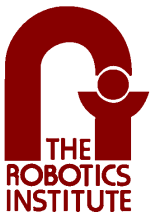
Differentially Constrained Motion Planning with State Lattice Motion Primitives

Mihail N. Pivtoraiko

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Robotics



The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

February 2012

Thesis Committee

Alonzo Kelly, Chair
Matt Mason
Tony Stentz
Steve LaValle, University of Illinois

II

ABSTRACT

Robot motion planning with differential constraints has received a great deal of attention in the last few decades, yet it still remains a challenging problem. Among a number of reasons, three stand out. First, the differential constraints that most physical robots exhibit render the coupling between the control and state spaces quite complicated. Second, it is commonly accepted that robots must be able to operate in environments that are partially or entirely unknown; classical motion planning techniques that assume known structure of the world frequently encounter difficulties when applied in this setting. Third, such robots are typically expected to operate with speed that is commensurate with that of humans. This poses stringent limitations on available runtime and often hard real-time requirements on the motion planner. The impressive advances in computing capacity in recent years have been unable, by themselves, to meet the computational challenge of this problem. New algorithmic approaches to tackle its difficulties continue to be developed to this day.

The approach advocated in this thesis is based on encapsulating some of the complexity of satisfying the differential constraints in pre-computed controls that serve as *motion primitives*, elementary motions that are combined to form the solution trajectory for the system. The contribution of this work is in developing a *general approach* to constructing such motion primitives, given a model of robot mobility. Moreover, the approach allows an unprecedented amount of *pre-computation* in this domain, which yields a dramatic increase in planning efficiency even for systems with complex kinematics and dynamics. Lastly, the proposed motion primitives are fully compatible with a *wide range* of planning algorithms and allow such useful techniques as incre-

mental planning and multi-directional search to be used in the context of planning with differential constraints.

These ideas are demonstrated and validated on a number of relevant systems, both in simulation and in real experiments. This work promises to enable capable and reliable motion planners with differential constraints, as encountered in many realistic robot systems with practical utility, operating efficiently in cluttered, partially known environments.

ACKNOWLEDGEMENTS

I would like to thank very many colleagues and friends for my memories and growth during graduate school years. I would like to thank my advisor, Al Kelly, who gave me the tools, guidance, and opportunities to pursue research topics that always fascinated me. I would also like to thank Tony Stentz, Matt Mason, and Steve LaValle for their thoughtful advice and diligent service as members of my thesis committee.

The Robotics Institute is a unique place because of its people. I would like to thank fellow classmates Ross Knepper, Tom Howard, Colin Green, Dean Anderson, Martin Stolle, Mike Stilman, Peter Barnum, Dmitry Berenson, Jonathan Huang, Lillian Chang, Boris Sofman, Gil Jones, Maxim Makatchev, Young-Woo Seo, Kiho Kwak, David Bradley, David Ferguson, David Silver, David Thompson and so many others for their support and friendship over the years. The RI staff, particularly Cindy Glick, Sumitra Gopal, Jean Harpley, Deb Harvard, Suzanne Lyons Muth, Pamela Sellitti, Cheryl Wehrer, among others helped in so many ways to make my years of graduate school as enjoyable as they have been.

I would also like to thank many colleagues with whom I was fortunate to work during these years. Mark Maimone, Larry Matthies, Issa Nesnas, Adrian Stoica, Gabriel Udomkesmalee, Rich Volpe, Brian Wilcox and others from the Jet Propulsion Laboratory introduced me to space robotics. Mike Bode, Herman Herman, Tom Pilarski, Pete Rander, Randy Warner and so many other co-workers on PerceptOR and LAGR projects, as well as Drew Bagnell, Lei Cui, Tom Galluzzo, Dov Katz, Moslem Kazemi, Tommy Liu, Jean-Sebastien Valois and others on the ARM-S

team have been greatly supportive during our collaboration in some truly exciting work involving building and testing real autonomous robots.

I am particularly thankful to my parents, Nick and Zina, and my little sister Violetta, who have given me an unquantifiable amount of support and encouragement.

Portions of this work were made possible through a number of grants. Outdoor robot navigation work was part of several DARPA programs, including PerceptOR and LAGR. Regional motion planning research was conducted at the Robotics Institute of Carnegie Mellon University under contract to NASA/JPL as part of the Mars Technology Program. The work in rover navigation has been also supported by NASA/JPL under the Strategic University Research Partnership (SURP) program. Control set generation work was also supported by the NASA Graduate Student Researchers Program Fellowship. The results of this thesis were also influenced by the work supported by DARPA ARM-S program. The views and conclusions contained in this document are mine and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Motion planning with differential constraints is recognized as a challenging problem in robotics. The search space dimensionality, system dynamics, uncertainty and partial knowledge about the environment further contribute to the difficulty of this problem. Solving it in *real time* – i.e. on a time scale commensurate with human motion – is even more challenging. In addition, on-board computation of mobile robots is less powerful than that of static, immobile systems. Yet a wide variety of real robots, from wheeled systems to air and water vehicles, rely on this type of motion planning to move efficiently enough to be useful in their applications. Despite several decades of active work in this field, it is still an open area of research.

The approach taken in this thesis is to reduce the computation complexity of on-line, on-board motion planning with differential constraints by performing as much off-line pre-computation as possible. System reachability is analyzed *a priori*, and the constraints are encapsulated in *motion primitives*, pre-computed trajectories that are known to be feasible motions. While it is not

tractable to pre-compute all possible trajectories for the system, this thesis attempts to generate representative sets of primitives that capture the maneuverability of the system as well as possible given a model of robot's mobility. Motion planning is then formulated as searching a graph that includes these motion primitives as atomic state transitions. In this manner, motion planning with differential constraints is reduced to fast unconstrained search in the space that encapsulates the constraints and is generated via significant, but off-line, pre-computation. The objective of this thesis is the design of motion primitives that lead to efficient planning in this setting.

While motion primitives have been advocated in motion planning in the past [44; 99], this thesis provides an innovative solution in several respects. First, the structure of the primitives proposed herein makes them compatible with an unprecedented variety of efficient search techniques, including incremental and multi-directional search – techniques that constrained planning of this type is rarely able to benefit from. This is important for minimizing computational requirements, as well as for enabling the planner to react to the inevitable and frequent changes in the environment representation that are commonplace in robotics.

Second, the approach leads to a greater generality of application and ease of development of motion primitives. Unlike many preceding approaches, the proposed motion primitives are computed *automatically*, given a model of the system mobility. In fact, they can even be modified by the robot itself, should there be a significant change in the system, e.g. due to a mechanical failure.

This thesis attempts to demonstrate that the proposed motion primitives lead to new gains in motion planning by reducing computational requirements and enabling real-time motion planning with differential constraints with the on-board computing capacity of autonomous robot systems, operating in real environments and facing the multitude of challenges that are inevitable in this setting.

1.1 MOTIVATION

Motion planning with differential constraints encompasses nonholonomic and kinodynamic planning and is recognized as an important and hard problem in robotics [22]. A great variety of approaches is being proposed to this day ([11; 12; 29; 45; 58; 76; 81; 87; 92; 106] among others). Differential constraints are important to consider in motion planning because many relevant types

Figure 1.1.1: **Several examples of autonomous robots**. Some of the project and robot names for which I had the privilege of developing motion planners, from top left to right: PerceptOR, FIDO Continuous Navigation, ARM-S, LAGR, UPI, Single-Cycle Instrument Placement on Rocky8.

of robotics platforms impose them. Systems such as wheeled car-like vehicles and tractor-trailer combinations are classical examples of such kinematics constraints [97; 98; 127]. In addition, most physical robots moving at speed are subject to dynamics effects due to inertia, e.g. drift, which also results in velocity constraints. Humanoid walking robots [95], needle steering [4; 57] and other systems have been noted to exhibit constraints of this type as well.

Even though differential constraints are commonplace in robotics, their degree of severity varies. Many applications attempt to ignore such constraints because that significantly simplifies motion planning. It is often done by relaxing the requirements on the robot (reducing speed, simplifying robot's environment), by pushing the complexity to the robot's hardware (e.g. omnidirectional wheels, etc.) or by attempting to build motion controllers that would eliminate the disturbances caused by ignoring such constraints at planning time (although, in principle, this is as difficult as the original problem).

While in some applications these strategies are reasonable ones, the resulting performance losses are too great in a variety of relevant scenarios. The examples of systems that cannot afford

to ignore their mobility constraints include mobile robots with large minimum turning radius, such as autonomous construction and mining equipment (at the time of this writing, being developed by research [34] and commercial communities, e.g. Caterpillar®, Volvo®, etc.), tractor-trailer systems [97] and indoor robots pushing carts [124]. In addition, there are a variety of systems exhibiting significant dynamics effects, such as vehicles moving at high speed [26], flying robots [38], autonomous boats [56], etc. Ignoring the differential constraints of such systems will lead to substantially lower productivity than what these platforms are capable of, due to overly conservative operation. In the worst case, a complete failure, perhaps resulting in collision and damage to the vehicle, will occur as the robot controller becomes unable to execute a trajectory that was planned without regard to the system's differential constraints.

Besides its relevance in applications, motion planning with differential constraints is a very interesting and difficult theoretical problem. There are at least four specific considerations that make it especially challenging in the context of robotics. First, in its general formulation, this problem is recognized to be *NP*-hard [22]. Exact solutions only exist for point masses with L_∞ bounds on velocity and acceleration in one-dimension [102] and two-dimensions [23]. Second, robots typically must be able to operate with speeds commensurate with those of human operators. This poses stringent limitations on available runtime, and often hard real-time computation requirements, a related topic in computer science research. Third, the relationship between the control and state spaces of the robot under differential constraints is quite complicated. In some cases, even simulating the system at the desired fidelity level becomes prohibitively expensive [136]. Fourth, robot's environments are typically partially or entirely unknown. Moreover, models of these environments change frequently and continuously due to modeling artifacts that are difficult or impossible to eliminate, such as sensor noise [64]. In this setting, many classical motion planning techniques that assume known and static structure of the world face significant challenges.

1.2 PROBLEM STATEMENT

Now that the problem of motion planning with differential constraints has been motivated, it is defined here in concrete terms. First, relevant nomenclature is introduced in the following section, followed by an identification, in Section 1.2.2, of four specific aspects of the problem that make it a challenging one and that will guide the development of the presented solution. Finally, Section

1.2.3 establishes the scope of the inquiry by identifying other related, though separate, problems.

1.2.1 TERMINOLOGY

This section describes several concepts that are central to this thesis and defines a number of related terms and symbols. Specifically, it defines the notions of robot state, control, trajectories, paths and their relation to the problem of motion planning with differential constraints, which is also rigorously formulated here.

STATE SPACE

The robot *state space*, X, is a representation of a robot's location in the world, its configuration, velocity and other aspects of its condition, for example: position (x, y, z) and orientation (ϕ, θ, ψ), linear and angular velocities $(v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)$, configurations of joints $(j_0 \dots j_n)$ such as steering angle of a car-like vehicle, etc.

Special symbols denoting certain elements of state space include the *initial state*, $x_I \in X$, and *final state*, $x_F \in X$ of the vehicle. These are also known as *boundary conditions*, as the goal of motion planning is to find a motion that steers the system from x_I to x_F. Some intermediate state along the trajectory is denoted as $x_i \in X$.

In many applications, a vehicle must avoid certain states. This may be due to obstacles in the environment that would make a collision dangerous for the robot, e.g. mechanically. There may also be states of *inevitable collision* that, while not being in collision, would invariably lead the vehicle to collide, e.g. due to dynamics. There may also be arbitrary, user-defined, *no-go* regions, e.g. areas of loose soil where the vehicle would be likely to get stuck. We denote by $X_{\text{obs}} \subset X$ a set of all such forbidden states. We also define $X_{\text{free}} = X \setminus X_{\text{obs}}$, a set of states that are free from such limitations and are allowed for the vehicle to be in.

CONTROL SPACE

Robots are physical devices that react to the influence of excitation signals, such as mechanical, electrical or other types. A set of such signals, or *inputs*, comprises the *control space*, $U \subset \mathbb{R}^m$, of the robot.

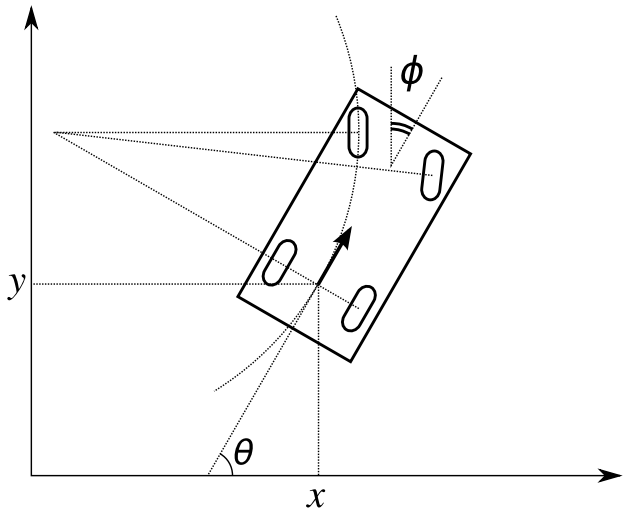


Figure 1.2.1: **Car-like system model**. The figure shows a simple system model that represents the kinematics of a typical four-wheeled vehicle with car-like mobility. The variables x and y are the 2D coordinates of a reference point of the vehicle with respect to a Cartesian frame, and θ is the angle between the positive x-axis and the main axis of the vehicle; ϕ is the steering angle.

The set of inputs depends on the platform and implementation. A wheeled mobile robot can be controlled directly in the input space of wheel torques or at a higher level, e.g. body-frame linear and angular velocities, where wheel torques are computed by a controller.

SYSTEM MODEL

A robot is modeled as a *system*, a set of interacting components that reacts in certain ways to the control signals applied to it. In this thesis, this system is represented by a time-invariant differential equation:

$$\dot{x} = f(x,u) \tag{1.2.1}$$

where $x \in X$, $u \in U$, and $\dot{x} = \frac{dx}{dt}$ is the time derivative. This equation is referred to as a *system model*, also known as a *state transition equation*. It encapsulates the differential constraints of the robot. One basic example of a transition equation is given below.

Example 1.2.1 (Car-like Transition Equation)**.** *A basic model of a robot with car-like mobility is shown in Figure 1.2.1, adapted from [128]. State space of the car is three-dimensional:*

$$[x, y, \theta]^T \in X = \mathbb{R}^2 \times S^1$$

where x and y are the 2D coordinates of a reference point of the vehicle with respect to a Cartesian frame, and θ *is the angle between the positive x-axis and the robot's main axis. The control space is two-dimensional:*

$$[u_1, u_2]^T \in U = \mathbb{R}^2$$

where u_1 *is linear velocity of the car, and* u_2 *is its steering angle.*

The model features two types of differential constraints. One expresses rolling of the wheels without slipping which causes the vehicle to move tangentially to its main axis. This non-integrable equality *constraint is expressed by the following equation*

$$\dot{y}\cos\theta - \dot{x}\sin\theta = 0$$

In addition, there is a non-integrable inequality *constraint that represents a mechanical limit on the steering angle of the car:*

$$\frac{L}{R} \geq |\tan(u_2)|$$

where L is the wheelbase and R is the minimum turning radius of the car.

With these definitions, the transition equation (1.2.1) for the car-like robot takes the form:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ \frac{L}{R} \end{bmatrix} u_2 \tag{1.2.2}$$

□

CONTROL

A *control* is a continuous mapping from time to control space, $u : \mathbb{R} \to U$; in other words, the trajectory specifies a control signal to apply to the system at a given point in time. The domain of this function is a closed interval $[t_I, t_F] \in \mathbb{R}$, where t_I is starting time, and t_F is final time of the trajectory. Note that the symbol u, when used as a vector, still denotes a value of input, an element of U, whereas when written as a function, $u(t)$, it denotes a control.

Time is the most typical trajectory parametrization because time is monotonically increasing, continuous and always defined. Other parameterizations, such as distance traveled, can also be utilized. Without loss of generality, we can also re-parameterize a trajectory such that its domain is an interval $[0, 1] \in \mathbb{R}$.

For convenience, we define a function space of all controls, $\mathcal{U}$, that can be input into the system for execution.

TRAJECTORY

A *trajectory*, $\tau_u : X \times \mathbb{R} \to X$ is a projection of a control on state space under the state transition equation (1.2.1) and is obtained by integration:

$$\tau_u(x_I, t) = \int_{t_I}^{t} f(x(t'), u(t'))dt' + x_I \tag{1.2.3}$$

Equation 1.2.3 cannot be integrated in closed form for most non-trivial robot systems. Numerical methods such as Euler or multi-step Runge-Kutta must be applied instead. Balancing the fidelity of this integration with runtime performance is one of the challenges of motion planning. If the required simulation fidelity is high, this model integration, also referred to as *simulation*, may slow down planning significantly. The projection of the trajectory on the position subspace of X, e.g. coordinates $[x, y]^T$ in Example 1.2.1 , is referred to as a *path*.

