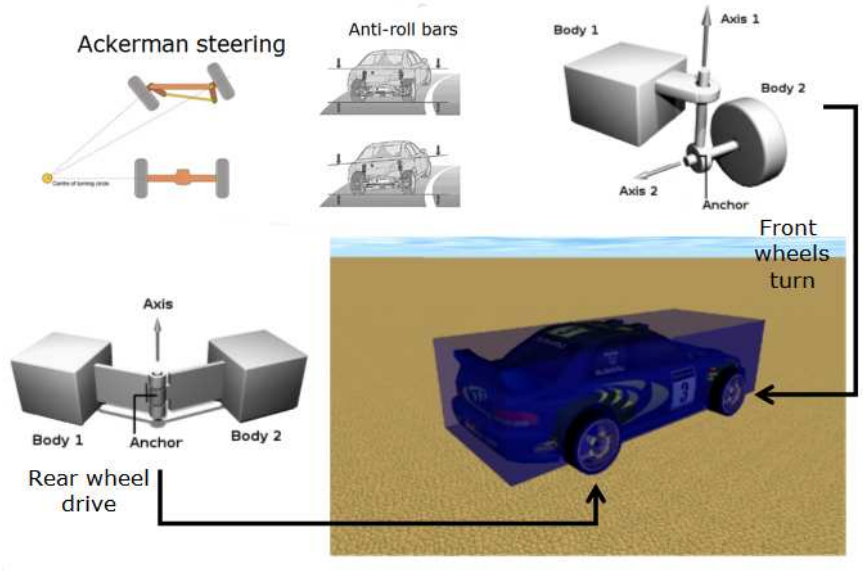

Figure 7.3 : We use a car-like system as our testbed in this work. The car is modeled as five rigid bodies, the chassis and the four wheels, connected through four joints. The front joints allow the wheels to be steerable, while the back joints allow the car to accelerate. In order to have a car that does not flip over often in the physics-based simulation we have to: (a) apply controls to the wheels that follow the Ackerman steering model and (b) to simulate the effect of anti-roll bars that real cars have.

tree closer to the goal. The last alternative we tested is the RRT* [US03] that integrates an A* like heuristic in the RRT algorithm.

The results of our experiments in terms of computation time are shown in Fig. 7.5. The cost for the computation of the probabilistic roadmap is included in the computation time of the IST. The proposed algorithm is able to outperform all

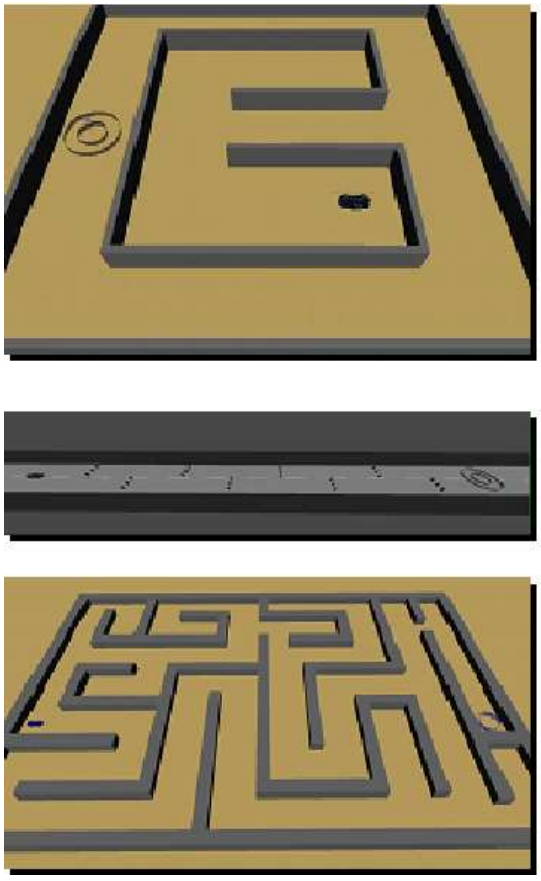

Figure 7.4 : The bug trap, iso-test and maze workspaces.

the alternative techniques in all workspaces. In some cases, the speedup is close to one order of magnitude. `IST` not only outperforms the uninformed planners but is also able to perform better than `RRT`-"goal bias" and `RRT`*. The best results are achieved for the maze-like environment. In this workspace a heuristic that uses workspace or $\mathcal{C}$ knowledge is considerably advantageous.

Fig. 7.6 shows a different statistic. It provides the average duration of a path computed by the algorithms on the maze. It is noteworthy, that although `IST` computes the solution faster, it is also able to compute a better path by taking advantage of the heuristic. The maze is large enough to allow us to see this difference. For the other two scenes, almost all of the algorithms computed trajectories of similar duration.

It must be noted that further improvement in path quality is to be expected by

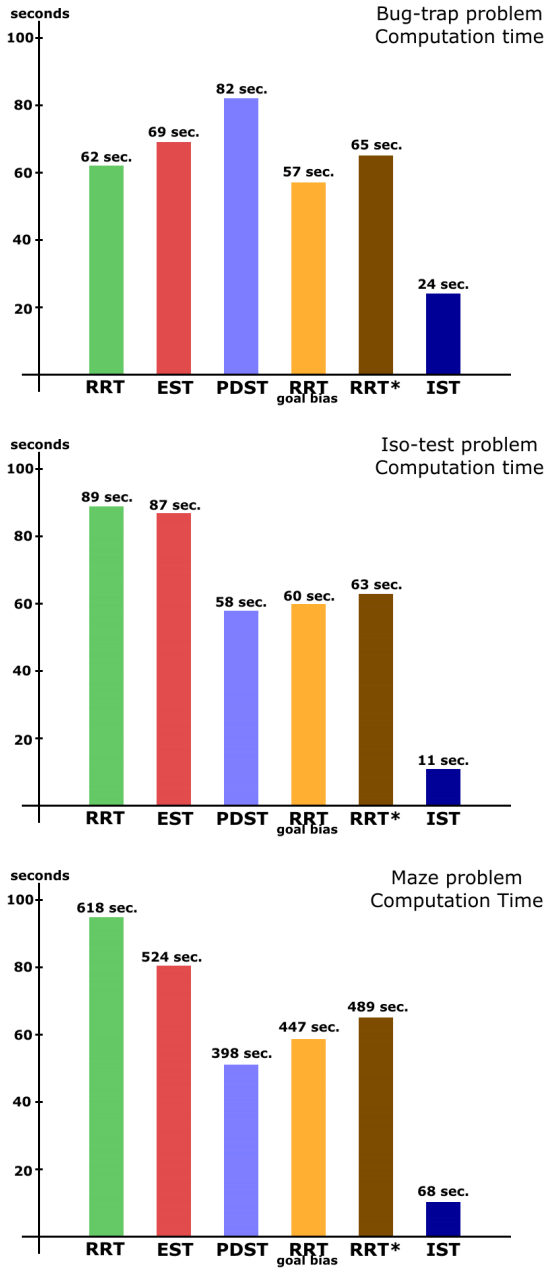

Figure 7.5 : Comparison of computation time on the bug trap, iso-test and maze workspaces. Averages of 50 experiments.

implementing an anytime approach as the Anytime RRT algorithm [FS06]. Anytime planners continue searching after a solution has been already found with the objective of improving path quality. The time to compute the first solution

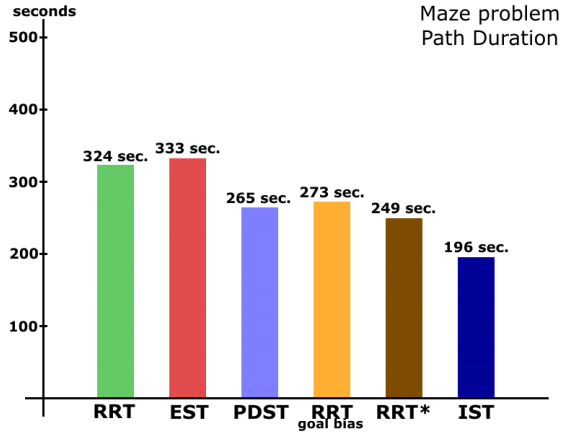

Figure 7.6 : Comparison of path duration on the maze scene. Averages over 50 experiments.

path with Anytime RRT is the same as with RRT, since the first algorithm uses the second as an initialization procedure. The hope is that since IST appears to perform better than RRT in problems with physically-simulated dynamics, an anytime version of IST will have similar advantages. We are planning to investigate the properties of an anytime version of IST in future work.

Discussion

Heuristic search can be very beneficial in guiding the operation of sampling-based planners in such challenging problems so as to avoid regression and focus the search in the part of the state space that is beneficial to the solution of a problem. Nevertheless, heuristic search in continuous spaces is not as straightforward as in traditional discrete AI problems. For example, it is not obvious how to scale the heuristic parameter against the true path cost. Furthermore, it is still very important that eventually the entire state space will be covered in order to be able to provide with the guarantee of probabilistic completeness.

This work emphasizes the importance of heuristic search in sampling-based kinodynamic planning and describes an algorithm, the Informed Subdivision Tree (IST), that achieves informed search, while still providing probabilistic complete-

Differential Drive

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{V}_L \\ \dot{V}_R \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot \frac{R}{2} \cdot (V_L + V_R) \\ \sin\theta \cdot \frac{R}{2} \cdot (V_L + V_R) \\ \frac{R}{2 \cdot L} \cdot (V_R - V_L) \\ \alpha_L \\ \alpha_R \end{pmatrix} \quad \begin{array}{c} |V| \leq 3\frac{m}{s} \\ |\dot{\theta}| \leq 20\frac{deg}{s} \\ |\alpha| \leq 0.6\frac{m^2}{s} \\ |\ddot{\theta}| \leq 3\frac{deg^2}{s} \end{array}$$

Car-like

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{V} \\ \dot{s} \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot \cos s \cdot V \\ \sin\theta \cdot \cos s \cdot V \\ \sin s \cdot v \\ \alpha \\ t \end{pmatrix} \quad \begin{array}{c} -0.5\frac{m}{s} \leq V \leq 3\frac{m}{s} \\ |s| \leq 4\frac{deg}{s} \\ \alpha \leq 0.6\frac{m^2}{s} \\ |t| \leq 1\frac{deg^2}{s} \end{array}$$

Table 7.1 : State Update Equations and Control Limits.

ness guarantees. The algorithm integrates the heuristic information in an adaptive subdivision scheme. The subdivision is used to appropriately discretize the state space and estimate in an online fashion the algorithm's performance in state space exploration. The heuristic information is used for the selection of the next cell in the subdivision from where the tree data structure of the sampling-based planner will be expanded from. At the cell level the algorithm operates in a depth-first manner. Then the algorithm moves on to select an edge within the cell given the best path cost from the starting state and a penalty factor. This means that at the level of selecting an edge within a cell the algorithm operates in a breadth-first manner. In this way, there is no need to scale the heuristic with the true path cost, while we still achieve an A* like behavior overall. Heuristic information is also employed in the propagation step of the algorithm. Experiments on various workspaces for a physically simulated system show that `IST` outperforms uninformed sampling-based kinodynamic planners as well as some existing informed variants.

7.2 Safe Replanning

A. Single Robot Simulations

We have experimented with three systems for testing replanning problems:

(1) A differential drive robot (DD-robot) with velocity controls V_L, V_R. This platform is reducible to a simpler holonomic robot, since we can retain the entire tree at each time step and the contingency plan is trivial: $V_L = V_R = 0$. **(2)** A DD-robot with acceleration controls α_L, α_R. The contingency plan is selected so that the wheel with the largest velocity magnitude is assigned maximum de-acceleration. The second wheel's acceleration is set so that its velocity reaches zero as the same time the first wheel stops. **(3)** A car-like robot that moves backwards and forwards with acceleration control α and steering velocity t. The contingency plan sets the de-acceleration parameter to its maximum value so as to reach a configuration with zero forward and steering velocities. Table 7.1 provides the state update equations for the last two systems and the bounds we have used for the controls. Parameters R and L are the radius of the wheel and the distance between a wheel and the robot's center. The total area that the robot must sense in all of the scenes is comparable. The sensing radius of the robot is equal to one tenth the width of the scenes.

The simulation component of our program is responsible for updating the map and transmitting it over socket communication to the planner. The integration of `STSR` with `IST` is executed on a different processor than the simulator and after the computation of a plan the planner communicates a sequence of controls back to the simulator. The planner was tested on an Athlon 1900MPs with one gigabyte of RAM.

Figure 7.7 shows the benefits of replanning with a selected duration for the next planning cycle. We have compared our algorithm that tests for safety only states that are T away from the root node of the tree with an approach that produces a tree where all the leaf nodes are safe. If the two approaches are provided with the same planning period, then `STSR` produces a much bigger tree, which allows the planner to better search the state-time space. The difference in the tree size is mainly due to the additional collision checking necessary to provide safety in the second case.

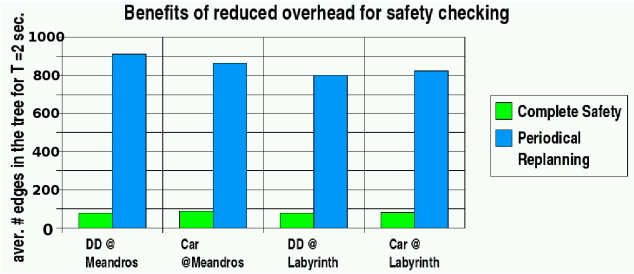

Figure 7.7 : Replanning with a known duration reduces the overhead of guaranteeing safety. For the same planning period STSR builds bigger trees.

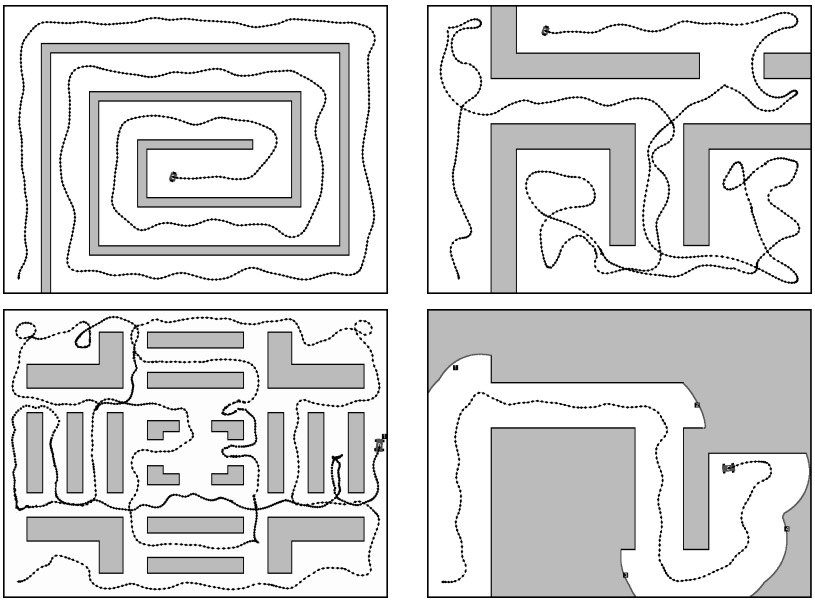

Figure 7.8 : Exploration of scenes (from left to right) "meandros", "rooms" with a DD-robot, "labyrinth" and "rooms" again with a car-like robot.