A *cusp* is a point in a path where linear velocity changes sign. While cusps are discontinuous in path curvature, they are not discontinuous in control space and are feasible motions. Similarly, a turn-in-place is a path point where the heading changes discontinuously, however a turn-in-place is continuous in the control space trajectory representation.

For convenience, we define a function space of all feasible trajectories $\mathcal{T}$ that result from

applying equation (1.2.3) to every element of $\mathcal{U}$.

BOUNDARY VALUE PROBLEM

Boundary value problem (BVP) is a problem of computing a control $u(t) \in \mathcal{U}$ such that the resulting trajectory satisfies boundary equality constraints and the state transition equation:

$$\dot{x} = f(x,u) \tag{1.2.4}$$
$$\tau_u(x_I,t_I) = x_I \tag{1.2.5}$$
$$\tau_u(x_I,t_F) = x_F \tag{1.2.6}$$

However, notice that $u(t)$ may not be unique. We can also define *optimal* BVP as the problem of computing a trajectory that minimizes a certain time-independent *cost function* $c : X \times U \to \mathbb{R}$ that assigns cost of giving the system an input $u \in U$ while it is in state $x \in X$. In practice, cost may be related to distance traveled by the vehicle, energy consumed, risk experienced, etc. The cost function is typically designed such that the robot's perceptual information is mapped into a scalar value that is consistent with the desired notion of motion cost.

Optimal BVP is then formulated as:

$$\begin{aligned} u^* = \quad & \underset{u\in\mathcal{U}}{\operatorname{argmin}} \int_{t_I}^{t_F} c(\tau_u(x_I,t),u(t))dt \\ \text{s.t.} \quad & \dot{x} = f(x,u) \\ & \tau_u(x_I,t_I) = x_I \\ & \tau_u(x_I,t_F) = x_F \end{aligned} \tag{1.2.7}$$

The domain of minimization above, $\mathcal{U}$, is known as a *search space* – a space of motion alternatives that is evaluated while the optimal motion is sought.

For several special classes of systems, such solutions are available in closed form [30; 115]. In general, however, numerical techniques must be utilized. We observe that system simulation is an important component of this problem because it is required to obtain $\tau_u(t)$ as part of evaluating the cost integral in (1.2.7).

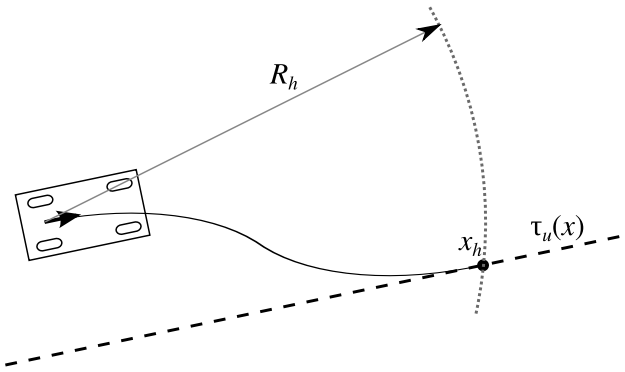


Figure 1.2.2: **Controller path following**. A receding horizon controller attempts to steer the system to remain on the reference path $\tau_u(x)$ (dashed line) by generating an appropriate *correction control* (solid line) to eliminate the cross-track error. The figure shows a popular approach that is similar to pure-pursuit: a reference trajectory sample x_i is chosen at the L_2 horizon, or *look-ahead*, radius R_h. In this setting, computing the correction control is identical to BVP.

CONTROLLER

As argued in Section 1.1, most real problems in robotics face challenges of uncertainty and unmodelled perturbations both in the robot's environment and in the robot's transition equation. Ignoring these effects may cause undesirable interaction with the environment, e.g. collision potentially resulting in damage, or an error in integrating the transition equation, which may grow very large over the course of executing a trajectory.

In order to mitigate such effects, it is common practice to equip robot's actuators with a module that monitors the evolution of the system with respect to a path that it is attempting to follow during motion, referred to as a *reference trajectory*, and issues corrective action in the event that the system deviates from this trajectory. This module, a *controller*, can be expressed as a process that implements a trajectory-following policy by generating a control based on error dynamics during trajectory following.

Figure 1.2.2 illustrates that designing a receding horizon controller is similar to solving the BVP: if the controller is built to utilize certain look-ahead along the reference trajectory, then its operation is similar to solving the BVP to steer the system to a point on the reference trajectory that is within the given time horizon. Based on the close relationship between the two concepts, the term controller will be generalized to include solving the BVP in the sequel.

Example 1.2.2 (Motivating feasible planning)**.** *The car-like robot, just like the one in Example 1.2.1 attempts to travel to a goal on the other side of a collection of obstacles that forms a narrow corridor (Figure 1.2.3). Suppose the motion planner is based on searching a grid, and thus is produces a trajectory through the corridor (black line). However, this path is infeasible, since the car-like robot cannot turn in place. As the vehicle moves and the controller attempts to guide the vehicle through the corridor, it is not able to do so. The only viable alternative is to back up and replan, thereby bringing the corridor out of the scope of the controller and back into the scope of the (infeasible) planner, and the process repeats. Without special behavior monitoring, the vehicle will cycle back and forth indefinitely.* □

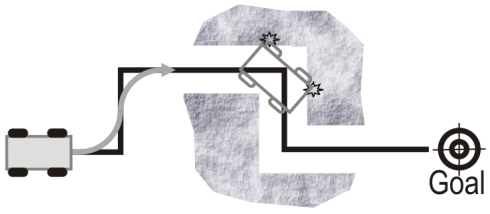


Figure 1.2.3: **Motivating feasible planning**. An example of a motion planning problem, where following the infeasible planned trajectory (black line) causes a failure. A controller is unable to "fix" the trajectory to enable the car-like robot to follow the path. Making this correction is as difficult as the original motion planning problem.

MOTION PLANNING

Notice that the BVP considers differential constraints of the system (via the state transition equation) and the boundary state constraints, but it does not consider environment constraints, such as collision avoidance. We formulate the *motion planning* problem as that of satisfying the boundary constraints while minimizing a cost function and assuring that the motion is collision free.

$$
\begin{aligned}
u^* = \quad & \underset{u \in \mathcal{U}}{\operatorname{argmin}} \int_{t_I}^{t_F} c(\tau_u(t), u(t)) dt \\
\text{s.t.} \quad & \dot{x} = f(x,u) \\
& \tau_u(x_I, t_I) = x_I \\
& \tau_u(x_I, t_F) = x_F \\
& \tau_u(x_I, t) \in X_{\text{free}}, \forall t \in [t_I, t_F]
\end{aligned}
\tag{1.2.8}
$$

In most real robotics applications, the model of the environment, including X_{free}, is partially unknown and uncertain. Therefore, the robot may not have all the information necessary to compute an optimal τ^*. The robot is only able to compute *information-optimal* solution, based on the information that it does have. This circumstance is frequently omitted in motion planning literature, although it is an extremely important one in robotics. As the robot's knowledge of the environment improves, the robot may need to *replan*, i.e. modify its motion plan per new information [129]. Methods that are able to improve the efficiency of replanning, e.g. due to pre-computation or caching previous on-line computation, are especially promising in this domain – which has been one of the guiding principles in this work.

ROADMAP

While the motion planning problem with differential constraints, as defined above, remains an open research question, a variety of approximate solutions have been proposed over the last several decades. The approach proposed in this thesis is most similar to a class of methods referred to as *roadmaps* [5; 61]. Such methods perform sampling in X in order to build up a directed, cyclic graph $\mathcal{G} = \mathcal{V} \cup \mathcal{E}$, where $\mathcal{V} \in X$ is a set of *vertices* and $\mathcal{E} \in \mathcal{T}$ is a set of trajectories between vertices, *edges*. The cyclic property of these graphs distinguishes them from the approaches based on trees [55; 87]. We will also use the term *lazy* roadmap, referring to the practice of constructing the roadmap without regard for obstacles and deferring collision detection until the graph is searched [19]. *Quasi-random* lazy roadmaps arrange vertices $\mathcal{V}$ with specific structure, e.g. a lattice, such that the roadmap need not be constructed explicitly before search [21].

The roadmap can be interpreted as a discrete set of states that are reachable by the system. The edges of the graph are weighted in such a way as to represent the costs of traversing them and are

based on the values of the cost function. The state values of successor vertices of any vertex are obtained by computing (simulating) the path originating at the vertex using the control trajectory associated with each corresponding edge.

A set of successor edges from a vertex is referred to as a *control set*. In general, each vertex has its own specific control set. However, when certain structure is imposed in the search space, a control set may be shared by some of the vertices. The cardinality of a control set of a vertex is its *outdegree*. The mean of outdegree values throughout the search space is *average outdegree*.

SAMPLING CONCEPTS

While most real-world systems are continuous, many hard problems like motion planning with differential constraints don't yet have exact solutions and must rely on approximation to be able to produce a solution within a reasonable amount of time. Often, there is a trade-off between the quality of the approximation and the effort it takes to compute the solution. Discretizing the continuum into a number of samples is one way to manage this trade-off.

Sampling is ubiquitous in many areas of computation, besides motion planning. Most planning algorithms operate by sampling the search space of the problem, e.g. space of motions, and by evaluating the samples while looking for the motion that optimizes (1.2.8). Two basic types of sampling are *randomized* and *deterministic*. The former relies on a random number generator to pick samples. The latter utilizes a certain systematic principle in making sample choices.

A large body of work has been dedicated to measuring and improving the quality of sampling. Some popular measures of sampling quality include *discrepancy* [101] and *dispersion* [132]. One of the benefits of deterministic sampling, such as lattice methods [125], is the relative ease of obtaining good sampling quality. One potential drawback, however, is lack of flexibility, without special considerations, to vary the degree of sampling in some parts of search space, e.g. especially interesting ones.

The forward and inverse state *sampling rule* $s : \mathbb{R}^m \to \mathbb{N}^m$ is the mapping between continuous and discrete state values. It transforms continuum vehicle states $x \in X$ into discretized values, e.g. vertices in the roadmap, $\hat{x} \in \mathcal{V}$. The semantics of roadmap edges can be related to sampling as well: the set of edges $\mathcal{E}$ can be thought of as a sampled subset of $\mathcal{T}$. In the sequel, it will be helpful to use the sampled version of state $\hat{X}$ and trajectory $\hat{\mathcal{T}}$ spaces; the symbols refer to the potentially

countably infinite subsets of the respective space:

$$\hat{X} = s(x), \forall x \in X$$

where one can observe that elements of $\hat{X}$ correspond to entire subsets of X, known as *cells*. The value $\hat{x}$ then serves as a "representative" value of the cell. E.g., if a cell is a certain ball in state space, $\hat{x}$ may be the center of that ball.

A sampled version of the cost function is also widely used in robotics and is known as a *cost map*. It is a collection of state space cells for which the cost function is evaluated given the representative state value of the cell. Evaluation of the cost of the trajectory then is reduced to a Riemann summation operation over a set of state cells that are occupied by the trajectory being evaluated. The set of such cells is referred to as a *swath* of a trajectory.

1.2.2 CHALLENGES

This thesis identifies four specific aspects of the problem of motion planning with differential constraints that account for its complexity. These challenges are *Feasibility*, *Optimality*, *Runtime* and *Completeness*. They are further described below. This thesis attempts to develop search spaces that addresses all four of these challenges. Specific measures are designed (in Chapter 5) to evaluate the degree to which these challenges are met.

Typically trade-offs exist between all of the criteria in any design problem. The optimal design problem is that of finding the best design given some, potentially application-dependent, weighted measures of them all. For the sake of generality, no optimal design will be attempted here, because the basis for such optimality would require assuming a particular application. However, the proposed design rules will at least be somewhat principled.

FEASIBILITY

In this context, *feasibility* refers to the challenge of solving the motion planning problem for a control $u(t)$ that satisfies the state transition equation (1.2.1). As mobile robots continue to be applied in increasingly difficult environments, travel at higher speeds, and perform more complex tasks, action feasibility of motion planner solutions becomes very important, as motivated in Section 1.1.

One simple counter-example of a search space that violates the feasibility property of a system in Example 1.2.1 is a roadmap with $\mathcal{V}$ sampled as a regular grid in X and with full vertex connectivity. For differentially steered mobile robots (or others which can execute turn-in-place actions), their actions can be modified such that a motion plan from this graph is followable, but it would be necessary for the vehicle to reduce its velocity to zero at curvature discontinuities in order to follow such a trajectory closely. If reference trajectory deviation greatly increases risk of failure, a better strategy would have been to incorporate feasibility into the search space design.

Feasibility is also a challenge in environments where terrain variation or environment effects (e.g. currents, loose soil, etc.) are significant. If the roadmap is designed without taking this variation into account, some edges may no longer satisfy the state transition equation within reasonable bounds, and may have to be ignored as viable motion alternatives. Unlike kinematic and dynamic constraints, feasibility due to terrain effects and environmental effects cannot always be determined in the *a priori* roadmap design.

In the context of physical systems, feasibility is related to continuity of the trajectory. The continuity is a matter of degree, e.g. C^1, C^2, etc. for continuity of the first and second derivatives, respectively. In practice, the degree of continuity is often chosen based on the application.

OPTIMALITY

The property of *optimality* refers to the quality of a motion solution. Specifically, it refers to the minimization in (1.2.8). As suggested in the definition of motion cost, often-used forms of optimality include minimum distance, time, energy, risk, although generally any measure of interest could be used. Two different forms of this concept are *local* and *global* optimality. Global optimality is achieved, if possible, when the minimum-cost solution with respect to $\mathcal{T}$, the set of all feasible motion alternatives that steer the system from x_I to x_F, is determined. Conversely, local optimality occurs when first derivatives of the cost with respect to the parameters that define the motion are zero and second derivatives are positive.

RUNTIME

An important aspect of the problem being considered is *runtime*, the clock time that it takes to compute a solution. Computational efficiency is of key concern to most problems in computer science,

and it is especially acute in robotics. The limits of acceptable runtime in this field are notably lower than in many other domains. A motion planner and any other algorithm that governs the action of a physical entity in motion, must be able to react quickly enough so as to avoid potential damage due to a collision or other undesirable interaction with environment. Many classical approaches to motion planning with differential constraints do not place sufficient emphasis on the algorithm runtime in real terms. One of the discriminators of this thesis is great concern with guarantees on planner runtime, and the aim to validate the proposed approaches on real platforms.

COMPLETENESS

Motion planning *completeness* is the property of a planning algorithm of being able to determine whether an action that satisfies the motion planning problem can be found, or determined not to exist. In planning for nontrivial mobile robot systems (e.g. with differential constraints, especially those that operate in obstacle-ridden environments) is, however, rarely guaranteed. Related concepts include *resolution-completeness*, an ability of the planner to satisfy the completeness property in the limit as the fidelity of its representation is increasingly improved [79], and *probabilistic completeness*, a property of the planner to be able to find a solution with probability of unity in the limit as its search space sampling density continues to increase [87].

1.2.3 SCOPE

This thesis makes several assumptions to define the scope of the research:

- A motion model (1.2.1) is known, although it is assumed to be inaccurate. This work does not consider the problem of discovering the model, e.g. system identification.
- Design of immediate, local reactive controllers is beyond the scope of this work. An attempt is made to avoid intersecting with this separate, yet related field. It is assumed that such a controller is available for the system, and it is capable of solving a two-point boundary value problem for the system, as illustrated in Figure 1.2.2. It is also assumed that benign, unmodelled environment variations (e.g. terrain shape, loose soil, currents, etc.) and other perturbations are eliminated by this controller.

- The controllability properties of the system allow a finite subset of its reachability tree to be an accurate representation of the system's maneuverability. The degree to which this representation is an accurate one will influence the quality of the search spaces being designed with the proposed methods.

- The finite representation of system reachability verifies translational symmetry: the reachability is invariant to the initial position of the robot in its environment. While a simulated robot operating on a flat plane would easily satisfy this assumption, real physical robots typically rely on low-level controllers to eliminate the inevitable perturbations that they experience while traversing their environment. This assumption is typically made in motion planning in order to alleviate the additional computational complexity of explicitly reasoning about these perturbations (both known and discovered during motion).

- Explicit reasoning about the uncertainty in the environment is not a specific requirement for building a functional and effective motion planner. Uncertainty of the robot and its environment is accounted for implicitly, however. The approach is nevertheless validated with substantial experimentation with physical autonomous robots in realistic settings.

1.3 APPROACH

The approach presented in this thesis builds upon a long heritage and many strengths of earlier motion planning techniques. In particular, the method builds upon the previous use of an implicit representation of the search space [21], the research in BVP solver and controller design ([53; 123] among others), incremental deterministic sampling [85], and an off-line analysis of the reachability set of the system for improved on-line performance [43].