Performance for high-level tasks: mapping

Figure 7.8 provides a qualitative evaluation of the workspace exploration paths produced by STSR. The robot is initially positioned at the bottom left corner of a scene and knows only the part of the environment that it can sense. No collision was observed during our experiments. If the ICS avoidance step is removed from the planner, however, then the vehicle collides within a few seconds of execution. The paths appear smooth and the robots do not unnecessarily revisit parts of the space that are already covered. Figure 7.9 displays a velocity profile for an exploration procedure. The robot velocity remains for a large duration of the exploration procedure close to its maximum value and does not fluctuate

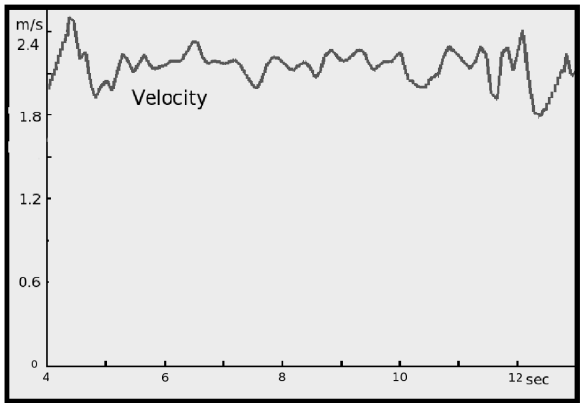

Figure 7.9 : The velocity profile for the car exploring "rooms" in Figure 7.8(d).

Time	DD-velocity	DD-acceleration	Car-like
Average Time in secs.	0.39	0.63	0.66
Maximum Time in secs.	0.90	1.57	1.19

Table 7.2 : Aver. cost in seconds to produce 250 edges.

considerably. Table 7.2 presents computational performance statistics. We ran 50 experiments for each robot and scene type and measure the time it takes for the planner to compute trees with 250 edges. We present: (a) the average time, (b) and the maximum time, that the planner requires to produce the tree. As expected systems with bounded acceleration are more difficult to plan for.

B. Distributed Simulations

Setup: We tested our algorithm on a distributed simulator that we developed and ran on an XD1 Cray cluster. The planner for each vehicle is running on a different processor and operates under time limitations imposed by a server that simulates ground truth. All data exchange is done via simple send and receive messages using sockets.

The simulated vehicular networks have been tested in three different environments. *Rooms* and *Random* are seen in Fig. 7.12. The first represents a structured environment with rooms and corridors, while the second is an unstructured environment. *Labyrinth* (Fig. 7.10) is a difficult scene that contains multiple narrow passages. Two types of vehicles have been tested, car-like robots, for which the dynamic equations are shown in Fig. 7.12, and differential drive

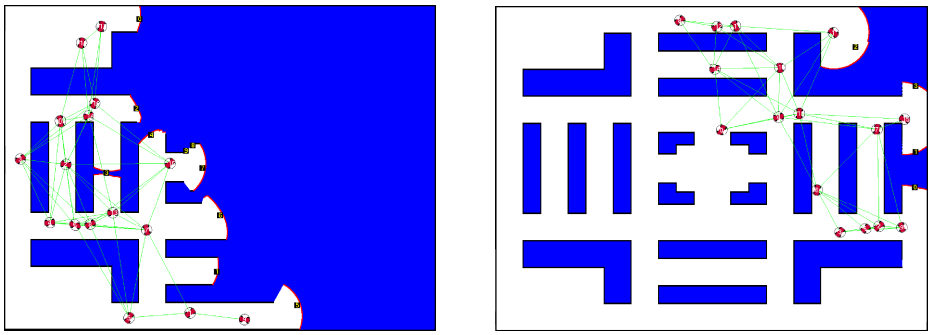

Figure 7.10 : Two snapshots of 16 vehicles exploring the labyrinth environment, while retaining a vehicular network.

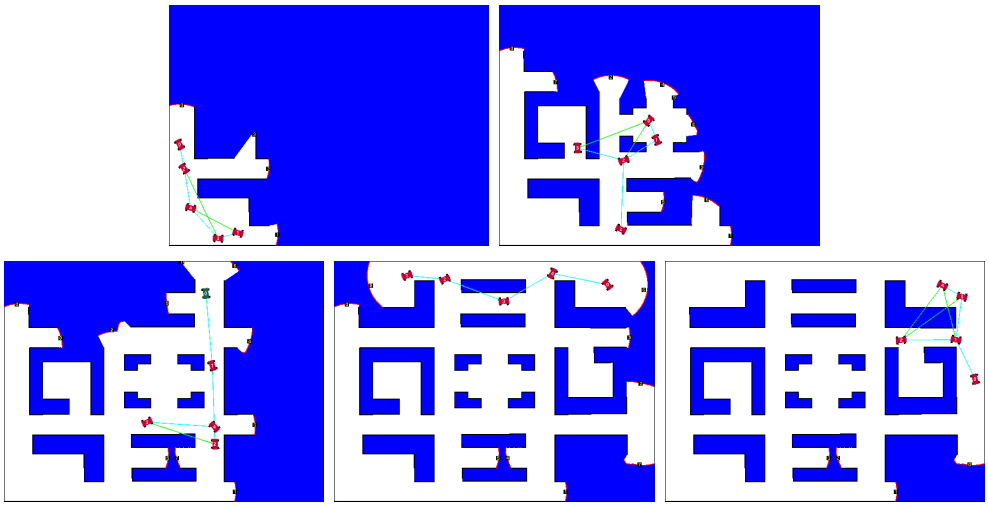

Figure 7.11 : Snapshots from an experiment in scene "labyrinth" with 5 vehicles, communication range at 25% of the scene width and sensing range at 15%.

robots with bounded acceleration. The car-like robots obey velocity bounds : $|V| \leq 3.5m/s$, and acceleration bounds: $\alpha \leq 0.8m^2/s$ as well as steering bounds: $|s| \leq 1deg/m$, $|t| \leq 4deg/s$. Vehicles have limited sensing and communication ranges. For contingencies, deacceleration maneuvers were used. The framework allows for plugging in other types of dynamics and contingencies.

We present here results from an application that combines many of the constraints we are interested in testing our algorithm. The vehicles have to solve a coordinated exploration task while retaining a network and avoiding collisions. They are initially located at the bottom left corner of the environment, close one

	Req 1		Req1 & Req2	
Nr Vehicles	1^{st} failure (sec)	success %	1^{st} failure (sec)	success %
2	287.10	10%	293.25	37.37%
4	21.00	0%	141.07	12.00%
8	3.67	0%	24.16	0%
16	3.00	0%	23.10	0%

Table 7.3 : Probability that networks of car-like vehicles succeed to explore when the first requirement only or the first two requirements are met.

	Req1 & Req3		All Requirements	
Nr Vehicles	1^{st} failure(sec)	success %	1^{st} failure(sec)	success %
2	113.10	0%	N/A	100%
4	21.53	0%	N/A	100%
8	4.31	0%	N/A	100%
16	3.00	0%	N/A	100%

Table 7.4 : Probability that networks of car-like vehicles succeed to explore when the seond requirement is not met or when all requirements are met.

to another, but at collision-free states forming a network with a single component. During each replanning cycle a simulated model builder and a task planner transmit to the vehicles the updated map and set of goals. The goals correspond to frontiers of the unexplored space and are assigned greedily so that large frontiers which are close to vehicles are being considered first. Experiments with up to 32 vehicles have been conducted.

We compare the max-plus algorithm against a simpler prioritized scheme described in more detail in [BTK07a]. In that scheme, the vehicles have unique global priorities. The planning step is the same as here. For the plan selec-

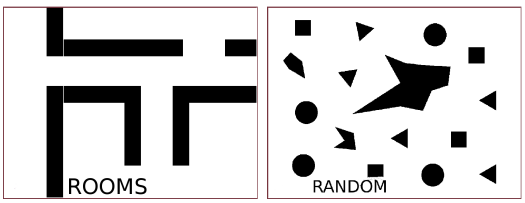

Figure 7.12 : Scenes “rooms” and “random”.

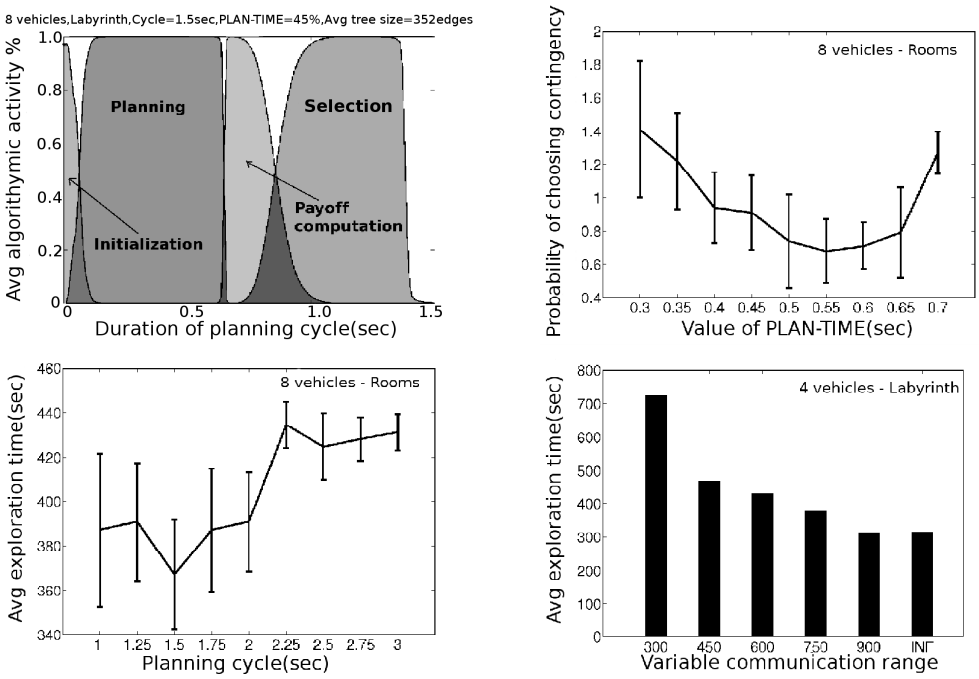

Figure 7.13 : Average activity profile during a cycle (left) and dependence on (from second to forth): `CYCLE_DURATION`, `PLAN_TIME`, and maximum communication range.

tion, each vehicle receives the choices of higher priority vehicles and then tries to choose its own plan so that it is compatible the higher priority neighbors. If no such plan exists, the vehicle chooses the contingency plan. At last the vehicle transmit its selection to its lower priority neighbors.

Feasibility: Table I exhibits the importance of the safety requirements in decoupled replanning. We measure the time (in seconds), that the vehicles can move without colliding with each other when Req. 2 and/or 3 (those necessary for safe multi-vehicle planning) are relaxed. The numbers reported show the time at which the first collision or loss of network connectivity occurs. The problem is so constrained for multiple vehicles, that often collisions cannot be avoided already since the 2nd replanning loop. The results are averaged out of 10 runs and are shown in columns labeled *failure*. If either one of the two requirements is absent, the vehicles will collide. When all requirements are enabled, then as expected, there is no failure. The columns labeled *success*, measure the percentage of suc-

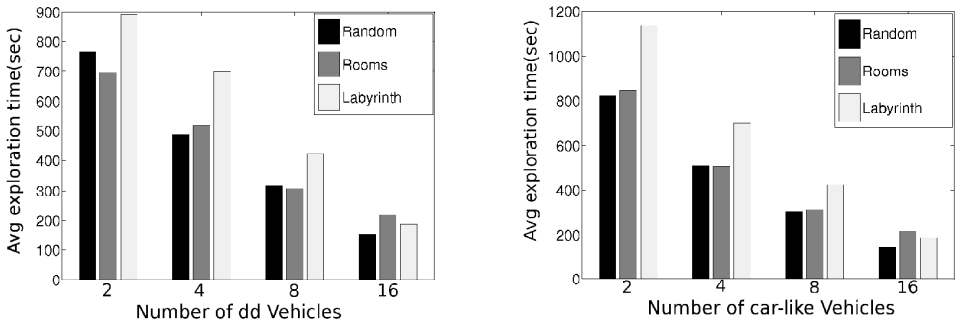

Figure 7.14 : Scalability results for three scenes: Random, Rooms, Labyrinth. Left: DD robots, Right: Car-Like robot

cessful exploration of the whole space without collisions. As we see, for small teams of 2 or 4 vehicles, there were some cases where the vehicles completed the task without one or both of the requirements. This is to be expected since the chances of an encounter are lower for such small teams.

Contingency Plans: One important advantage of the max-plus algorithm is that it avoids the use of priorities in coordination. In priority-based schemes, lower priority vehicles are overly constrained by the choices of higher priority agents. This may lead to frequent selection of contingency plans. We have experimented in scenes *Labyrinth* and *Rooms* for networks with 16 and 32 vehicles. Table II presents the number of times contingencies were selected using the simple prioritized scheme and max-plus. For 32 vehicles, max-plus chooses contingency plans considerably fewer times. Additionally, the results from the prioritized scheme have higher variation between different scenes and team sizes, while max-plus is much more consistent.

Scalability: The scalability properties of the algorithm are presented in Fig.

	Rooms		Labyrinth	
	16	32	16	32
Prioritized	3.61 %	24.5 %	1.35 %	8.42 %
Max-plus	0.98 %	2.26 %	3.04 %	4.84 %

Table 7.5 : Average Percentage of Cycles that V_i executes contingency.

7.14, which provides the average running times (10 runs per case) to complete exploration in the three scenes for car-like and DD vehicles. Increasing teams size from 2 to 16 results in 5 to 6 times faster exploration. This is a very encouraging result given that the simulated systems are very constrained, both due to network and kinodynamic constraints. Moreover, there is no significant variation in the performance of the algorithm when applied to systems with different dynamics.

Performance and Parameter Dependence: Fig. 7.13(left) shows the average activity profile of a vehicle during each cycle. The algorithm utilizes most of the replanning cycle in useful computations but the payoff computation takes up a non-trivial amount of time. In the selection step, max-plus does not let the processor idle, and in most cases is able to find an optimal or near optimal solution. The latter is confirmed by the second figure where we see the probability of executing contingency plans as a function of the portion of the planning cycle allocated to the motion planner (`PLAN-TIME`). This probability is very low $(0.7 - 1.5\%)$. Moreover, there is an optimum value at around 55%. For small `PLAN-TIME`, the planner has little time to produce enough plans, while for larger ones max-plus has not enough time to make a selection and the contingency is selected for safety reasons. The third figure shows that increasing the duration of the planning cycle can result in performance deterioration. The last figure shows that as the communication range increases, four vehicles finish the exploration faster, which is expected.

Chapter 8

Discussion

This chapter summarizes the contributions of this thesis and provides a discussion on certain limitations of the proposed techniques and related opportunities for future research.

8.1 Important Contributions

Chapters 3 through 6 introduced four new methodologies that address different motion planning challenges but which can be also combined to solve problems with multiple constraints (i.e., systems with physical constraints, replanning, distributed and scalable coordination).

Informed Kinodynamic Planning

The first contribution relates to the problem of planning motions for systems that have non-trivial dynamics and must obey differential constraints. The running example in this work is a car-like vehicle with significant drift. This is a non-holonomic system with small control influence (acceleration bounds) over momentum (velocity bounds). We proposed a new method that balances informed and methodical sampling-based kinodynamic planning in this context, called Informed Subdivision Tree (`IST`). The algorithm has been designed with the following points in mind:

- Its operation is compatible with the use of physics-based simulators for modeling the workspace and the moving systems.
- It is also designed so as to maximize the utilization of any available heuristic, given workspace, domain or query knowledge.