In general terms, the crux of the proposal is the generation of a search space – a roadmap, represented as a directed cyclic graph – that satisfies differential constraints by construction. The problem of motion planning is thereby reduced to unconstrained heuristic search in a search space of specific structure and appropriate dimensionality. This setup leads to a number of efficiencies that are critical to robotics applications, especially those facing frequent and unpredictable changes in the environment model, e.g. due to sensor uncertainty or other effects. The proposed structure of motion primitives leads to the ability to generate sets of such motion primitives automatically.

1.4 CONTRIBUTION

The specific contributions of this thesis are as follows:

- A new method for motion planning with differential constraints based on deterministic sampling. The method is demonstrated, in experiments with real robots, to offer unprecedented performance (with respect to the criteria in Section 1.2.2) in robotics applications with frequent and unpredictable changes in the environment model.
- A new type of motion primitives that possess the necessary structure to result in search spaces with cyclic graph topology, which offers a number of advantages over search spaces that are standard in this domain.
- The first method of automatic generation of motion primitives, generally applicable to a large class of systems with differential constraints, based solely on the given system transition model.

1.5 APPLICATIONS

The contributions of this thesis are relevant to any system featuring differential constraints, operating in a complex, unknown operational environment. A number of current mobile robot applications are of particular interest, including field robotics, exploration robotics, agricultural and mining systems, autonomous automobiles, and mobile manipulators.

1.5.1 OFF-ROAD NAVIGATION

The initial motivation for this work came from experiences with car-like mobile robots operating in cluttered natural terrain. Unstructured environments often pose significant challenges to mobile robots. The perception problem is known to be difficult; however, even if detailed, timely, and accurate perception information is available, it is still a difficult problem to fully utilize it in motion planning, especially under real-time constraints. Today's navigation algorithms often make mistakes that result in the vehicle being trapped in a set of dense obstacles with no way to proceed in the forward direction.

During the DARPA Perception for Off-Road Robotics (PerceptOR) project, our vehicle faced such problems several times an hour in challenging terrains. The initial ideas of the approach presented herein were investigated first on this program. An implementation was developed using most of the central concepts of this method: an explicit state lattice based on the mobility model of the PerceptOR vehicles (Figure 1.5.1) was precomputed, and the A* algorithm was utilized to search it [66]. The algorithm was invoked as an aggressive limited scope behavior when simpler alternatives failed to produce a solution.

Figure 1.5.1: **PerceptOR robot**. A preliminary version of our results was verified on PerceptOR vehicles, designed to navigate in natural environments.

Note that the final formulation of the method presented here achieves about two orders of magnitude speed up relative to the one utilized on PerceptOR. Custom, obstacle avoiding n-point turn maneuvers can now be generated in reactive fashion. The extended horizon of the planner would enable the generation of a backup maneuver all the way out of a discovered cul-de-sac, if this maneuver was considered preferable to a complicated n-point turn.

1.5.2 PLANETARY ROVER LOCOMOTION

Motion planning for planetary exploration rovers is in many ways similar to the terrestrial problem discussed above. However, it is also has many distinctive features. One significant difference is that rovers typically move fairly slowly, only several centimeters per second, for a variety of reasons including stringent weight, computational and power limitations. Moreover, the rovers are

typically designed to move in stop and go fashion. GESTALT, the rover navigation algorithm that drives the MER rovers uses higher level perception, such as stereo vision, while it is stationary and selects the best arc path to follow. Afterwards, the rover commits to driving the selected arc without perception, using only lower level sensors such as current sensors [47].

Despite these differences, the the solution presented herein can be applied to the rover locomotion problem in a straight-forward manner. One way to apply it is similar to the terrestrial case, i.e. as a backup behavior that is invoked only when the rover is no longer able to navigate further due to errors in perception, path following, state estimation, etc. In this case the rover would compute a rescuing path and get back on track autonomously. Future adoption of technologies such as this will undoubtedly reduce the effort and cost of manual rover path planning.

The presented planner may also be applied in other ways. For example, given its efficiency, it's reasonable to imagine that navigation algorithms such as GESTALT could be enhanced with lattice planning capability, such that instead of arc selection, they might perform lattice search and produce the complete path: not only an arc, but an "S" shape or anything else as required by the environment. Given the efficiency of the planner, the computation required is promising to be comparable to the conventional method of evaluating the requisite number of arcs and running an arbiter.

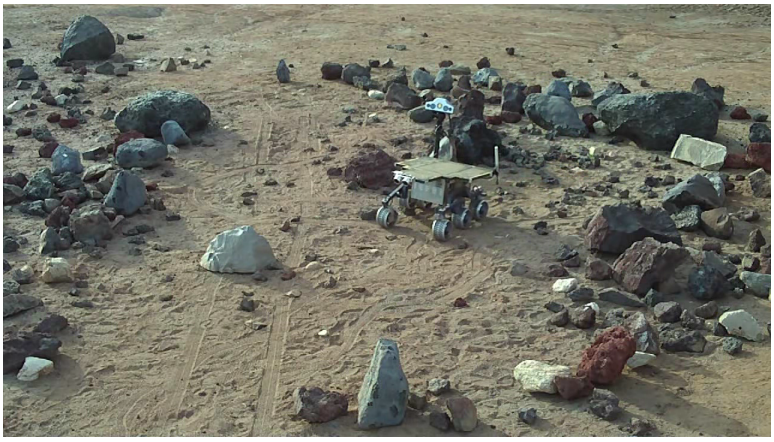


Figure 1.5.2: **FIDO rover**. Rover is moving autonomously under the guidance of the state lattice planner, implemented within the CLARAty software architecture.

As was discussed above, in simple to medium obstacle fields, the lattice planner compares in

efficiency to grid search, primarily because the heuristic we designed accurately guides the search such that backtracking is minimized. The search can be visualized as proceeding in depth-first manner, but leading invariably towards the goal, thanks to the ideal heuristic encoded in the look-up table. In many implementations, obstacle avoidance arc evaluation is performed like breadth-first search to unity depth: many arcs are evaluated, but only is ultimately used. With careful design, the presented planner could save computation and extend the planning horizon to the limits of perception.

An implementation of the state lattice planner has been integrated with the Coupled Layer Architecture for Robotic Autonomy (CLARAty) system at the Jet Propulsion Laboratory. This allowed us to test the application of the planner to rover locomotion through simulation using JPL's high-fidelity rover dynamics simulation system, ROAMS (Rover Modeling and Simulation). Figure 1.5.2 depicts an experiment with the FIDO research prototype rover performing autonomous navigation in rough terrain. The goal is to move toward the goal in a field spotted with large obstacles. An important motivation for this work is to assist in the single-cycle instrument placement task. Currently it takes three to four Martian days for rover drivers to move the rover to the desired science target. By using this planner, we believe the instrument placement task will be simplified by making the navigation efficient computationally, by fully utilizing high-level perception information by computing as long "blind drive" paths as perception allows, and by guaranteeing that the rover's navigator never gets stuck, as unlimited search will always find the path out of cul-de-sacs, whereas typical arc-based navigators typically are not designed for this purpose.

1.5.3 Reactive Motion Planning and Path Modification

By virtue of its efficiency, the presented planner can be useful in a variety of other tasks. In particular, it could be applied to automatic mobile equipment that performs routine and repetitive tasks, e.g. earth movers, harbor transporters, convoy vehicles, etc. It is worth noting that such systems can benefit from more structured environments than unknown natural terrain. It is possible to build a reactive motion planning system that will be able to modify its current path immediately upon perceiving an unexpected obstacle. Moreover, as we have shown in the discussion of admissibility of the heuristic, the new path will be optimal, which is likely to result in minimum deviation from the original course, just enough to negotiate the unexpected obstacle.

1.5.4 MINING AND AGRICULTURAL ROBOTICS

Resource extraction automation will revolutionize the mining and agriculture industries in the next century. Autonomous vehicles will lead to more stable, efficient, and cost effective platform design. Resources could be harvested in dangerous, remote environments including steep mountainside terraces, the ocean floor, asteroids, and the Moon. Mobile robots that better follow natural resource contours makes autonomous systems more profitable. Autonomous agricultural systems that generate minimum-length motion plans, satisfy mobility and environmental constraints, and extract resources at a faster rate can decrease overhead costs by reducing fuel consumption and maintenance. Reasoning about the kinematic constraints of the vehicle is important when the turning radius may be limited, since the vehicle may have to initially turn away from the intended target to achieve the desired state. Likewise, autonomous agricultural and mining systems often repeat similar motion plans in the same environment many times. By progressively adapting the search space to conform exactly to the particular environment, the optimality of the resulting motion plans is likely to increase, resulting in potentially faster, better, and cheaper systems.

1.5.5 AUTONOMOUS AUTOMOBILES

The automation of the streets and highways in modern cities using roboticized automobiles will lead to safer and more efficient transportation systems for humans. To achieve this, developments in perception, localization, mapping, communication, motion planning, and control must occur. Motion plan feasibility is important at the speeds associated with highway navigation. Reasoning about kinematic and dynamic constraints while performing swerves, lane changes, or simple lane following is necessary to maintain vehicle safety. Planning infeasible actions and relying on lower-level path followers and obstacle avoidance algorithms can be disastrous if the difference between the planned and actual actions is significant.

1.6 ORGANIZATION

This dissertation is organized in 9 chapters. First, terminology, concepts, and challenges addressed in this thesis are described in Chapter 1. Related work in mobile robot motion planning, navigation, and control follow this discussion in Chapter 2. In Chapter 3 we embark from our problem

definition and construct the proposed type of motion primitives, followed by a discussion of their features and requirements. The following Chapter 4 is dedicated to the details of exploiting the properties of these primitives for building an efficient motion planners using a variety of search algorithms. Chapter 5 embarks on a discussion of automatic design of state lattice motion primitives. It suggests quality metrics for such search spaces, so that alternative search space designs may be compared in order to select the design of better quality. Chapter 6 provides recommendations for developing state space sampling for improved performance of the resulting planner. Finally, Chapter 7 presents a set of algorithms for the automatic generation of the given type of motion primitives. A thorough account of experimental validation of the key results in this thesis are given in Chapter 8. The thesis concludes with a summary and a review of the contributions in Chapter 9.

1.7 Publication Note

Portions of the presented motion planning research in Chapters 3 and 4 appear in [109; 110]. The related field experiments are discussed in [113]. Extensions of this work for dynamic search spaces appear in [111; 114]. Approaches for automatic generation of motion primitives were given in [112].

CHAPTER 2

RELATED WORK

This thesis is concerned with developing a novel approach to motion planning with differential constraints that lends itself well to application on physical robots operating in difficult, challenging environments. Substantial prior research has advanced mobile robot motion planning and navigation to the point where mobile robots can successfully traverse open deserts, navigate urban environments, and explore the oceans and other bodies in our solar system. This section reviews prior work in mobile robot motion planning, and navigation related to the topics of this thesis.

2.1 VEHICLE CONSTRAINTS

A significant amount of work has been dedicated in recent years to the problem of smooth inverse trajectory generation for nonholonomic vehicles: finding a smooth and feasible path given two end-point configurations. While in general this is a difficult problem, recent progress in this

area produced a variety of fast algorithms. The ground-breaking work in analyzing the paths for nonholonomic vehicles was done by Dubins [30] and Reeds and Shepp [115]. Their ideas were further developed in algorithms proposed by Scheuer and Laugier [123], Fraichard and Ahuactzin [40], and Fraichard and Scheuer [41] where smoothness of paths was achieved by introducing segments of clothoids (curves whose curvature is a linear function of their length) along with arcs and straight line segments. Somewhat different approaches by Scheuer and Fraichard [122], and Lamiraux and Laumond [78], among others, have also been shown to solve the generation problem successfully and quite efficiently. On the other hand, Frazzoli, Dahleh and Feron [44] suggest that there are many cases where efficient, obstacle-free paths may be computed analytically, e.g. the systems with linear dynamics and a quadratic cost (double or triple integrators with control amplitude and rate limits). The cases that do not admit closed-form solutions can be approached numerically by solving appropriate optimal control problems (e.g. [37], [6]). A fast nonholonomic trajectory generator by Kelly and Nagy [63], Howard and Kelly [52] generates polynomial spiral trajectories using parametric optimal control.

2.2 Environment Constraints

Some of the methods described above also proposed applications to planning among obstacles. Since the beginning of modern motion planning research ([94], [93], [116]), there has been interest in the planning methods that constructed boundary representations of configuration space obstacles ([23], [2], [1] and others). The complexity of motion planning algorithms has also been studied ([24], [100], [3], [59]). With the advent of efficient C-space sampling methods ([10], [48]), there has been interest in algorithms that sample the space in deterministic fashion ([11], [9], others in [79]). Lacaze et al. [77] utilized these ideas to propose a method for planning over rough terrain using generation of motion primitives by integrating the forward model. Cherif [27] advanced these concepts by basing planning on physical modeling. Note that one of principal differences (and a novelty, to our knowledge) of our approach is leveraging the research in inverse trajectory generators to generate motion primitives under convenient discretization. Smoothing with primitives was also evaluated [39].

2.3 RANDOMIZED MOTION PLANNING

Also in the early 1990s, randomized sampling was introduced to motion planning ([8], among others). The Probabilistic Roadmap (PRM) methods were shown to be well-suited for path planning in C-spaces with many degrees of freedom ([62], [61], [54]), and with complex constraints, e.g. nonholonomic, kinodynamic, etc. ([74], [54], [25], [68]). Another type of probabilistic planning was Rapidly-exploring Random Trees (RRT) introduced by LaValle and Kuffner ([82],[83]). RRTs were originally developed for handling differential constraints, although they have also been widely applied to the Piano Mover's problem ([86]). Randomized approaches are understood to be incomplete, strictly speaking, but capable of solving many challenging problems quite efficiently ([21]).

As the randomized planners became increasingly well understood in recent years, it was suggested that their efficiency was not due to randomization itself. LaValle, Branicky and Lindemann [85] suggest an intuition that real random number generators always have a degree of determinism. In fact, Branicky et al. [21] show that quasi-random sampling sequences can accomplish similar or better performance than their random counterparts. The improvements in performance are primarily attributed to the more uniform sampling of quasi-random methods. For these reasons, Branicky et al. [21] introduced Quasi-PRM and Lattice Roadmap (LRM) algorithms that used low-discrepancy Halton/Hammersley sequences and a regular lattice, respectively, for sampling. Both methods were shown to be resolution-complete, while the LRM appeared especially attractive due to its properties of optimal dispersion and near-optimal discrepancy. In this light, our approach of sampling on a regular lattice can be considered to be one of building on the LRM idea and making it more efficient through a detailed analysis of a fit between the reachability graph of the system and the underlying sampling lattice.

2.4 IMPLICIT SAMPLING

Recent works have also discussed "lazy" variants of the above planning methods that avoid collision checking during the roadmap construction phase (e.g. [19], [17], [21], [120], [121]). In this manner the same roadmap could be used in a variety of settings, at the cost of performing collision checking during the search. An even "lazier" version is suggested, in which "the initial graph

is not even explicitly represented" [21]. In this regard, our approach of using an implicit lattice and searching it by means of a pre-computed control set that only captures local connectivity is similar to the Lazy LRM. Our contribution is in exploring the conjecture made in that work and successfully applying it to nonholonomic motion planning.

2.5 SAMPLING

The question of lattice sampling has also been raised in the context of motion planning utilizing control quantization. Bicchi, Marigo and Piccoli [16] as well as Pancanti, Pallottino and Bicchi [106] showed that, for systems that can be expressed as $\ddot{q} = g(q,u)$ that are not under-actuated, through careful discretization in control space, it is possible to force the resulting reachability graph of the system to be a lattice. It was also shown that this technique can be applied to a large class of nonholonomic systems. That approach presents a way to generate a path given its terminal points and shows how under suitable conditions a regular lattice of reachable points can be achieved. However, this is usually difficult to achieve, and under most quantizations the vertices of the reachability graph are unfortunately dense in the reachable set [86]. By contrast, our method uses an inverse path generator capable of generating essentially any feasible motion, so we can choose a convenient discretization of state space based on the problem demands and generate a custom set of controls to suit this discretization.

In this work we also build on a considerable amount of research in analysis of system reachability. One line of research uses planning methods themselves to analyze reachability of complex systems [15]. Since we focus on applications of motion planning of wheeled vehicles, we have developed a technique based on exhaustive search that has worked quite well for exploration of the reachability graph. Our method involves pruning this search early to analyze reachability efficiently.