- While informed, the algorithm is also methodical in its search operation, and provides the guarantee of probabilistic completeness.

- It automatically adapts its behavior between informed search and coverage-oriented search without manual intervention and while minimizing the dependence on parameters.

In simple parts of the state space `IST` is greedily guided by the available heuristic. In constrained parts, such as narrow passages, where the heuristic may not be beneficial, the algorithm automatically explores alternative routes for a solution. This behavior is a result of a combination of algorithmic tools. An adaptive subdivision scheme is used to estimate online the algorithm's performance in exploring the entire state space without depending on the existence of a proper metric. An edge penalization method minimizes regression issues, while an offline procedure is employed for caching promising motions for given states that maximize path diversity. `IST` not only solves many physically-simulated planning problems faster, it also results in better solution trajectories, as the algorithm promotes the expansion of trajectories with smaller path cost, in a manner similar to the operation of the A* algorithm [HNR68].

$\mathcal{C}$-space Guidance

This thesis also presents a way for designing a general heuristic for kinodynamic problems. The idea is to utilize the inherent separation between kinematic and dynamic constraints without the limitations of decoupled approaches. Kinematic information is used as a heuristic to bias the expansion of `IST` in the state space. To achieve this, we propose building a graph in $\mathcal{C}$ that has the properties of visibility-based roadmaps [SLN00] and roadmaps with useful cycles [NO04]:

(i) it is possible to connect every configuration to the roadmap with a collision-free path and

(ii) the roadmap is able to represent every homotopy class of paths in the $\mathcal{C}$

One of the disadvantages of existing algorithms for constructing such roadmaps is that they are slow to compute. We describe in this thesis a method for an approximate construction of such a roadmap that reduces computation time at the expense of creating a roadmap with more nodes than previous algorithms [SLN00, JSss]. The resulting technique is able to compute a roadmap that according to experimental results provides good visibility and path coverage properties, while the construction of the graph consumes a smaller amount of time than competitive algorithms.

Safe Replanning for Systems with Drift

The third contribution involves the problem of replanning under dynamic constraints, especially for tasks that involve partial observability and for systems with significant drift. The proposed technique, Short-Term Safety Replanning (`STSR`), provides safety guarantees for collision avoidance even under limited computation time. `STSR` is a general framework for such applications by extending previous work on kinodynamic planning [LK05a, LK01a, HKLR02, FA04, FDF02, FKS06] and uses new ideas to achieve good performance.

In particular, `STSR` manages to reduces the amount of collision checking necessary for providing safety guarantees. It can be also integrated with `IST` in order to take advantage of informed planning and reuse computations from previous planning cycles. The simulated experiments suggest favorable computational performance against alternatives and the computation of smooth, safe paths.

Distributed Safe Kinodynamic Replanning

Finally, the last contribution extends the above safe replanning framework to the case of multi-agent cooperating systems. It corresponds to a novel integration of sampling-based kinodynamic planners with message-passing protocols for the distributed solution of planning problems that involve vehicles with dynamic constraints. It extends the work on safe, real-time sampling-based planning to the case of multiple communicating vehicles. The method provides safety guarantees in terms of collision avoidance as well as in terms of retaining a connected communication network. The algorithm has been implemented on a distributed simulator and the results on vehicles with acceleration constraints confirm the safety properties of the approach in a workspace exploration application. A comparison over priority-based schemes shows that the distributed protocol offers improved scalability. The proposed method allows for plugging in other types of dynamic constraints and can also be integrated with higher-level approaches for distributed task assignment and distributed state estimation.

8.2 Limitations and Future Directions

As mentioned in the introduction, the motion planning field is in the process of a transition from improving algorithms for the traditional geometric model of planning to investigating increasingly more challenging problems. The work in this thesis, as part of this transition, has investigated some interesting problems that go beyond geometric path planning and proposed some new tools to deal with them. Nevertheless, there are many issues that still have to be addressed in order for these techniques to be applicable to real systems, either physical or simulated ones. This section lists and outlines some of those issues, which also correspond to exciting opportunities for future research that closely interface with the work in this thesis.

Uncertainty and Errors

All the planning and replanning tools in this work assumed that an action selected by the planner is executed perfectly by the system. While this may be true in autonomous agents in simulations, this is typically not the case with real physical systems. It is, however, important to note that replanning can be also used as a tool that deals with uncertainty. By reducing the planning cycle, the hope is that the errors become smaller between cycles and the planning algorithms are able to achieve state estimates with smaller errors. Nevertheless, an important research direction that has the potential to directly address issues related to uncertainty is the integration of planning tools like those proposed in this work with algorithms for probabilistic, Bayesian estimations, such as Kalman filters and particle filters.

Path Quality and Anytime Planning

Path optimality is an issue that is typically overlooked in the motion planning literature due to the computational complexity of the problem, focusing instead on feasibility. However, it is very important in many applications, especially in visualization and computer games, where the agents are expected to move along smooth paths. A search-based approach that is promising in this direction, is anytime planning. The idea of anytime planning is inspired by anytime search algorithms, which first compute a solution and then attempt to incrementally improve its quality. An anytime equivalent for `IST` should also be taking advantage of the algorithm's tools, such as the subdivision and the heuristic, especially if it comes in the form of a roadmap-based heuristic, in order to improve the quality of the path.

Integration with Prediction Modules

In order to be able to deal effectively with problems that involve mobile and dynamic obstacles it is necessary to properly integrate motion planners with sensor-based prediction modules. Such algorithms are necessary in replanning tasks in order to define the regions in the state-time space that correspond to obstacles. If they are not available, then a planning algorithm can either employ simplistic prediction models (i.e., treat moving obstacles as static ones or assume they are going to retain current velocity and path) or resort to conservative approximations (i.e., all valid velocities, accelerations and directions of motions are possible). In the first case, planning among moving obstacles will often result in collisions, while the second case is too conservative and causes the moving system to stop and falsely assume that an impeding collision is about to occur. Efficient sensor-based prediction models should lie somewhere in between these two approaches, allowing for a more opportunistic planning that has low probabilities of collision with the moving obstacles.

Effects of Realistic Communication

For the problem of distributed motion coordination, we have assumed perfect communication within a predefined radius. In reality, however, the communication range is neither regular nor predictable. One way that the existing technique could still be applied is through the use of a precomputed communication map of the environment. Such a map would provide with a way to infer given the positions of two vehicles whether they can communicate or not, at least up to a specific level of confidence. Another challenge that realistic communication would impose to the algorithm is related to bandwidth. In the simulations presented here bandwidth was virtually unlimited compared to bandwidth in a wireless setup. It would be interesting to study how the algorithm can operate under a limited bandwidth scenario. Furthermore, the scheme described in Chapter 6 assumes a level of synchronization between the different units in the environment. An asynchronous version of the protocol, where the different vehicles do not start

and end their planning cycles at the same point in time, is a very interesting direction for future research. Finally, a question that arises in this context, is whether the technique scales to hundreds or even thousands of systems operating in parallel.

Interactive Planning

It is often the case in visualization applications that a user controls the motion of a simulated character. As the complexity of such characters increases, it is desirable for the human users to specify only a high-level motion control (e.g., direction and speed of motion). In order for the motion of the character to appear realistic, however, a planner is needed to specify the low level motion control so as to achieve collision avoidance, safety, satisfaction of complex motion constraints, adaptability to a constrained terrain, while conflicting minimally with the user's selections.

Novel Physics-based Planning Problems

The type of physics-based simulation that has been used in this work deals with articulated rigid bodies. Nevertheless, there are many interesting planning problems that involve fluids and deformable body simulations. For example, physically realistic planning for boats or manipulating fabrics in the industrial sector. To approach such problems it is necessary to integrate motion planners with new types of physics-based simulators and adapt the techniques to the characteristics of deformable and fluid objects.