2.6 MOTION PRIMITIVES

There has been significant interest recently in developing *motion primitives*, specially designed controls that facilitate motion planning, particularly under kinematics and dynamics constraints on robot motion. The controls are developed during construction of the planner (*pre-computed*) and

represent *feasible* motions, i.e. those that satisfy the constraints of the system. Motion planners can utilize these primitives to enact efficient search in state space by ignoring system constraints, instead focusing on the environment and other constraints – thereby improving efficiency of the planning. The role of primitives in planning and the importance of their quality have been motivated both in deterministic [32; 46; 90] and randomized [44] planning domains. Their importance was also noted in the related area of reactive obstacle avoidance in the context of mobile robot navigation [20; 49; 80]. A number of popular approaches to kinematic and kinodynamic planning can readily incorporate primitives in their design. The requisite *local planner* in [55; 61; 87] can be implemented as a process that chooses an appropriate element from a set of primitives [44]. In deterministic approaches [11; 29; 34; 46; 60; 90], the vertex expansion (set of edges emanating from a vertex) can be a pre-determined set of primitives based on the state value that the vertex represents. Dynamic path modification was done in [67].

Motion primitives have been designed in the past through sampling control space in such a way as to result in good sampling in state space in terms of discrepancy, dispersion or path diversity [20; 32; 46; 49; 80; 107]. We refer to this line of work as *control-sampling* primitives. In general, designing such primitives is difficult due to the complexity of the relationship between the robot's control and state spaces under kinematics and dynamics constraints. Motivated by this, we propose *state lattice* primitives, designed via a reverse process. First, we establish an attractive sampling rule in state space, perhaps one that is convenient and efficient for the planning problem (e.g. commensurate with the robot's world model, such as an occupancy grid). Then, we compute the controls that steer the system between these samples using a boundary value problem (BVP) solver. The approach can be viewed as a way of extending the Lazy LRM [88] to handle kinematics and dynamics constraints by leveraging the related research in BVP. Such solvers are available for a variety of systems, such as car-like [115], chained form [98; 107], as well as in rough terrain [53] and dynamics [72; 131] settings.

CHAPTER 3

MOTION PRIMITIVES

This section presents a progression of design principles that results in the creation of a search space that is the topic of this work and that was introduced in Section 1.3.

3.1 Regular Lattices and Grids

In problems where analytic representations are not convenient, a useful step in problem formulation is to establish a sampling policy in order to avoid attempting to search the entire continuum. As suggested in Section 1.2, beneficial sampling policies include those that cover a larger volume of the state space with fewer samples. It is natural to extend this concept from values of state space to paths to functions, elements of $\mathcal{T}$. Similar to the state space, the continuum of motions that are executable by the system can also be sampled to make computation tractable. This thesis attempts to develop a sampling of motions that is both convenient to design and effective for motion

planning.

Although there are alternatives of a probabilistic nature, this work explores search structure afforded by deterministic sampling mechanisms which produce regular lattices of states. A principal advantage of regular sampling is translational invariance. Any control that joins two different states will also join all other pairs of identically arranged states. Through an application of the same argument to all paths joining any vertex of a roadmap to its neighbors, it becomes clear that the same set of controls emanating in a repeating pattern from a given state can be applied at every other instance of the repeating unit. Therefore, in this case of a regular lattice, the information encoding the connectivity of the search space (ignoring obstacles) can be precomputed, and it can be stored compactly in terms of a canonical set of repeated primitive motions, the control set of the lattice.

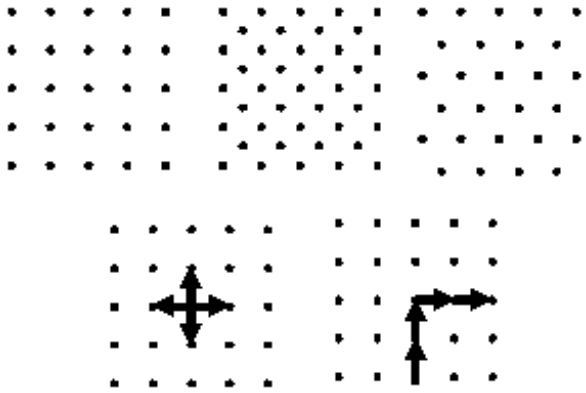


Figure 3.1.1: **Regular Lattices**. Top: rectangular, diamond, and triangular (hexagonal) lattices. Bottom Left: Controls for a 4-connected lattice. Bottom Right: Discontinuous heading change.

The simplest case of a regular lattice is a rectangular grid (Figure 3.1.1). In this case, it is common to use a control set consisting of North, South, East, and West motions to create a "4-connected" graph. While this is straightforward, the transitions between controls that occur at the vertices can be problematic in practice due to discontinuity in some of the state dimensions. Consider, for example, a path which enters a vertex from the south and then leaves toward the east. When heading changes across a vertex in a generated path, its execution at any nonzero linear velocity implies an instantaneous (and therefore impossible) heading change. Similarly, even a transition between two different path curvatures at the same heading is infeasible at nonzero ve-

locity; discontinuity in time derivatives, such as velocity and acceleration, are equally undesirable. The elimination of such infeasible plans requires a mechanism to enforce continuity at the vertices.

3.2 State Lattice Motion Primitives

Assuming lattice state discretization outlined earlier, motion primitives are defined here to be the controls that connect roadmap vertices (states) and that are feasible motions. Further, enforcing state continuity requires that the state dimension vector be augmented with appropriate dimensionality. A simple example of such a search space is the state lattice for a Reeds-Shepp car [115] in Figure 3.2.1. This system has the same kinematics as the one in Example 1.2.1, but with a restriction that its steering angle can have only three values, $[-\phi_{\max}, 0, \phi_{\max}]^T$, where $\phi_{\max}$ is the maximum steering angle of the robot. A slightly more involved example of a state lattice for a car-like robot without such a restriction is shown in Figure 3.2.2.

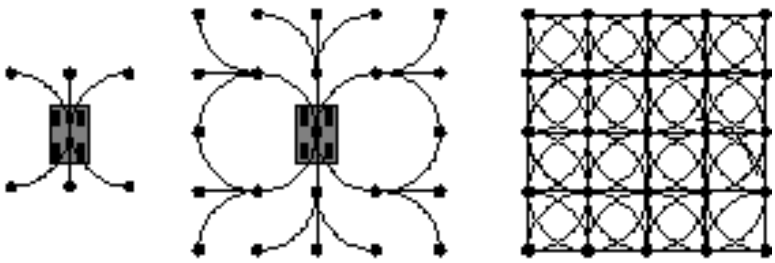


Figure 3.2.1: **State lattice for Reeds-Shepp car**. The Reeds-Shepp car can move forward and backward and it can drive straight or turn left or right at a fixed curvature. The presented rudimentary control set is derived from these basic controls by choosing their length such that only four heading values are used. Left: The designed control set precisely hits vertices in a rectangular grid. Center: The reachability tree to depth 2. Right: The reachability tree obtained by copying the control set at every vertex in a C space with 4 headings. Each dot represents 4 distinct vertices overlaid over each other, each representing different values of heading. While this search space will not generate a turn exceeding the chosen curvature, and while heading is continuous across vertices, the instantaneous transitions among curvatures at the vertices do not respect steering rate limitations.

By following these principles, we arrive at the construct, referred to as a *state lattice*, a roadmap

that consists of motion primitives developed in such a way as to connect the vertices of the roadmap, sampled as a lattice in state space. By sampling regularly, the connectivity of this roadmap is completely specified with its control set. Any systematic graph search algorithm can be utilized to find the shortest path in the state lattice graph.

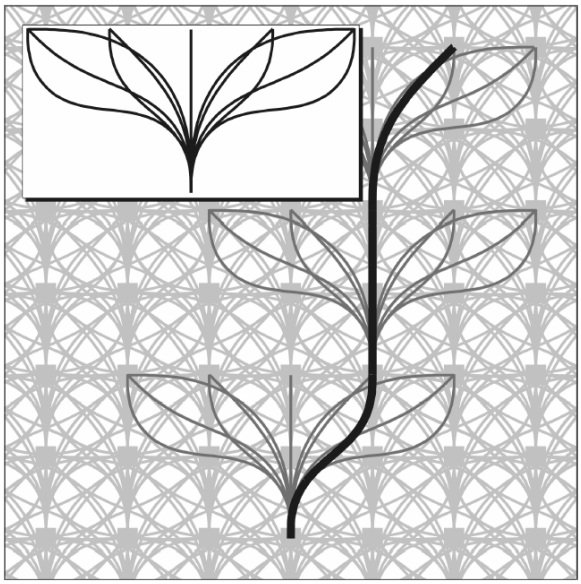


Figure 3.2.2: **State lattice for general car-like system**. This is one of the potential designs of a state lattice for a car-like system in Example 1.2.1. The motion primitives are regularly arranged in state space; to design them, a convenient state sampling rule is chosen (e.g. a low discrepancy lattice), and a controller is used to connect the samples via feasible motions. Given heading discretization in this particular example, such a set is to be defined for each of the 8 discrete values of initial heading, the multiples of $\pi/4$; symmetry can be exploited to define only two such sets, corresponding to the initial heading in the interval $[0, \pi/2)$.

The process of constructing the state lattice assumes a capability to determine a control to the system that steers it from one state to another. In general, finding this control is identical to solving a BVP; exact solvers are available only for simple systems [30; 42; 115], but a number of approximate solutions have been developed [52; 63], and this area continues being investigated in robotics due to its parallels with the problem of controller design.

Fortunately, the work in constructing the state lattice can be performed off-line, without affecting planner runtime. Once it is constructed and represented compactly with a control set, the state

lattice is readily used by the search algorithm that omits consideration of differential constraints.

3.3 PROPERTIES

In this section we review the properties of the proposed type of motion primitives, including their advantages and limitations. This presentation directly addresses the challenges of the problem that have been identified at the outset (Section 1.2.2).

3.3.1 FEASIBILITY

As suggested earlier, the feasibility of the motion plans computed using the state lattice is guaranteed by the design of the motion primitives (developed to be feasible motions) and by the appropriate dimensionality of the roadmap vertices that guarantees state continuity at the connections of the motion primitives.

However, in most realistic scenarios in robotics, the planned trajectory is rarely executed verbatim. There are frequent perturbations due to unmodelled effects and uncertainty both in the models of the environment and the vehicle itself. As described in 1.2.1, a standard approach to resolving this difficulty is by introducing a separate trajectory following controller module. It generates corrective input based on error feedback to ensure that the planned trajectory is followed closely. Including a trajectory following controller in the design does not offset the value of planning feasible motions, since infeasible ones are impossible to follow, as suggested in Example 1.2.2.

Another matter of concern with respect to feasibility is that only approximate BVP solvers are available for most non-trivial systems. As a consequence, computed trajectories may not connect the lattice states exactly, and small gaps may exist at the connections. Even though these gaps may be closed linearly, it is nevertheless required that the overall trajectory be Lipschitz continuous with a constant that is reasonably small in order to make sure that the trajectory would lend itself to following with reasonable accuracy [117]. This evaluation may require extensive computation, but it takes place off-line and does not affect the complexity of on-line planning.

Significant disturbances in the environment, such as dynamics effects due to moving on slopes or in wind, may be overwhelming for a controller, but they may be accounted for in planning as additional state variables. If these effects can be modeled in a discretized fashion, they would

naturally fit in the state lattice framework. For example, different control sets may be developed for various discrete values of these effects, without any significant increase in computational complexity (notice that this design change would not increase the number of motion alternatives that are to be evaluated during planning).

3.3.2 OPTIMALITY

As with most discretization techniques, a certain loss in the attainable minimization of trajectory cost, with respect to continuum, results from the constraint that the motions are arranged in a particular manner. For example, minimizing the length of primitives may be more challenging if a constraint on endpoint arrangements is placed. Even if an optimal search technique is used, e.g. A* [104], the computed motion is only optimal with respect to the chosen motion discretization. However, as suggested in Section 1.2.1, this trade-off is frequently made in this domain, since exact solutions are unknown for most of the systems of practical interest.

Moreover, the freedom of choosing any desirable state sampling is an important benefit and may allow fitting the search space to the known structure of the environment. This strength of the approach was utilized recently to fit search spaces to such settings as parking lots [90], roads [118], mines [34] and indoor environments [119].

3.3.3 RUNTIME

The runtime advantage of the proposed method is an especially salient one. It is enabled in two important ways:

- Off-line pre-computation of most of the components of the motion planner, including the good state and control sampling, trajectory cost evaluation, and even the search heuristic;
- Compatibility with some of the most efficient planning methods, including incremental and anytime search, multi-directional search etc.

The pre-computation property is afforded by the lattice regularity property that allows designing primitives to be position-invariant. Experience with fielded applications demonstrated that position-invariance assumptions are often satisfied in practice, as long as a trajectory following

controller absorbs external perturbations [65; 66; 81; 114]. Once computed, position-invariant primitives can be utilized anywhere in the search space, thereby moving integration of the controls to the planner design phase. This is more efficient than affine invariance [43; 46], since transformation of primitives during search is limited to translation.

Compatibility with a wider range of search algorithms than is typically attainable in planning with differential constraints is afforded by the structured reachability tree pruning rules. Motion primitives have been previously designed by sampling the control space directly [11; 60]. Such approaches typically guarantee feasibility by elaborating the primitives $e_s \in \mathcal{E}$ by choosing a control $u_s \in \mathcal{U}$ and integrating (1.2.1) for a certain Δt. The successor vertex v_s is established at the endpoint of e_s. Since, without special considerations [106], such motions and their endpoints will be dense in X, these planners attempt to prune the edges that are very similar, redundant or otherwise do not contribute to efficient exploration of X. For example, if the endpoint of a certain edge e_s is a vertex v_s, then a second edge e'_s terminating in v'_s is discouraged if distance $\rho(v_s, v'_s)$ is small. To this end, these approaches establish a discretization of X into cells, as shown in Figure 3.3.1. If a cell contains v_s, then e'_s is ignored if v'_s would occupy the same cell. The proposed lattice primitives may be used identically, except that the need to detect similar edge end-points is eliminated, since all motions are designed to arrive at specific state values, e.g. v_c in Figure 3.3.1. A similar pruning rule is in effect in this setting, except that it is developed off-line during search space design. As Chapter 4 further discusses, the application of a variety of attractive search techniques, such as incremental search [71], incremental sampling [88] and bi-directional search [75] is not possible without enforcing the graph structure as proposed here.

One drawback that may be experienced with the proposed primitives is the potentially significant computation that may be required to design this type of primitives (perhaps running the BVP solver repeatedly). However, this computation is off-line and does not affect the runtime of the planner.

3.3.4 COMPLETENESS

By providing the freedom to choose an arbitrary sampling of state space for primitive endpoints, high quality state sampling policies (low discrepancy and dispersion) may be utilized, leading to efficient exploration of state space during search. State space is typically simpler to sample

effectively than control space. However, the resolution-completeness property is not guaranteed because continuously increasing state sampling resolution does not, in the limit, necessarily also take control sampling resolution towards its limit [135]. It is important, however, that the sampling in control space is directly induced by the state lattice, as motions that fit the lattice are found *a posteriori*. Thus, sampling effectiveness of the state lattice is inherited in control sampling to a greater degree than other approaches that only use state sampling as a pruning rule [11]. As Figure 3.2.2 illustrates, the resulting primitives feature good path diversity, suggesting good sampling quality [31]. As described in Chapter 8, this was also observed during experimentation in simulation and with real robots.

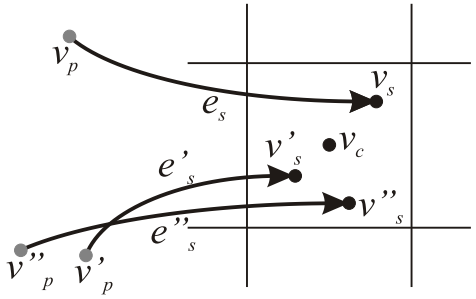


Figure 3.3.1: **State cell predecessors**. Three control-sampling primitives, edges $\{e_s, e'_s, e''_s\}$, emanate from their corresponding predecessor vertices $\{v_p, v'_p, v''_p\}$ and arrive at successor vertices $\{v_s, v'_s, v''_s\}$. The square cell represents a region in state space used for reachability tree pruning [11]. As all three successors land in the same cell, they are considered to be identical. Without special considerations to designing primitives, successor states will be dense. State lattice primitives, however, may be steered to converge to the same state v_c, leading to a structure that can be exploited for efficient planning.

CHAPTER 4

SEARCH TECHNIQUES

This chapter is devoted to a discussion of the details of applying standard search techniques to state lattices. As was mentioned earlier, the essential principle of the present approach is that, with the definition of the search space we have constructed, motion planning is reduced to unconstrained search in this space. The goal of this chapter is to review several established search algorithms and describe how they can be applied in the context of state lattice search spaces.

4.1 DETERMINISTIC SEARCH

Since the control set is an implicit representation of the search space, the roadmap is incrementally elaborated during search, so that only the states that are explored by the search are stored in memory. Any systematic graph search algorithm is appropriate for finding a path in the lattice. It is typically desired that the planner return optimal paths with respect to the desired cost criterion

and it be efficient. This section discusses A* [51] and D* [71; 130] search algorithms applied to state lattices, as they are some of the more popular among the deterministic search algorithms.

The strengths of deterministic, exhaustive search include attractive guarantees, such as optimality (under certain conditions, such as heuristic admissibility) and resolution-completeness, if afforded by the search space. One drawback, however, is the so-called "curse of dimensionality", the exponential growth of complexity with dimension of the search space. Nevertheless, this search technique remains attractive for systems that can be modeled well in several dimensions, including car-like [90], tracked [34], flying [46] and other systems of practical interest.

4.1.1 TRAJECTORY SWATHS

It is widely accepted that one of the most computationally intensive procedures in planning is collision detection and estimating the motion cost. In order to compute this cost accurately, it is necessary to simulate the behavior of the vehicle, subject to the corresponding control in the environment. In mobile robotics applications, the cost of robot motion is often represented via a workspace-based cost map, as suggested in Section 1.2.1. Then, the cost estimation is achieved by convolving the vehicle frame along the workspace projection (path) of the edge in question (Figure 4.1.1). Since the motion alternatives can be pre-computed, their paths (workspace projections) can be cached as well, perhaps in the form of trajectory swaths, the set of cost map cells $C_s \subset \mathbb{N}^2$ that are occupied by the robot footprint during motion (gray cells in Figure 4.1.1). Hence, instead of the costly vehicle simulation, perhaps using Runge-Kutta or similar methods, edge cost computation can be reduced to accumulating the cost over a pre-computed set of map cells, resulting in potential orders of magnitude speed-up in overall planning.

4.1.2 STATE LATTICE SEARCH HEURISTICS

Well-informed search heuristics have the potential to increase the efficiency of the search substantially [108]. Developing good heuristics for planning with differential constraints is a challenging problem, and various approaches are being developed [34; 41].

Among the simplest options for a heuristic estimate in the given context is the Euclidean distance metric. Weighted Euclidean distance, where dimensions are scaled differently, and Mahalanobis distance can also be applied. Such metrics are computationally efficient and can be defined

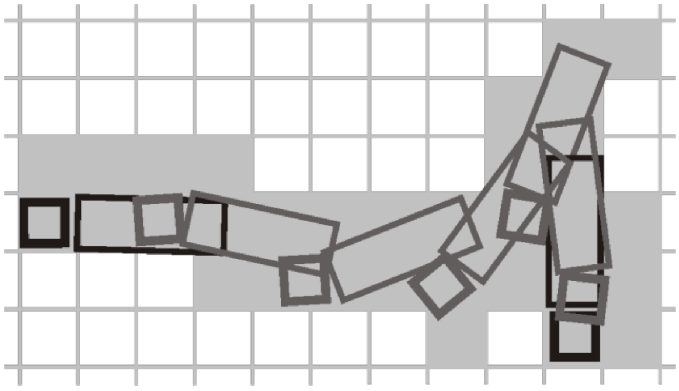


Figure 4.1.1: **Trajectory swath**. One of the motion primitives from a state lattice developed for a tractor-trailer robot is shown. It is a maneuver that re-orients the robot by 90° in heading. The trajectory swath was computed using Runge-Kutta simulation and cached off-line. During on-line planning, the evaluation of the cost of this maneuver with respect to a cost map is reduced to a summation operation over a memory array, resulting in approximately 100x speed-up.

to satisfy the admissibility requirement [108]. However, in many cases, such approaches vastly underestimate the true path length in the lattice, resulting in inefficient search.

Better informed heuristics can be designed by using the vehicle controller. Since this module typically is endowed with the system model, it has the potential of estimating the heuristic cost of a motion more accurately. However, as Example 4.1.1 illustrates, even though a heuristic based on Reeds-Shepp distance [115] is a much better informed one for a car-like robot than the Euclidean distance, it still suffers from significant inaccuracy.

Similar to trajectory swaths in Section 4.1.1, state lattice structure enables a straight-forward pre-computation of the free-space costs of motions, stored in a look-up table, leading to a very well-informed heuristic in terms of actual cost of feasible motions [69; 90; 109]. Such a Heuristic Look-Up Table (HLUT) can be setup as a multi-dimensional array, indexed by x_F (assuming x_I is at the workspace origin). Each element of this array is a scalar that represents the cost of steering the system from the origin to x_F, in free space. Given a state space ball of certain radius around the origin, a set of roadmap vertices in the ball is given by lattice structure, and the heuristic value is pre-computed for each of the vertices. By the translation symmetry, heuristic evaluation during on-line planning is reduced to a simple table dereference. This approach enables well-informed

and fast heuristic evaluation in the state lattice structure.

An inherent limitation of pre-computing the heuristic dataset is the memory requirement for storage. Even though the cost and amount of computer memory is continuously increasing, it is interesting to seek methods to alleviate this requirement. One approach is to utilize an *approximate* HLUT that has more sparse sampling than the planner search space. The samples that are omitted from the HLUT may be interpolated. In this manner, it is possible to alleviate the storage requirement significantly, while still preserving much of the efficiency of this heuristic method.

Example 4.1.1 (HLUT Heuristic for a Car-like Robot)**.** *This example illustrates the HLUT Heuristic and contrasts it with other popular types of heuristics in this domain. For simplicity of exposition, the car-like system in Example 1.2.1 is reused here. State space is assumed to be four-dimensional, including 2D position, heading and steering angle. Path length is chosen as cost of motion. Recall that here we look at free-space heuristics (no assumptions about the environment are being made), so obstacles were not considered in the comparison. An illustration of the resulting HLUT is given in Figure 4.1.2.*

Figure 4.1.3 compares HLUT to two other heuristic candidates, Euclidean and Reeds-Shepp distance. It shows the ratios of the distance given by these two metrics to path length pre-computed in the HLUT.

Euclidean distance is very fast to evaluate, but unfortunately results in a poor estimate of the car-like path length. Reeds-Shepp distance results in significantly better heuristic estimates, but it still notably lower than the HLUT values (Figure 4.1.3). This metric assumes that the vehicle utilizes only three controls: turns left and right at the maximum value of steering angle and straight line motion. This leads to paths that are discontinuous in curvature (i.e. infeasible to execute without stopping). Even though continuous motion results in longer path length, it is nevertheless typically preferred over discontinuous alternatives in practice. For the car-like system, utilizing a pre-computed HLUT results in over 100x speed-up in planning runtime, as Figure 4.1.4 demonstrates.

To conclude this example, the performance of two variants of approximate HLUT are demonstrated. One removes the steering angle dimension of the initial state, x_I, for each of the queries being considered. Instead of storing that value, it sets all values of x_I's steering angle to zero. As expected, this approximation leads to a certain runtime increase during planning, as Figure 4.1.5a) shows. A similar experiment was reproduced by removing the final steering angle dimen-

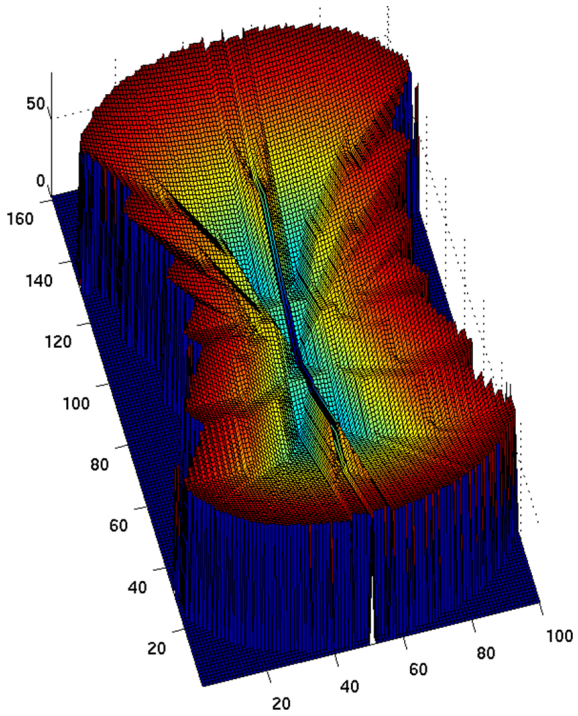


Figure 4.1.2: **Car-like HLUT**. A visual representation of an HLUT for a car-like robot from Example 1.2.1. Figure shows a 3D view of the dataset, where the Z-axis represents the cost (path length) of moving the car from the origin position (the center of the plane defined by the X and Y axes) to a different position with the same orientation. Note that moving along the initial orientation entails low cost, but moving perpendicular to it ("sideways") incurs higher cost due to a maneuver that is required under nonholonomic constraints.

sion in Figure 4.1.5b). The results suggest that removing the final steering angle dimension results in a smaller increase in runtime, and is therefore preferred. □

4.2 Incremental Search

In many applications featuring physical robots, it is beneficial to perform incremental search: once a plan is computed, it is incrementally updated (ideally, while reusing previous computation) should new information about the environment invalidate it [71; 130]. This capacity enables the planner to react quickly to the changes of the world model, including those due to uncertainty

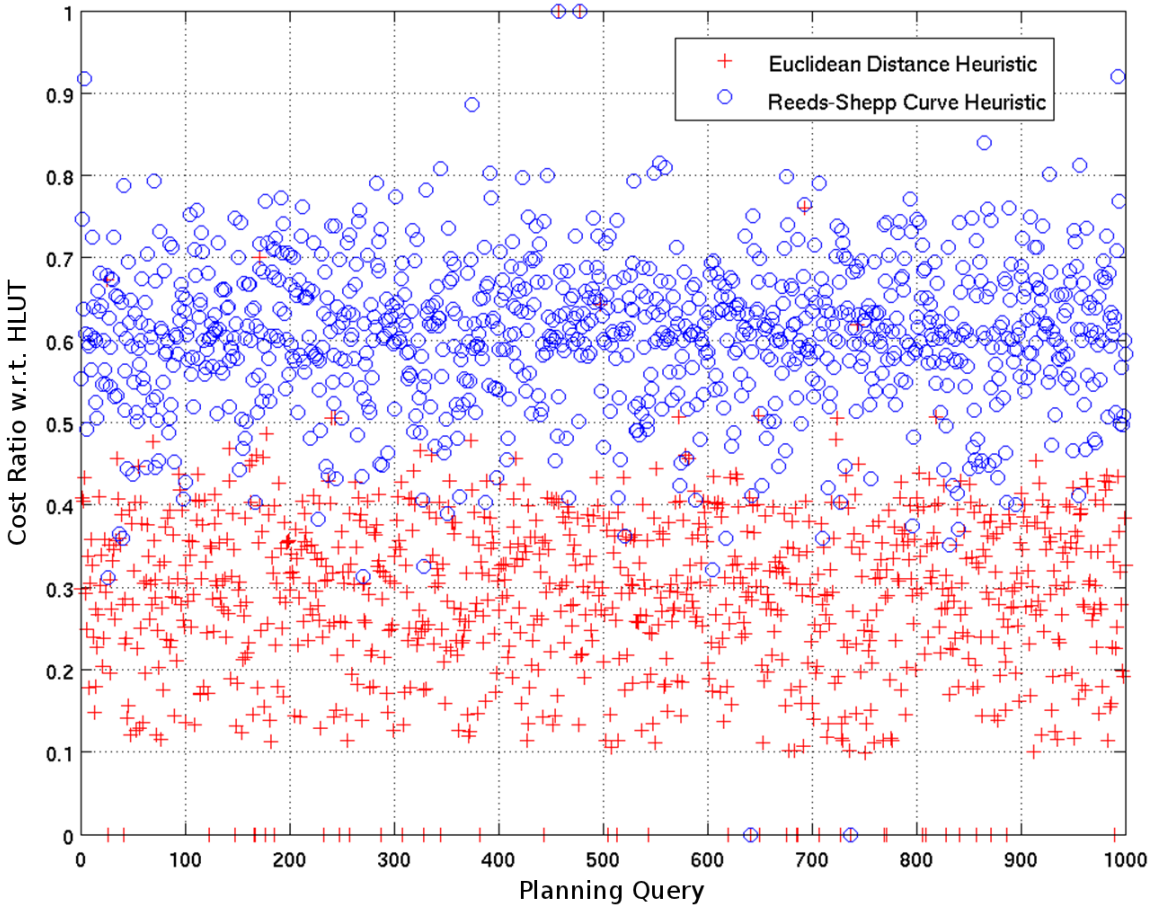


Figure 4.1.3: **Heuristic comparison**. Euclidean and Reeds-Shepp distance metrics are compared with the HLUT, in the context of a car-like system. The units of each of the three metrics were maintained to be the same. Reeds-Shepp and Euclidean distances are fairly significant underestimates of the true path of the car-like robot. This observation motivates an inquiry into better-informed heuristics for this type of motion planning.

and noise of the perception, localization and other systems – a frequent and inevitable phenomenon in robotics. This type of search is a typical component in many fielded robotics systems, since plan repair can be vastly more efficient than replanning from scratch [64; 89; 129].

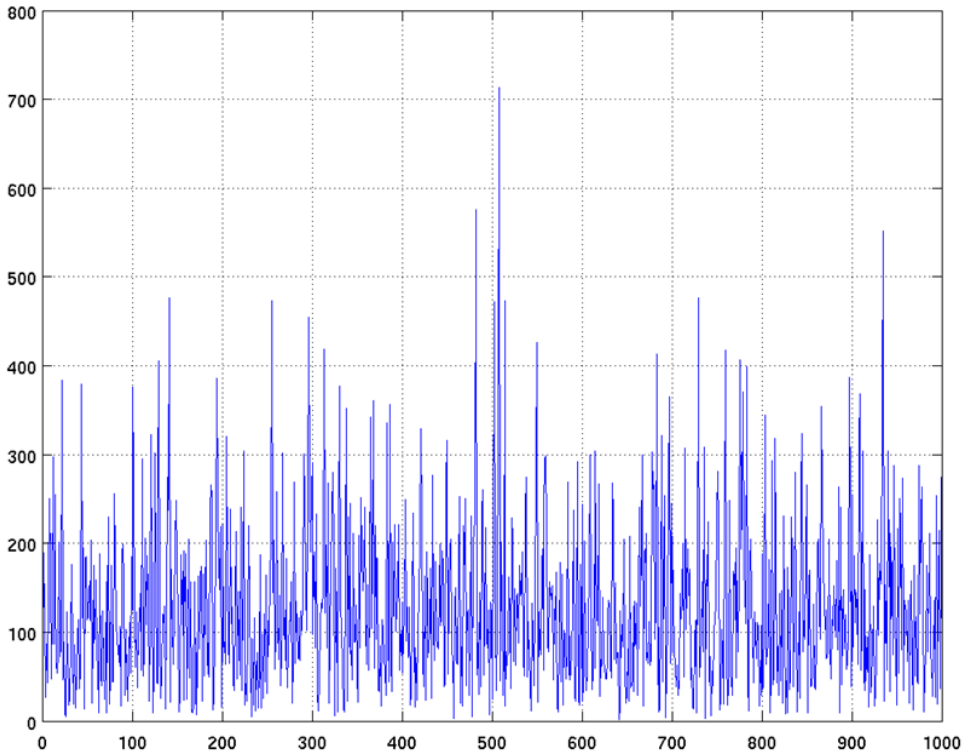


Figure 4.1.4: **HLUT vs. Euclidean Comparison**. This figure plots the (seconds) runtime ratios, of executing a set of motion planning queries with a car-like robot model with two A* planners, where the only difference was the choice of the heuristic: HLUT and Euclidean distance. The plot demonstrates that utilizing an HLUT leads to over 100x speed-up in planning runtime.

4.2.1 MANAGING PREDECESSORS

As discussed in Section 3.3.3, motion primitives that are designed by sampling the control space without regard for state space structure typically lead to tree-shaped search spaces [11; 60; 87]. In contrast, the most efficient formulation of incremental search requires cyclic graph topology. The operation of such search requires an ability to enumerate and keep track the edges that arrive at a particular state, e.g. the edges $\{e_s, e'_s, e''_s\}$ leading to a cell containing v_s in Figure 3.3.1. A key component of incremental search algorithms is the *rhs*-value, the one-step look-ahead cost value, that is defined as in [71]:

$$rhs(v_s) = \begin{cases} 0 & \text{if } v_s = x_{init} \\ \min_{v_p \in Pred(v_s)} (g(v_p) + c(v_p, v_s)) & \text{otherwise} \end{cases} \tag{4.2.1}$$

where $g(v_p)$ denotes the cost of the vertex v_p, $Pred(v_s)$ is the set of the predecessors of v_s, e.g. $\{v_p, v'_p, v''_p\}$ in Figure 3.3.1. One of the ways of reusing previous computation is the capacity to pick a different predecessor of a state in the event that the current predecessor edge increases its cost. However, as illustrated in Figure 4.2.1, the predecessor edges cannot be swapped in general because the distance between their endpoints $\rho(v_s, v'_s) \neq 0$ (by their density). The Lipschitz continuity argument may be invoked to ignore this gap, but it is not clear that a bound on the Lipschitz constant may be guaranteed *a priori*. This issue is resolved with motion primitives that conform to graph structure in X: this distance is zero, since all predecessors converge to the identical v_c. Note that this process of managing predecessors is similar to *resolution equivalence* in traditional applications of incremental search in grids, as analyzed by Tompkins [133].