Hierarchical Planning

There has been significant work in the motion planning literature on how to influence the construction of a configuration space roadmap using workspace information. In this work, we presented an incorporation of the $\mathcal{C}$-space heuristic in the operation of the kinodynamic planner. Taking these two lines of work together can lead in the definition of a hierarchical model for kinodynamic planning. In this hierarchy, a high-level and very fast planner first solves problems in the workspace. The output is then fed to the $\mathcal{C}$-space level, which constructs a roadmap with the visibility and path deformation properties. The roadmap's output is used in the kinodynamic planner as a heuristic function.

This hierarchical model can also take advantage of advances in CPU and GPU technology and the presence of multiple cores in a system. Each level could be running at a different core with a different frequency and they can be communicating asynchronously. It is very interesting to investigate how such a hierarchical scheme operates in replanning applications, where the high and workspace level module can be updated very fast given sensory data and the following levels have a smaller frequency of operation.

Other types of Systems and Applications

The tools developed in this thesis could be also applied in problems involving different types of systems. For example, articulated systems with dynamics is one possible direction that is different from the type of mobile vehicles studied here. Although the physically simulated vehicles in section 7.1 are actually five body articulated systems, examples such as manipulator arms and humanoids, involve additional constraints and challenges. Moreover, there are two very interesting categories of motion planning applications that this work could also be applied. In sensor-based tasks (i.e., monitoring problems), the planner would be responsible to control not only motions but also sensing operations. In multi-agent applications, agents can act as adversaries (e.g., pursuit-evasion) and it is very interesting to study the interaction between planning and game theory.

The above areas involve some very exciting challenges and opportunities for further research, which often relate to a variety of many different fields. They also depend critically on the existence of robust and efficient motion planners. Thus, an interest in the core computational capabilities of motion planners, which this work has also focused on, will continue into the future.

Bibliography

[ABD+98] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. pages 155–168, 1998.

[AGM98] J.-M. Ahuactzin, K. Gupta, and E. Mazer. Manipulation planning for redundant robots: A practical approach. 17(7):731–747, July 1998.

[ATBM92] Jean-Manuel Ahuactzin, El-Ghazali Talbi, Pierre Bessière, and Emmanuel Mazer. Using genetic algorithms for robot motion planning. pages 671–675, 1992.

[BATM94] Pierre Bessière, Juan-Manuel Ahuactzin, El-Ghazili Talbi, and Emmanuel Mazer. The 'Ariadne's clew' algorithm: Global planning with local methods. pages 39–47, 1994.

[BB07] B. Burns and O. Brock. Single-query motion planning with utility-guided random trees. In *Proc. of the IEEE Intl. Conference on Robotics and Automation*, Rome, Italy, April 2007.

[BBT02] M. Bennewitz, W. Burgard, and S. Thrun. Finding and optimizing solvable priority schemes for decoupled path planning for teams of mobile robots. *Robotics and Autonomous Systems*, 41(2):89–99, 2002.

[BCL+03] K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plaku, and L. E. Kavraki. Multiple query probabilistic roadmap planning using single query planning primitives. In *2003 IEEE/RJS International Conference*

on Intelligent Robots and Systems (IROS), pages 656–661, Las Vegas, NV, October 2003.

[BDG85] J. Bobrow, S. Dubowsky, and J. Gibson. Time-optimal control of robot manipulators. *Int. Journal of Robotics Research*, 4(3), 1985.

[BFK06] J. van den Berg, D. Ferguson, and J. Kuffner. Anytime path planning and replanning in dynamic environments. In *IEEE ICRA*, pages 2366–2371, May 2006.

[BK91] O. Brock and O. Khatib. Elastric strips: A framework for integrated planning and execution. In *1999 International Symposium on Experimental Robotics*, 1991.

[BK00] R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In *IEEE ICRA*, pages 521–528, San Fransisco, CA, April 2000.

[BK07] K. E. Bekris and L. E. Kavraki. Greedy but safe replanning under kinodynamic constraints. In *ICRA*, Rome, Italy, April 2007.

[BL91] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, 10(6):628–649, December 1991.

[BL06] F. Boyer and F. Lamiraux. Trajectory optimization applied to kinodynamic motion planning for a realistic car model. In *IEEE Intl. Conf. on Robotics and Automation*, pages 487–492, May 2006.

[BM02] D. J. Balkcom and M. T. Mason. Time optimal trajectories for bounded velocity differential drive vehicles. 21(3):199–217, 2002.

[BMSS05] W. Burgard, M. Moors, C. Stachniss, and F. Schneider. Coordinated multi-robot exploration. *IEEE TR*, 21(3), 2005.

[Boh01] Robert Bohlin. Path planning in practice: Lazy evaluation on a multi-resolution grid. 2001.

[BOvdS99] Valérie Boor, Mark H. Overmars, and A. Frank van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planner. pages 1018–1023, 1999.

[BP83] R. Brooks and T. Lozano Perez. A subdivision algorithm in configuration space for findpath with rotation. pages 799–803, 1983.

[BTK07a] K. E. Bekris, K. I. Tsianos, and L. E. Kavraki. A decentralized planner that guarantees the safety of communicating vehicles with complex dynamics that replan online. In *IROS (submitted)*, 2007.

[BTK07b] K. E. Bekris, K. I. Tsianos, and L. E. Kavraki. A distributed protocol for safe real-time planning of communicating vehicles with second-order dynamics. In *ROBOCOMM*, 2007.

[BV06] J. Bruce and M. Veloso. Safe multi-robot navigation within dynamic constraints. *Proc. of the IEEE*, 94(7):1398–1411, 2006.

[Can88] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, 1988.

[CBR02] M. Christopher Clark, Tim Bretl, and Stephen Rock. Applying kinodynamic randomized motion planning with a dynamic priority system to multi-robot space systems. In *Aerospace Conference*, 2002.

[Cha87] B. Chazelle. Approximation and decomposition of shapes. In J. T. Schwartz and C. K. Yap, editors, *Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[CL02] P. Cheng and S. LaValle. Resolution complete rapidly-exploring random trees. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 267–272, 2002.

[CL03] Peng Cheng and Steve M. LaValle. Exploiting group symmetries to improve precision in kinodynamic and nonholonomic planning. 2003.

[CLH+05] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms and Implementation*. MIT Press, Boston, 2005.

[CR87] J. Canny and J. Reif. New lower bound techniques for robot motion planningproblems. pages 49–60, 1987.

[CRL03] C. Clarc, S. Rock, and J.-C. Latombe. Dynamic networks for motion planning in multi-robot space systems. In *Intl. Symp. of Artificial Intelligence, Robotics and Automation in Space*, 2003.

[CRR91] J. Canny, A. Rege, and J. Reif. An exact algorithm for kinodynamic planning in the plane. *Discrete and Computational Geometry*, 6:461–484, 1991.

[CSL01] Peng Cheng, Zuojun Shen, and Steven M. LaValle. RRT-based trajectory design for autonomous automobiles and spacecraft. *Control Sciences*, 11(3-4):51–78, 2001.

[dBvKOS00] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications, 2nd Ed.* Springer-Verlag, Berlin, 2000.

[DK07] R. Diankov and J. J. Kuffner. Randomized statistical path planning. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'07)*, San Diego, CA, 2007.

[DKT06] D. V. Dimarogonas, K. J. Kyriakopoulos, and D. Theodorakatos. Totally distributed motion control of sphere world multi-agent systems using decentralized navigation functions. *ICRA*, 2006.

[DLOS98] A. De Luca, G. Oriolo, and C. Sampson. *Feedback Control of a Nonholonomic Car-lie Robot*, chapter Robot Motion Planning and Control, pages 171–253. Lecture Notes in Control and Information Sciences. Springer, NY, 1998.

[DXCR93] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. In *Journal of the ACM*, volume 40, pages 1048–1066, 1993.

[DZKS06] M. B. Diass, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: a survey and analysis. *Proc. of the IEEE*, 94(7):1257–1270, July 2006.

[EHS01] M. Egerstedt, X. Hu, and A. Stotsky. Control of mobile platforms using a virtual vehicle approach. *IEEE Transactions on Automated Control*, 46(4):1777–1782, November 2001.

[EL00] S. Ehmann and M. C. Lin. Swift: Accelerated distance computation between convex polyhedra by multi-level marching. 2000.

[FA04] T. Fraichard and H. Asama. Inevitable collision states - a step towards safer robots? *Advanced Robotics*, 18(10):1001–1024, 2004.

[FDF02] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control and Dynamics*, 25(1):116–129, 2002.

[FKS06] D. Ferguson, N. Kalra, and A. Stentz. Replanning with rrts. In *ICRA*, 2006.

[FS06] D. Ferguson and A. Stentz. Anytime RRTs. In *IEEE/RSJ Intelligent Robots and Systems (IROS-06)*, pages 5369–5375, Beijing, China, Oct. 2006.

[FW88] S. Fortune and G. Wilfong. Planning constrained motion. In *STOC*, Chicago, 1988.

[GG95] K. Gupta and Z. Guo. Motion planning with many degrees of freedom: sequential search with backtracking. *IEEE Transactions on Robotics and Automation*, 6(11):897–906, 1995.

[GHK99] L. J. Guibas, C. Holleman, and L. E. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, volume 1, pages 254,259, October 1999.

[GKP02] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. In *NIPS-14*. MIT Press, 2002.

[GKX07] R. Gayle, K. R. Klinger, and P. G. Xavier. Lazy reconfiguration forest: An approach for planning with multiple tasks in dynamic environments. In *ICRA*, pages 1316–1323, Rome, April 10-14 2007.

[GLM96] S. Gottschalk, M. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. In *Proc. ACM SIGGRAPH'96*, pages 171–180, 1996.

[GM02] B. P. Gerkey and M. J. Mataric. Sold!: Auction methods for multi-robot coordination. *IEEE TRA*, 18(5):758–786, Oct 2002.

[GO97] J. Goodman and J. O'Rourke. *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.

[GO02] R. J. Geraerts and M. H. Overmars. A comparative study of probabilistic roadmap planners. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Nice, France, 2002.

[HA88] Yong Koo Hwang and Narendra Ahuja. Path planning using a potential field representation. Technical report, University of Illinois, October 1988.

[HA92] Y. K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, February 1992.

[HK00] Christopher Holleman and Lydia E. Kavraki. A framework for using the workspace medial axis in PRM planners. pages 1408–1413, 2000.

[HKL+98] D. Hsu, L.E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. pages 143–153, 1998.

[HKLR02] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *Intl. Journal of Robotics Research*, 21(3):233–255, 2002.

[HLM99] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications*, 9(405):495–512, 1999.

[HNR68] P. E. Hart, N. J. Nillson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Syst. Sci. Cybenetics (SSC-4)*, 2:100–107, 1968.

[Hol83] J. M. Hollerbach. Dynamic scaling of manipulator trajectories. Technical Report Memo 700, MIT AI Lab, 1983.

[Hom] Dedicated Short Range Communications (DSRC) Home. http://www.leearmstrong.com/dsrc/dsrchomeset.htm.

[HS96] D. Halperin and M. Sharir. A near-quadratic algorithm for planning the motion of a polygon in a polygonal environment. *Discrete and*

Computational Geometry, 16:121–134, 1996.

[HSS84] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: PSPACE-hardness of the "warehouseman's problem". 3(4):76–88, 1984.

[JSss] L. Jaillet and T. Simeon. Path deformation roadmaps. *The International Journal of Robotics Research*, 2008 (in press).

[Kav95] Lydia E. Kavraki. *Random Networks in Configuration Space for Fast Path Planning*. PhD thesis, Stanford University, January 1995.

[KD86] S. Kambhampati and L. S. Davis. Multiresolution path planning for mobile robots. 2(3):135–145, September 1986.

[Kha86] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[KJCL97] M. Khatib, H. Jaouni, R. Chatila, and J.-P. Laumond. Dynamic path modification for car-like nonholonomic mobile robots. In *Intl. Conference on Robotics and Automatiion*, pages 2920–2925, Albuquerque, NM, April 1997.

[KKL96] L. E. Kavraki, M. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA '96)*, pages 3020–3026, Minneapolis, MN, 1996.

[KL94] Lydia E. Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration space for fast path planning. volume 3, pages 2138–2145, 1994.

[KLMR96] L. E. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan. Randomized query preprocessing in robot motion planning. 1996.

[KM04] M. Kallman and M. Mataric. Motion planning using dynamic roadmaps. In *ICRA-04*, volume 5, 2004.

[Kod89] D. E. Koditschek. Robot planning and control via potential functions. In *The Robotics Review 1*, pages 349–367. MIT Press, 1989.

[KPLM98] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shell: A higher-order bounding volume for fast proximity queries. 1998.

[KSLO96] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE TRA*, 12(4):566–580, Aug. 1996.

[KV06] J. R. Kok and N. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7:1789–1828, 2006.

[KVdP06] M. Kalisiak and M. Van de Panne. RRT-Blossom: RRT with A Local Flood-Fill Behavior. In *IEEE Intl. Conf. on Robotics and Automation*, 2006.

[Lat91] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[LaV06a] S. LaValle. *Planning Algorithms*. Cambrdidge university Press, 2006.

[LaV06b] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[LB02] S. M. LaValle and M. S. Branicky. On the relationship between classical grid search and probabilistic roadmaps. 2002.

[LBL04] F. Lamiraux, D. Bonnafous, and O. Lefebvre. Reactive path deformation for nonholonomic mobile robots. *IEEE TR*, 20(6):967–977, 2004.

[LC91] M. C. Lin and J. F. Canny. Efficient algorithms for incremental distance computation. pages 1008–1014, 1991.

[LDK04] S. Loizou, D. Dimarogonas, and K. Kyriakopoulos. Decentralized feedback stabilization of multiple nonholonomic agents. In *ICRA*, volume 3, pages 3012–3017, 2004.

[LFV04] F. Lamiraux, E. Ferre, and E. Vallee. Connecting exploration trees using trajectory optimization methods. In *ICRA*, pages 3987–3992, April 2004.

[LK99] Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. pages 473–479, 1999.

[LK01a] S. LaValle and J. Kuffner. Rapidly exploring random trees: Progress and prospects. In *WAFR*, pages 293–308, 2001.

[LK01b] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *IJRR*, 20(5):378–400, May 2001.

[LK02] Andrew M. Ladd and Lydia E. Kavraki. Generalizing the analysis of PRM. pages 2120–2125, 2002.

[LK04] A. M. Ladd and L. E. Kavraki. Measure theoretic analysis of probabilistic path planning. *IEEE TRA*, 20(2):229–242, April 2004.

[LK05a] A. M. Ladd and L. E. Kavraki. Fast tree-based exploration of state space for robots with dynamics. In *WAFR*, pages 297–312, 2005.

[LK05b] A. M. Ladd and L. E. Kavraki. Motion planning in the presence of drift, underactuation and discrete system changes. In *Robotics Science and Systems-2005*, June 2005.

[LL96] Florent Lamiraux and Jean-Phillippe Laumond. On the expected complexity of random path planning. pages 3306–3311, 1996.

[LM91] M. C. Lin and D. Manocha. Fast interference detection between geometric models. *The Visual Computer*, 11(10):542–561, 1991.

[MSTY05] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. *Artificial Intelligence Journal*, 161(1-2):149–180, 2005.

[Mur07] R. M. Murray. Recent reseach in cooperative control of multi-vehicle systems. *(submitted) ASME Journal of Dynamic Systems, Measurement, and Control*, 2007.