4.2.2 Processing Edge Cost Updates

Moreover, in grids, it is typically easy to determine the set of cells (and, consequently, edges) that are affected when a region in the workspace changes cost. With motion primitives of arbitrary length and form, this computation is more involved, since the edges may span several cells. However, by structure regularity, this computation can be done *a priori*. Once we compute the swaths of the motion primitives, we collect and store a list of edges that pass through the origin cell. By translation symmetry, this list may be reused anywhere else in the search space.

More formally, for every change in the cost, $c(x_i, x_j)$, of the directed edge from the vertex x_i to x_j, a replanning algorithm requires recomputing the priority of x_j and potentially inserting it into the priority queue. Assuming a map cell $m_{ij} \in \mathbb{N}^2$ changes cost, the planner needs to know the set of vertices V_c that potentially need to be re-inserted into the priority queue with new priority. Thus, the planner requires a mapping $Y : \mathbb{N}^2 \to V_c$. This mapping is simple in the case of classical applications of D* in 2D grids, but it is nontrivial in the case of the state lattice.

To develop this mapping, we need to compute the *swath* of a motion, a set of cost map cells $C_s \subset \mathbb{N}^2$ that are covered by the robot as it executes the motion. The cost of an edge that represents this motion is directly dependent on the costs of map cells in C_s. Once we pre-compute the control

set, it is also possible to pre-compute the swaths of the edges in it. Once we have a mapping from a set of edges in the control set to the map cells in their swaths, that mapping is then inverted to obtain a set of edges that are affected for a map cell. By translation symmetry, this set of edges is the same for every map cell. It can be precomputed for the same reasons as the control set itself and can be used readily to incorporate cost map changes in re-planning.

Example 4.2.1 (Nonholonomic D*)**.** *This example illustrates the use of the proposed state lattice search space to enable, for the first time, the application of D* search algorithm [70; 130] to motion planning with nonholonomic constraints. The car-like robot model from Example 1.2.1 is once again reused for this example. Figure 4.2.2a) shows the initial plan that involved a large number of vertex expansions (gray cells). A point obstacle was introduced that would cause the robot (red 6-wheeled vehicle frame) to collide, if it were to execute this path. Figure 4.2.2b) shows the motion plan after replanning. Runtime and plan cost values are given in Table 4.1. Note the significant reduction in planning time due to reusing previous computation and a slight increase in the cost of the motion due to the generated obstacle avoidance maneuver. Figure 4.2.3 illustrates the predecessor invalidation set that was pre-computed for this system.* □

Plan Type	Iterations	Runtime, sec.	Motion Cost
Initial plan	11040	6.0	66.021
Replan	512	0.3	66.053

Table 4.1: **Nonholonomic D***. Runtime and cost data for the test in Figure 4.2.2. Note the significant reduction in planning time due to reusing previous computation and a slight increase in the cost of the motion due to the generated obstacle avoidance maneuver.

4.3 INCREMENTAL SAMPLING

The advent of randomized planning in recent years has spurred inquiry into effective sampling methods that avoid the "curse of dimensionality" while offering better solution quality and completeness guarantees than standard randomized approaches. Deterministic incremental sampling techniques have been proposed as viable alternatives [21; 73; 88]. One of them is the Halton points, a d-dimensional generalization of the van der Corput sequences of d bases, one for each coordinate

[50]. A basic version of such incremental sampling in square grids can be thought of as increasing discretization resolution by a factor of 2^{di}, $i \in \mathbb{N}$.

The planners that do not enforce structure in edge connectivity would be required to regenerate the plan from scratch every time the sampling resolution is incremented. However, lattice primitives enable the reuse of previous computation via the same mechanism that allows incremental search. Due to regular structure, the connections between primitives belonging to different resolution levels become trivial, thereby allowing the results of planning at different resolution to be reused. One application of this approach is *anytime* planning, where the quality of the computed plan is improved with more computation.

4.4 MULTI-RESOLUTION SEARCH

By virtue of the state lattice's general representation as a directed graph, it naturally can be extended with multi-resolution enhancements. Significant planning runtime improvement was achieved in the literature via a judicious use of the quality of representation of the planning problem, e.g. [18; 36; 105; 134] among others. In field robotics, it is frequently beneficial to utilize a high fidelity of representation in the immediate vicinity of the robot (perhaps within its sensor range), and reduce the fidelity in the areas that are either less known or less relevant for the planning problem. Lower fidelity of representation is designed to increase search speed, while higher fidelity provides better quality solutions. Since traditionally grids have been utilized in replanning, the notion of varying the quality of problem representation has been identified with varying the resolution of the grid. However, this thesis relies on the discretization of both the state and motions. Since the term "resolution" is typically used in 2D scenarios, we have termed managing the fidelity of state lattice representation as *graduated fidelity*.

In designing the connectivity of regions of different fidelities, care must be taken to ensure that all fidelities consist of motions that are feasible w.r.t. the robot's mobility model. If this rule is violated, mission failures become possible due to limit cycling at the border of fidelity regions due to obstacles, as suggested in Example 1.2.2. In order to avoid such difficulties, it is sufficient to ascertain that all levels of fidelity include feasible motions. For example, the connectivity of low fidelity regions could be a strict subset of that of the high fidelity regions.

To implement graduated fidelity planning and efficient replanning in a manner similar to the

framed-quadtree multi-resolution planning [134], the presented design requires almost no modification. Once the state lattice graph is separated into regions of different fidelity as desired, each region uses its own control set to achieve the chosen fidelity (resolution of vertices and connectivity of edges between them). Each control set provides the successors of a vertex being expanded during search. Care must be taken to design the control sets such that they would adequately span the boundaries between the regions. Note that control set design is the sole procedure needed to enable graduated fidelity. The search algorithm requires no changes and will achieve the desired effects automatically.

Moreover, thanks to graph representation of the state lattice, it is also fairly straight-forward to extend the concept of graduated fidelity by allowing the different regions of fidelity to move or change shape between replans. This becomes useful to enable the high fidelity region to remain centered around the moving vehicle. If the search space remains fixed to the world frame, a subset of the search space changes connectivity as it moves along with the vehicle. Such flexibility results in a dynamic search space, which complements dynamic replanning algorithms to allow greatly increased planning efficiency.

As shown in [111], enabling such flexibility in the search space once again requires hardly any change to the replanning algorithm. The change of connectivity between replans is presented to the algorithm as a change in cost of the affected graph vertices, similar to the change in cost due to perception updates. The two kinds of cost changes appear identical to the replanning algorithms, and they are equally efficient in handling both. Thus, the graduated fidelity extension of state lattice planning is conceptually simple and easy to implement, while offering unprecedented planning efficiency under differential constraints.

In this section, we describe an approach to reducing the computational complexity of planning with state lattices at the cost of selectively reducing the quality of representation. Some regions of the workspace, e.g. those that are partially or completely unknown, can be represented at lower fidelity, and the regions that are best known or most relevant for the problem are represented at highest fidelity.

We extend the above definition of the state lattice by assuming that it consists of subgraphs $\mathbf{G_1}, \mathbf{G_2}, \ldots, \mathbf{G_n}$. The arrangement of vertices and edges in each subgraph is assumed to be regular, but this arrangement may be different among subgraphs to reflect the differences in the fidelity of representation. This composite search space is maintained to remain a directed cyclic graph, so

that replanning algorithms can be utilized to reuse previous computation while replanning.

We define *modifying* a subgraph as arbitrarily changing its position (in coordinates that do not affect its connectivity, namely the translational ones) and shape (the extent of its boundary in state space). After a subgraph is modified, some of the vertices near its boundary are set to belong to a different subgraph. Note that the vertices do not move, they simply change ownership from one subgraph to another. For example, as a subgraph changes position, it "gains" the vertices and edges on its leading interface and "loses" vertices and edges on its trailing interface.

After all subgraphs are modified as desired, a replanning procedure needs to be executed to repair the plan. This is the same procedure that repairs the plan due to other changes in the search space (e.g. a perception update). To see why, note that deterministic replanning algorithms reuse computation by storing previously computed costs of vertices in the graph. As a result of modifying the subgraphs, some vertices change their cost due to their connectivity under a different subgraph. It is entirely transparent to the replanning algorithm whether they changed cost due to new edge costs from a perception update, or due to modification of subgraphs. Thus, the only required change to replanning algorithms to enable graduated fidelity is a process to make them aware of the vertices that have new subgraph ownership. This is performed by the function *convert_vertex*, presented in Algorithm 4.1.

The function *convert_vertex* is executed on each vertex v_b that changes subgraph ownership. If the vertex has not been expanded at all so far, the function returns. Otherwise, we note all vertices that may contain v_b as a predecessor – it is exactly the set of successors of v_b under the edge connectivity of its previous subgraph $\mathbf{G_i}$, denoted as $Succ_i(v_b)$. Further, we remove any back-pointers from these successor vertices v_s to v_b by examining the predecessors of v_s, $Pred(v_s)$. Effectively, we undo the effects of a previous expansion of v_b. Lastly, if this change resulted in a change of cost of any successor vertex, we insert the affected vertices and v_b itself into the priority queue. D* variants can detect this cost change automatically by recomputing the *rhs*-value. Note that this procedure is likely to cause replanning from the farthest affected vertex to the robot (assuming the search direction from the goal to the robot). Thus, more previous computation is reused if such changes occur closer to the robot.

This procedure is illustrated in Figure 4.4.1, using a simplified search space for ease of visualization. In this example, the search space consists of two subgraphs, $\mathbf{G_1}$ (black square vertices) and $\mathbf{G_2}$ (gray circle vertices). Arrows of similar colors are the edges. $\mathbf{G_2}$ is a small subgraph, consisting

```
Input: graph vertex v_b
if v_b was previously expanded then
    foreach v_s ∈ Succ_i(v_b) do
        remove v_b from Pred(v_s);
        update_vertex(v_s);
    end
    update_vertex(v_b);
end
```

Algorithm 4.1: The function *convert_vertex*(v_b). It enacts a change of subgraph ownership of the vertex v_b. After it has been called on all vertices that changed subgraphs, the search space becomes ready for a standard replanning process. The *update_vertex* function is assumed to be a component of the chosen search algorithm that determines whether a vertex needs to be processed during replanning.

of six vertices, highlighted with a gray bounding box. The rest of the search space belongs to $\mathbf{G_1}$. In this example, $Succ_1(v_i)$, $\forall v_i \in \mathbf{G_1}$ is 4 nearest neighbors, and $Succ_2(v_j)$, $\forall v_j \in \mathbf{G_2}$ is 8 nearest neighbors. A real implementation of graduated fidelity under differential constraints would utilize the same algorithm, but a more sophisticated connectivity of the subgraphs.

The effect of the initial plan is shown in Figure 4.4.1a). The search proceeded from the goal vertex v_g to the robot v_r. The vertices with white centers ("hollow") were expanded during search, and solid vertices, pointed to by the arrows, remain in the priority queue. The resulting motion plan is highlighted with a thick patterned line.

Next, suppose we move $\mathbf{G_2}$ to the right, as shown in Figure 4.4.1b). The six crossed-out vertices change subgraph ownership due to this move, and we execute *convert_vertex* on each of them. The previous expansion of vertices v_1 and v_2 is undone: their edges (dotted arrows) are removed. In Figure 4.4.1c), moving $\mathbf{G_2}$ is completed, and the affected vertices are inserted into the priority queue. When the search algorithm begins replanning, these vertices will be expanded, using the edge connectivity of $\mathbf{G_2}$, if they are relevant for the problem at hand, as deemed by the heuristic. Figure 4.4.1d) completes our example and shows a new motion plan (thick patterned line) due to moving $\mathbf{G_2}$ to the right.

The same algorithm would work with subgraphs of different dimensionalities by using additional "connecting" edges. They are used as part of the expansion of a vertex that lies on the boundary of a subgraph, in order to connect it with the other subgraph. A simple example of con-

necting a 2D subgraph $\mathbf{G_1}$ to a 3D subgraph $\mathbf{G_2}$ is shown in Figure 4.4.2. Additional edges (gray arrows in the figure), denoted by $Succ_{12}(v_b)$ are used in order to generate suitable connectivity from a vertex v_b in subgraph $\mathbf{G_1}$ to $\mathbf{G_2}$. In this case, we have connected this vertex to several vertices in $\mathbf{G_2}$ representing all possible values of the third dimension. Algorithm 4.1 can be utilized here with a minor modification. In addition to using the vertices in $Succ_i(v_b)$ in the *foreach* loop, we also use the vertices from $Succ_{ij}(v_b)$. This addition certainly increases the branching factor of boundary vertices, however the complexity effect can be insignificant if only a small number of vertices require the extra edges.

Example 4.4.1 (Graduated fidelity)**.** *To illustrate graduated fidelity concepts, the car-like system model from previous examples is slightly modified: it is endowed with a capacity to turn in place similar to differential drive vehicles. This is helpful in that it enables highlighting the differences in search space connectivity in the high and low fidelity regions. The former here is chosen to be just like the original car-like system, and the latter is a basic 8-connected grid (Figure 4.4.3).*

With this setup, by applying unmodified D, nearly an order of magnitude runtime improvement is attained, as shown in Figure 4.4.4, while the vehicle actually never has to resort to point turn motions, since high-fidelity region travels with the vehicle. This results in an execution advantage as well, since a point turn presumably would have required slowing the vehicle to a complete stop.* □

4.5 RANDOMIZED SEARCH

Lattice primitives may be utilized in randomized search in a manner that is similar to other types of primitives [44]. The local planner component, as suggested in [55; 61; 87] and similar planners, may be designed to choose an element of a set of primitives that is a good fit to extend the tree or the graph towards a random sample. However, by virtue of the regular structure of the state samples, lattice primitives would enable additional capacity to execute parallelized kinodynamic planning, e.g. bi-directional [75], as well as a series of independent searches [91]. Figure 4.5.1 illustrates that, unlike control-sampling primitives that would likely require multiple BVP solutions to connect partial planning results, the layout of lattice primitives makes this connection automatic.

Example 4.5.1 (State Lattice Bi-RRT)**.** *To illustrate the application of randomized search in a*

state lattice, this example applies bi-directional RRT search [75] to the car-like system. The paths of the motions computed by the RRT and A searches for three different planning queries are shown in Figure 4.5.2. Performance statistics on these tests are given in Table 4.2.*

□

Trial Num.	Bi-RRT		A*	
	Runtime, sec.	Motion Cost	Runtime, sec.	Motion Cost
1	4	194	10	73
2	1	39	5	30
3	0	42	1	23

Table 4.2: Comparison of a Bi-RRT and A* in the state lattice.

4.6 On-Demand Expansion

Two significant challenges of applying optimal graph search techniques include search space quality and computational complexity. Motion plans generated by graph search are only as good as the quality of the underlying search space. Search spaces that represent many detailed routes through cluttered environments generally outperform those with coarser approximations of all feasible motions. Computational complexity balances out the desire to represent very detailed, expressive search spaces because it is expensive in memory and processing to represent numerous nodes and evaluate all edges. In this section, an *adaptive* state lattice search space is developed in a manner similar to the approach by Ferguson and Stentz [35]. It can be thought of as the middle ground between a small, *canonical*, set of motion primitives that allows low planning runtime, but may suffer from losses in solution quality, and a larger, *expressive*, set of primitives that allows better quality at the cost of runtime. The search algorithm is endowed with monitoring logic to detect cases where most of the alternatives in the canonical control set are being invalidated due to environment constraints. In this case, the expressive control set is used temporarily; it's greater variety of motions may suggest a route that was not evaluated by the canonical set. For example, a global motion planner with both variable fidelity and expressiveness can benefit from a notable runtime improvement by searching expressively in perceived regions and coarsely in inferred, unknown, or

previously traversed areas.

Figure 4.6.1 illustrates this idea. The green paths represent the pre-computed motion alternatives for a canonical (red) path. The brown obstacle in the figure illustrates that, even though the canonical path may become blocked, the path alternatives may still remain viable options to enable the vehicle to move towards its goal while avoiding the obstacle. The black square cells illustrate the lattice cells that are different from the cells covered by the canonical path. Maximizing the number of these cells (set difference between the set of cells covered by the canonical path and the set of cells covered by a path alternative) is beneficial, since it increases the probability that the alternative will remain collision free even if the canonical path is blocked. This observation is well in line with other results regarding path diversity in this setting [20; 33; 49].