[Nil69] N. J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *1st International Conference on Artificial Intelligence*, pages 509–520, 1969.

[NO04] D. Nieuwenhuisen and M. H. Overmars. Useful cycles in probabilistc roadmap graphs. In *IEEE Intl. Conf. on Robotics and Automation*, pages 446–452, 2004.

[O'D87] C. O'Dunlaing. Motion planning with inertial constraints. *Algorithmica*, 2(4):431–475, 1987.

[OFL04] P. Ogren, E. Fiorelli, and N. E. Leonard. Cooperative control of mobile sensor networks: Adaptive gradient climbing in distributed environments. *IEEE Tr. on Aut. Control*, 49(8):1292–1302, 2004.

[O'R04] J. O'Rourke. Visibility. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 643–663. Chapman and Hall/CRC Press, New York, 2004.

[OS06] R. Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Tr. on Aut. Control*, 51(3):401–420, 2006.

[Ov94] Mark H. Overmars and Petr Švestka. A probabilistic learning approach to motion-planning. pages 19–37, 1994.

[OY82] C. O'Dunlaing and C. K. Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, 6:104–111, 1982.

[PBC+05] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE TRA*, 21(4):587–608, 2005.

[PDKC03] G. A. S. Pereira, A. K. Das, V. Kumar, and M. F. M. Campos. Decentralized motion planning for multiple robots subject to sesnsing and communication constraints. In *Work. on Multi-Robot Systems*, 2003.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[Per83] T. Lozano Perez. Spatial planning: a configuration space approach. February 1983.

[PF05] S. Petti and T. Fraichard. Partial motion planning framework for reactive planning within dynamic environments. In *AAAI Intl. Conf. ICAR*, Barcelona, Spain, September 2005.

[PK04] K. Plarre and P. R. Kumar. Extended message passing algorithm for inference in loopy gaussian graphical models. *Ad Hoc Networks*, 2:153–169, 2004.

[PK06] E. Plaku and L. E. Kavraki. Quantitative analysis of nearest-neighbors search in high-dimensional sampling-based motion planning. In *Intl. Workshop on Algorithmic Foundations of Robotics (WAFR)*, New York City, NY, 2006.

[PSFB06] L. Pallotino, V. G. Scordio, E. Frazzoli, and A. Bicchi. Decentralized and scalable conflict resolution strategy for multi-agent systems. In *Int. Symp. on Mathematical Theory of Networks and Systems*, 2006.

[QK93] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *Proc. IEEE Int. Conf. on Rob. and Autom.*, pages 00–00, 1993.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. 1994.

[RBK07] M. Rickert, O. Brock, and A. Knoll. Balancing exploration and exploitation in motion planning. In *Proc. of the International Conference on Robotics and Automation, ICRA-07*, 2007.

[Rei79] J. H. Reif. Complexity of the Generalized Mover's Problem. In *20th Annual IEEE Symposium on Foundations of Computer Science*, pages 421–427, San Juan, Puerto Rico, October 1979.

[RK92] E. Rimon and D. Koditschek. Exact Robot Navigation Using Artificial Potential Functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, Oct. 1992.

[RS90] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific J. Math.*, 145(2):367–393, 1990.

[SA01] G. Song and N. M. Amato. Randomized motion planning for car-like robots with c-prm. In *IROS*, pages 37–42, Maui, Hawaii, Nov. 2001.

[SAS84] Micha Sharir and Elka Ariel-Sheffi. On the piano movers' problem: IV. various decomposable two-dimensional motion planning problems. 37:479–493, 1984.

[SB95] G. Singh and A. J. Bernstein. A highly asynchronous minimum spanning tree protocol. *Distributed Computing*, 8(3):151–161, March 1995.

[SBD+02] E. Schmitzberger, J. L. Bouchet, M. Dufaut, W. Didier, and R. Husson. Capture of homotopy classes with probabilistic road map. In *IEEE/RSJ Int. Conf. on Robots and Systems*, 2002.

[Sch87] H. M. Schaettler. On the optimality of bang-bang trajectories in $\mathcal{R}^3$. *Bull. AMS*, 16(1):11–36, 1987.

[SD88] Z. Shiller and S. Dubowsky. Global time-optimal motions of robotic manipulators in the presence of obstacles. In *IEEE Intl. Conference on Robotics and Automation*, Philadephia, 1988.

[SH85] G. Sahar and J. Hollerbach. Planning of minimum-time trajectories for robot arms. In *IEEE Intl. Conference on Robotics and Automation*, St. Louis, 1985.

[Sha04] M. Sharir. Algorithmic motion planning. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 1037–1064. Chapman and Hall/CRC Press, New York, 2004.

[SI06] M. Saha and P. Isto. Multi-robot motion planning by incremental coordination. In *IROS*, 2006.

[SL03a] G. Sanchez and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. *Robotics Research, STAR 6*, pages 403–407, 2003.

[SL03b] G. Sanchez and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *ISRR*, pages 404–417, 2003.

[SLN00] T. Simeon, J.-P. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6), 2000.

[Smi06] Russell Smith. *Open Dynamics Engine: v05. User Guide*, February 2006.

[SS83a] Jacob T. Schwartz and Micha Sharir. On the piano movers' problem: I. the case of a two-dimensional rigid polygonal body moving admidst polygonal barriers. 36:345–398, 1983.

[SS83b] Jacob T. Schwartz and Micha Sharir. On the piano movers' problem: III. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. 2(3):46–75, 1983.

[SS84] Jacob T. Schwartz and Micha Sharir. On the piano movers' problem: V. the case of a rod moving in three-dimensional space amidst polyhedral obstacles. 37:815–848, 1984.

[SS85] E. Sontag and H. Sussmann. Remarks on the time-optimal control of two-link manipulators. In *Proc. of the* 24^{th} *Conf. on Decision and Control*, Ft. Lauderdale, 1985.

[SSLO98] S. Sekhavat, P. Svestka, J.-P. Laumond, and M. H. Overmars. Multilevel path planning for non-holonomic robots using semi-holonomic subsystems. *IJRR*, 17:840–857, 1998.

[Šve97] P. Švestka. *Robot Motion Planning using Probabilistic Road Maps*. PhD thesis, Utrecht University, the Netherlands, 1997.

[TPK04] H. G. Tanner, G. J. Pappas, and V. Kumar. Leader-to-formation stability. *IEEE TRA*, 20(3), June 2004.

[US03] C. Urmson and R. Simmons. Approaches for Heuristically Biasing RRT Growth. In *Intl. Conf. on Intelligent Robots and Systems*, volume 2, pages 1178–1183, 27-31 Oct. 2003.

[WAS99] Steven A. Wilmarth, Nancy M. Amato, and Peter F. Stiller. Motion planning for a rigid body using random networks on the medial axis of the free space. pages 173–180, 1999.

[Wil88] G. Wilfong. Motion planning for an autonomous vehicle. In *IEEE Intl. Conference on Robotics and Automation*, Philadephia, 1988.

[YH00] M. Yokoo and K. Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):189–212, 2000.

[YJSL05] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle. Dynamic-domain rrts: Efficient exploration by controlling the sampling domain. pages 3856–3861, Barcelona, Spain, 2005.

[YL07] A. Yershova and S. M. LaValle. Improving motion planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics*, 23(1):151–157, February 2007.

[YLVZ04] X. Yang, L. Liu, N. H. Vaidya, and F. Zhao. A vehicle-to-vehicle communication protocol for cooperative collision warning. In *MOBIQUITOUS-04*, 22-26 Aug. 2004.

[ZKB07] M. Zucker, J. Kuffner, and M. Branicky. Multipartite rrts for rapid replanning in dynamic environments. In *IEEE Int. Conf. on Robotics and Automation, ICRA-07*, 2007.