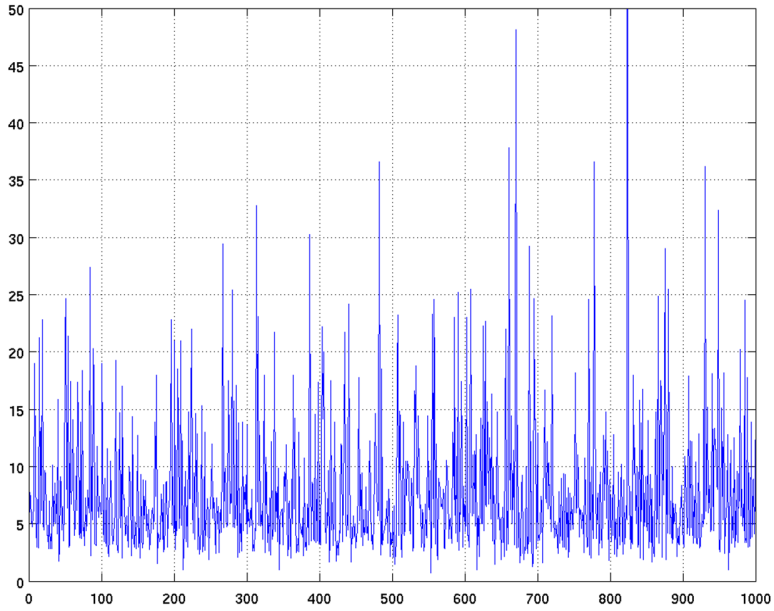
By regularity of the state lattice, a set of motion alternatives developed to fit state discretization at the origin can be reused anywhere else in the workspace. The approach has an added benefit of moving the computation of substituted variants of canonical motions from on-line to the off-line planner design stage. Based on earlier experience, this approach alleviates some computation burden during optimization-enabled navigation, especially for higher-order parametrized motions, such as clothoids [52] – at the cost of a more coarse-grained adaptation of paths to the environment.

Given a pre-computed set of motion primitives that adheres to the underlying state lattice structure, we now turn to methods of using it during planning. The most straight-forward method of using such a set is to augment the standard vertex expansion procedure to consider motion alternatives, should the canonical motion be blocked. For its application in this manner, we make a simple modification to the set of path alternatives. We have noted in prior work that shorter motion primitives appear to lead to better completeness properties of the resulting planner. Therefore, we sort the motions in the alternatives set by path length. If a canonical motion is not collision-free, we attempt to substitute it with alternatives from the set, in increasing order of path length. We stop this process when a collision-free alternative is found. This motion is used *en lieu* of the canonical path. From this point, the standard best-first planning process proceeds. Since the procedure is very similar to the typical vertex expansion, we refer to it as on-demand expansion to highlight the fact that additional motions are evaluated only when necessary, i.e. on-demand.

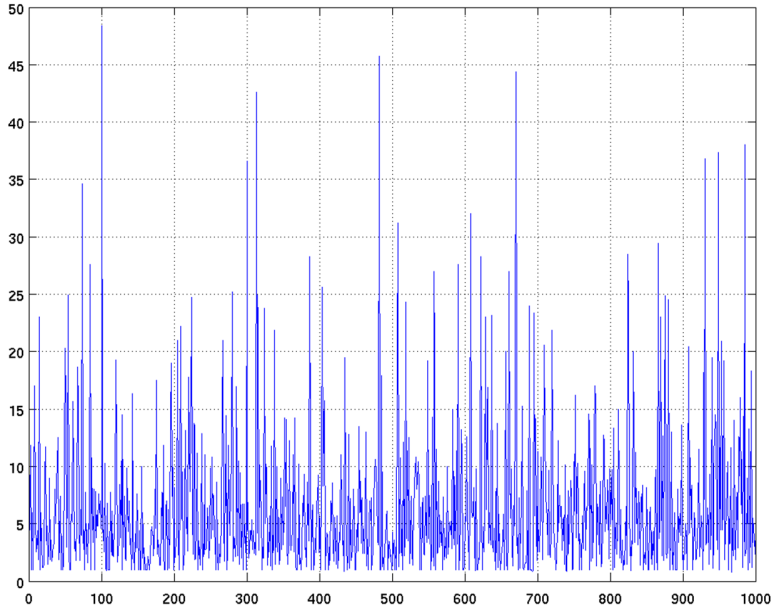
The capacity to perform on-demand expansion, enabled by the state lattice structure, is significant because it provides for a greater diversity of roadmap edges without any effects on the vertices (since all on-demand motion alternatives connect at the same vertex). As shown in Section 8.5,

this feature leads to as much as 2x runtime improvement in challenging planning problems.

This chapter reviewed the details of applying standard search techniques to the search spaces based on state lattice structure. As was mentioned in previous chapters, the essential principle of the present approach is that the proposed search space structure reduces motion planning with differential constraints to unconstrained search, while enabling several search paradigms previously under-utilized in this type of motion planning.



(a) Initial steering angle removed



(b) Final steering angle removed

Figure 4.1.5: **Approximate HLUT**. Planning runtime with two variants of approximate HLUT are compared here with the exact HLUT. A set of identical planning queries were computed with each of the heuristics, and the runtime (seconds) ratios are plotted on the vertical axis. The 1000 experiment repetitions are distributed across the horizontal axis. Top: HLUT with the initial steering angle dimension removed; average runtime increase w.r.t. full HLUT is 7.2x. Bottom: HLUT with the final steering angle dimension removed; average runtime increase w.r.t. full HLUT is 4.4x. The latter case (removing final steering angle dimension) leads to a smaller runtime increase and is therefore preferred.

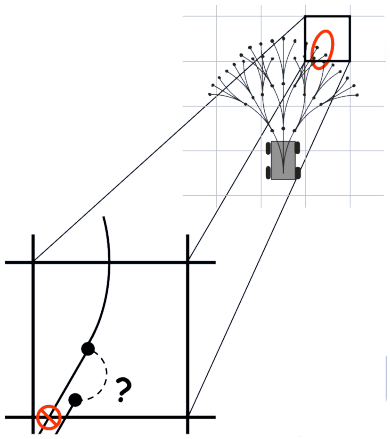
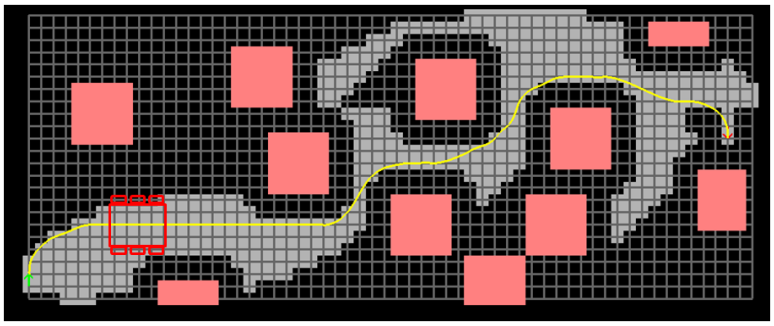
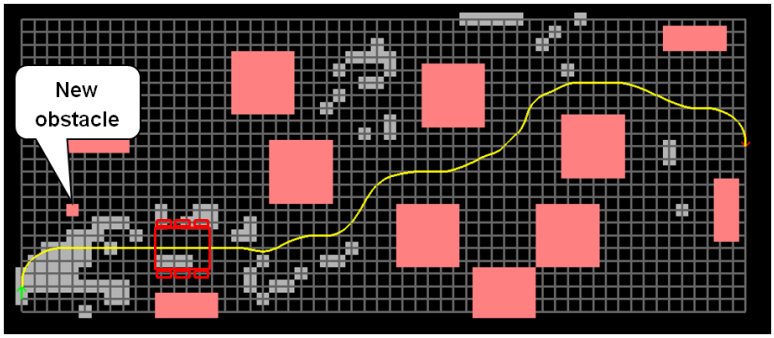


Figure 4.2.1: **Motivation for graph structure**. Sampling control space without regard for state space structure may not result in the cyclic graph structure that is necessary to reuse previous computation. For example, tree-based search spaces where vertices are dense in X, would not allow reusing previous computation via predecessor redirection. In case a predecessor edge becomes invalidated due to new environment information (red stop-sign in the figure), an attempt to reuse an alternative predecessor would result in a gap in the trajectory (question mark).



(a) Initial plan



(b) Replan

Figure 4.2.2: **Nonholonomic D***. A simple 2D example of incremental planning with D*. An initial plan (a) and an incremental replan (b) to "fix" the initial plan that was blocked by a point obstacle (new point obstacle in figure placed in collision with red vehicle frame). Grey cells are expanded vertices.

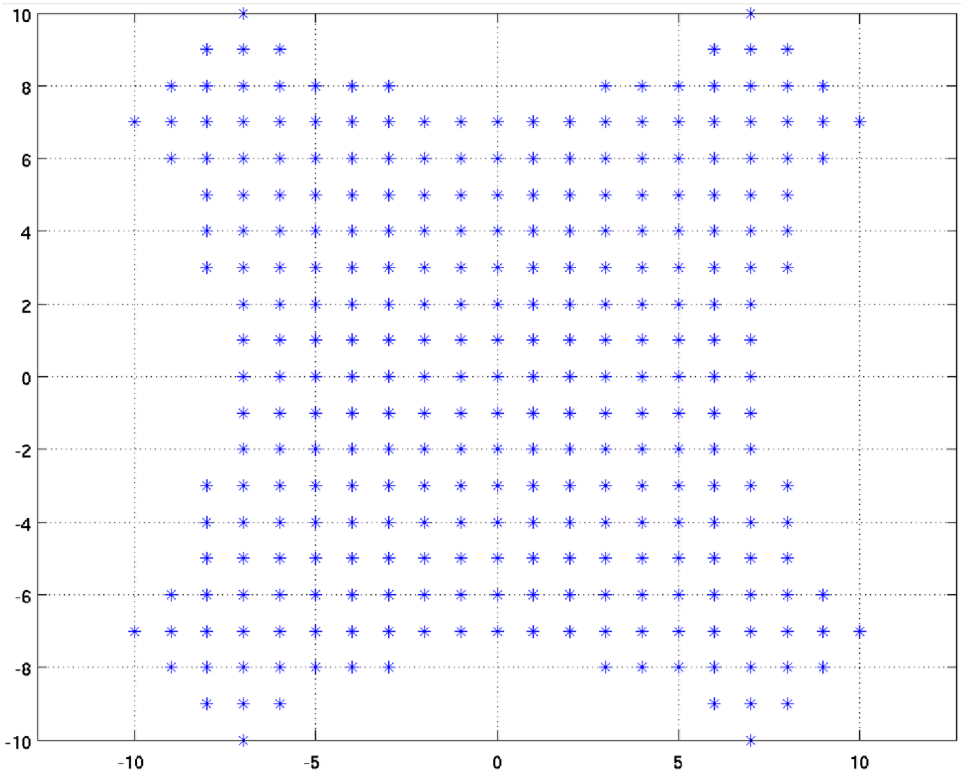


Figure 4.2.3: **Pre-computed invalidation set**. A set of states that are affected by a map cell that changes cost can be pre-computed and reused. This set was generated for the car-like system in Example 1.2.1.

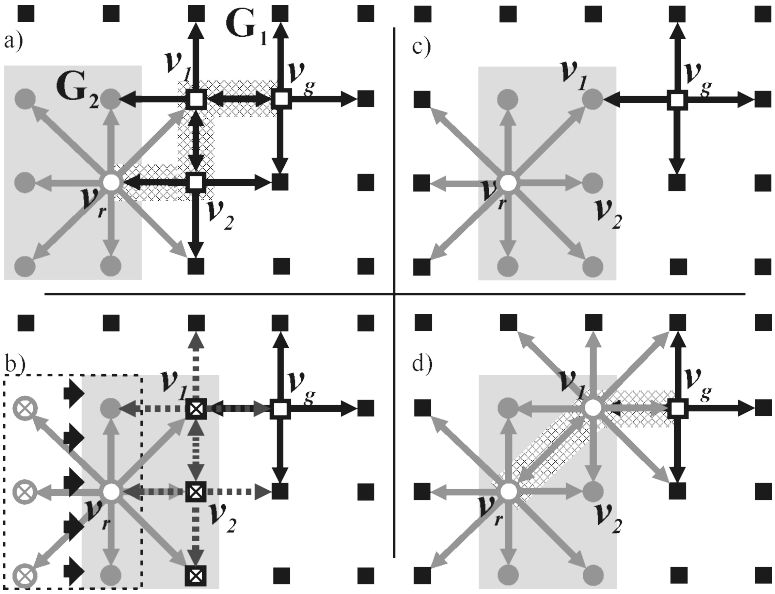


Figure 4.4.1: **Dynamic search space**. Maintaining the connectivity between two subgraphs of different fidelities of representation. $\mathbf{G_1}$ is a static subgraph (black square vertices), and $\mathbf{G_2}$ (gray circular vertices) moves w.r.t. the former. Arrows are edges. Hollow vertices have been expanded. Subfigure a) shows the initial plan (thick patterned line), as it originates in $\mathbf{G_1}$ and proceeds into $\mathbf{G_2}$; b) as $\mathbf{G_2}$ moves from left to right, the six crossed out vertices change subgraph ownership, and *convert_vertex* is executed on each of them, which results in undoing the previous expansions of v_1 and v_2; c) shows the completion of moving $\mathbf{G_2}$: the vertices v_1 and v_2 now belong to $\mathbf{G_2}$ and are available for re-expansion, if necessary, when the search algorithm performs replanning; lastly, d) shows the result of replanning in the search space from c), where due to re-expansion of v_1 under $\mathbf{G_2}$ edge connectivity, a new plan is found.

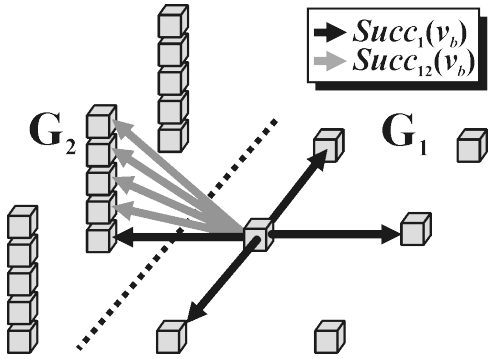


Figure 4.4.2: **Spanning dimension boundary**. The 2D subgraph $\mathbf{G_1}$ (4-connected grid) is connected to another subgraph $\mathbf{G_2}$ of a higher dimension. Black arrows are the standard node expansion (4 nearest neighbors), and gray arrows are additional edges that connect the two subgraphs.

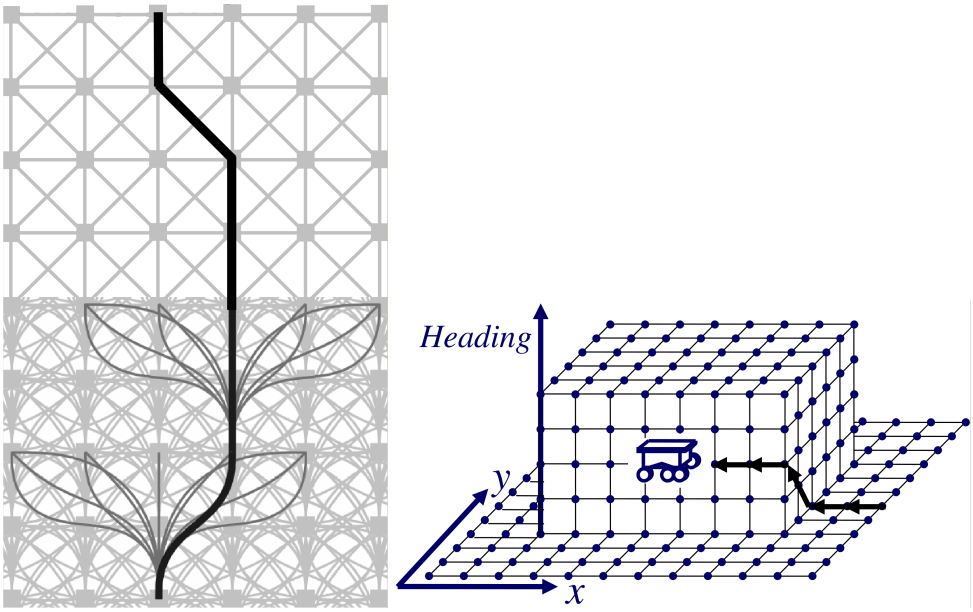


Figure 4.4.3: **Graduated fidelity example**. A simplified representation of two state lattices of different fidelities, seamlessly merged due to structure: a three-dimensional roadmap that includes position (x, y) variables and heading is allocated in the vicinity of the vehicle, and a lower-dimensional one, with just position variables, is further away.

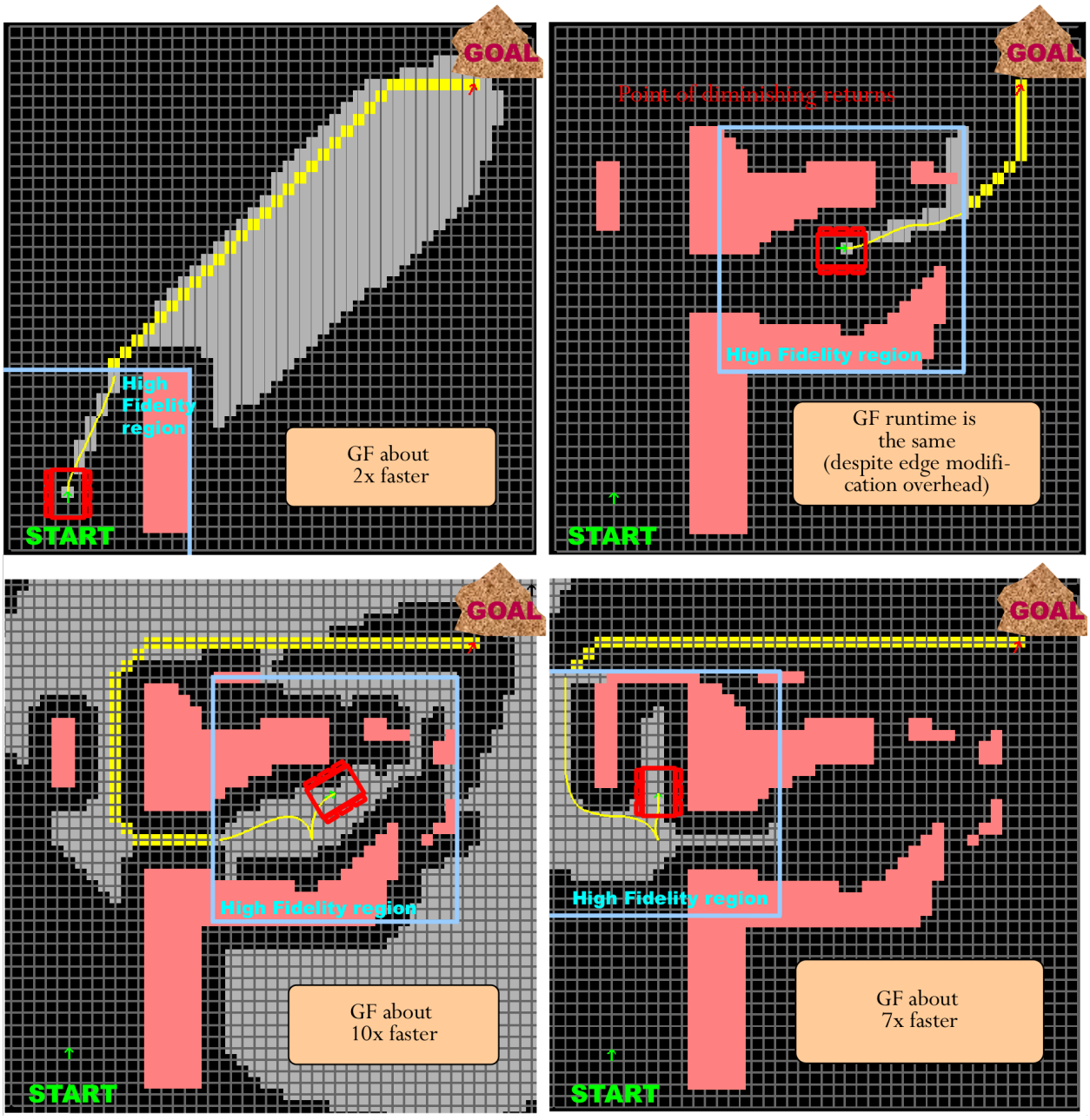


Figure 4.4.4: **Graduated fidelity runtime**. Illustration of the effects of graduated fidelity.

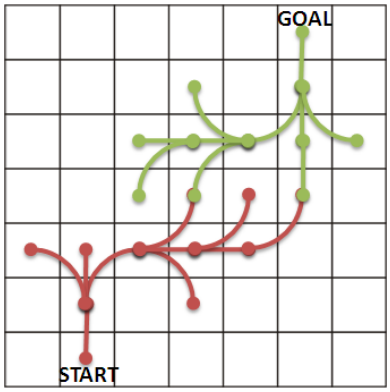
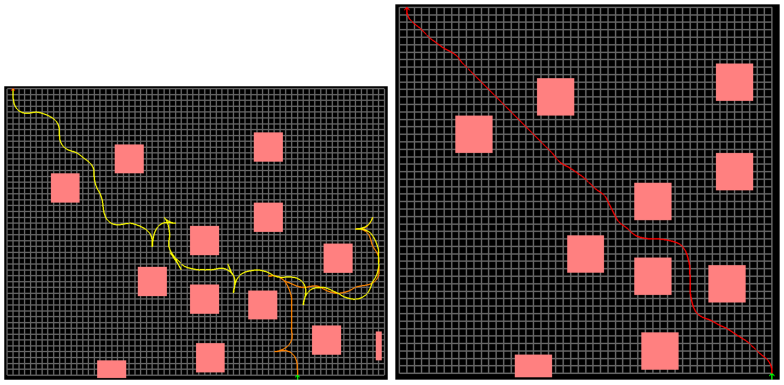
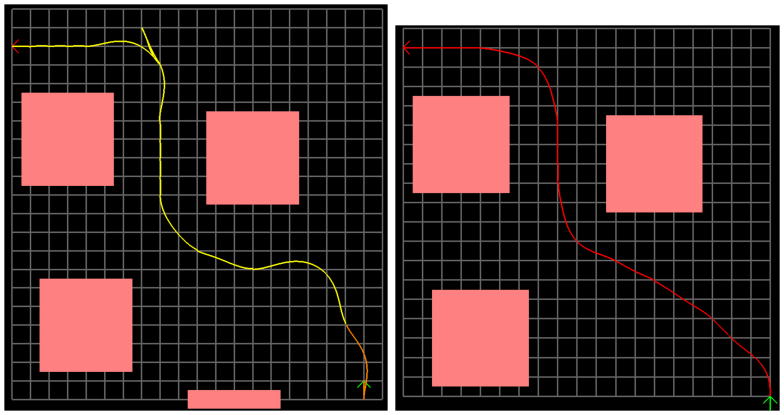


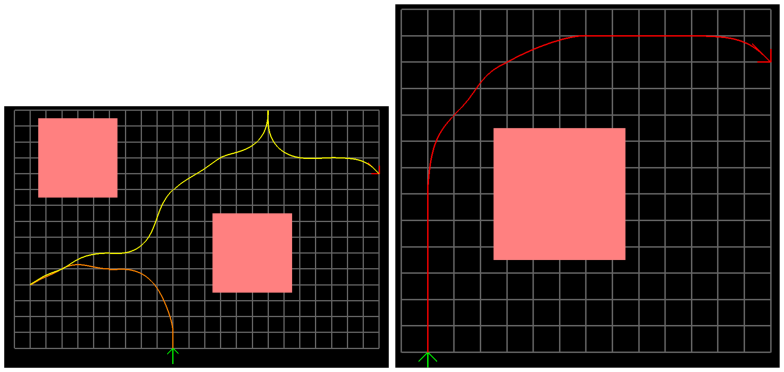
Figure 4.5.1: **State Lattice Bi-RRT**. The problem of gaps at the connections of the leaves of the two trees is eliminated with lattice primitives due to the regularity of endpoints in state space.



(a) Bi-RRT (b) A*



(c) Bi-RRT (d) A*



(e) Bi-RRT (f) A*

Figure 4.5.2: **Bi-RRT results**. Motion plans computed with Bi-RRT search (left) and A* (right). The performance results are presented in Table 4.2.

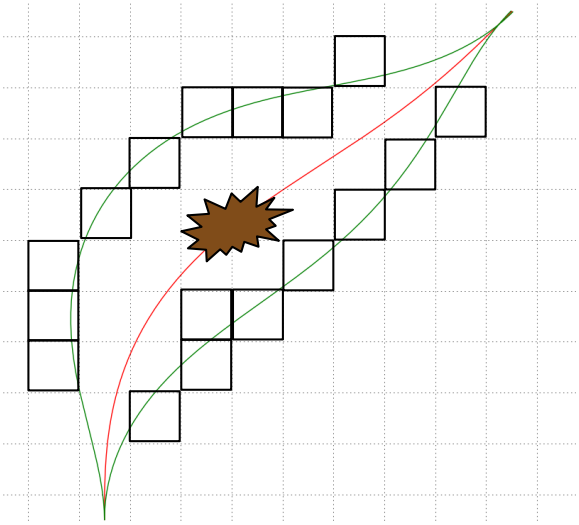


Figure 4.6.1: **On-demand expansion**. Pre-computed sets of primitives for on-demand expansion – a mechanism for managing the expressiveness-runtime trade-off. The red path represents a single path in the canonical set while the green paths represent the possible alternatives.

CHAPTER 5

SEARCH SPACE QUALITY MEASURE

As formulated in Section 1.2.2, the goal of this thesis is choosing a motion planner design that is attractive with respect to the specific challenges that contribute to most of the difficulty of the problem of motion plannning with differential constraints.

Towards that end, a specific search space type, the state lattice, was introduced. Having motivated the proposed type of motion primitives in Chapters 3 and 4, this Chapter begins the discussion on the principles of their design. Assuming a system model and an appropriate controller, the principles are organized in three categories: setting up metrics for comparing alternative search space designs in an effort to arrive at the optimal one with respect to the design criteria in Section 1.2.2 (this Chapter), developing the appropriate sampling rules in state space (Chapter 6) and, finally, designing a near-minimal set of state lattice primitives (Chapter 7) based on the chosen structure.

5.1 SEARCH SPACE DESIGN FORMULATION

Section 1.2.1 poses the motion planning problem as optimization in the space of motion alternatives that the system is capable of, the control space $\mathcal{U}$. This space is referred to as search space because it is the domain of optimization, often implemented as search. Due to the complexity of this problem, it is common practice to solve its approximate formulation, where the search space is a sampled subset of the control space, $\hat{\mathcal{U}}$. In the context of this thesis, the search space is the power set of the set of edges in the state lattice roadmap, $\hat{\mathcal{U}} = \mathcal{P}(\mathcal{E}) \setminus \emptyset$. By regular structure and position-invariance, $\mathcal{E}$ is the union of a repeating control set $\mathcal{E}_O$ at the origin and all its variants at the position offsets of each of the roadmap vertices. The cardinality of $\mathcal{E}$ is thus related to the number of vertices in the roadmap, and in general this set is countably infinite. The problem of designing a search space is hereby formulated as finding a control set $\mathcal{E}_O$ such that the resulting search space $\hat{\mathcal{U}}$ maximizes a certain utility function.

$$\hat{\mathcal{U}}^* = \underset{\hat{\mathcal{U}} \subset \mathcal{U}}{\operatorname{argmax}} \mathcal{F}(\hat{\mathcal{U}}) \tag{5.1.1}$$

where the utility function is a sum of the three objectives related to the specific challenges specified in Section 1.2.2: optimality, runtime and completeness:

$$\mathcal{F}(\hat{\mathcal{U}}) = \mathcal{F}_o(\hat{\mathcal{U}}) + \mathcal{F}_r(\hat{\mathcal{U}}) + \mathcal{F}_c(\hat{\mathcal{U}}) \tag{5.1.2}$$

The feasibility criterion in Section 1.2.2 does not appear in the objective function because all of the $\hat{\mathcal{U}}$ being considered are assumed to consist of exclusively feasible motions.

In the sequel, choosing a maximizer $\hat{\mathcal{U}}^*$ will be identified with choosing a corresponding control set $\mathcal{E}_O{}^*$, since one set directly influences the other. Also, notice that the maximization domain above is the entire control space of the system, which is difficult to explore. As suggested above, this thesis makes a simplifying assumption that the $\hat{\mathcal{U}}$ being considered belong to the state lattice type of discretized search spaces. Similar assumptions are common in this domain due to the difficulty of the original problem.

5.2 SEARCH SPACE PSEUDO-METRIC

While the formulation of search space design in the previous section is fairly straight-forward, it is difficult to apply it in a realistic setting, unfortunately. For example, it is not clear how to estimate the components of the objective function (5.1.2). Instead, it appears reasonable to re-frame the objective function to be a relative one that is based on comparing two given search spaces. It is more straight-forward to determine if one search space is more attractive than the other, based on relevant performance qualities. In fact, such comparisons have already been made in this thesis, e.g. during the discussion of the discriminators of the proposed type of search spaces with existing alternatives.

To this end, a search space *pseudo-metric* is proposed as the map $\tilde{\rho} : \hat{\mathcal{U}} \times \hat{\mathcal{U}} \to \mathbb{R}$. It is a *relative* objective function that measures "merit" of one search space with respect to the other. It is specifically designed not to be an actual metric because it is convenient, in this setting, to avoid making it positive definite. In fact, signed values of this function will gain special meaning: if the value is positive, then one of the arguments of the function (say, the first one) is a search space with a better score than the other one, and if the value is negative, then vice versa. In this manner, stochastic optimization techniques, such as simulated annealing or genetic algorithms, can be easily constructed to pursue and further improve the more promising of the two search space candidates. If, for example, one of the arguments to $\tilde{\rho}(\cdot)$ is the best candidate seen so far and the pseudo-metric value is positive, then the other is a promising new search space to consider further.

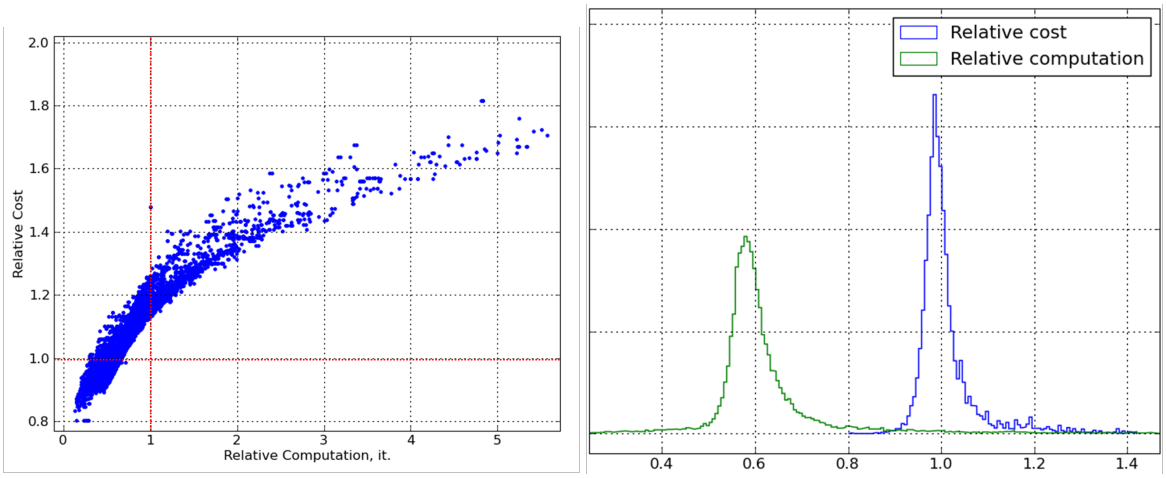
5.2.1 EMPIRICAL EVALUATION

A natural approach to evaluating $\tilde{\rho}(\cdot)$ is to base it on direct measurements of its performance as part of an actual motion planner. Since the present inquiry focuses on search space design, it is assumed that the choice of the search algorithm is determined by the application. Compared to other search spaces, it is a particularly good assumption for a state lattice search space because it is compatible with a large variety of algorithms used in planning.

Thus, given the two state lattice search space candidates, $\hat{\mathcal{U}}_1$ and $\hat{\mathcal{U}}_2$, being compared, a search algorithm of choice is paired with one of them. The resulting motion planner is used to compute a large number of planning queries, e.g. for all vertices in $\hat{\mathcal{V}}_R$ in a state space ball of radius R around

the roadmap origin, similar to the computation of the HLUT, as described in Section 4.1.2. *

For each planning query, executed with a motion planner developed in the above manner, we collect two scalars: the cost of the motion and the runtime (in terms of seconds or planner iterations); these correspond to the Optimality and the Runtime search space criteria we seek to optimize. To evaluate the Completeness criterion, we record the number of elements in $\hat{\mathcal{V}}_R$ for which the planner was able to find collision-free trajectories. This represents a measurement of *coverage*, an *estimate* of the Completeness property via direct measurement of the planner's performance. Even though the cost and runtime measurements in free space are meaningful, the coverage measurement is not (recall that all state lattice vertices are reachable by construction) and must be carried out under environment constraints. A Monte Carlo evaluation with a range of random worlds is one way to obtain this measurement. In conclusion, Feasibility criterion is not evaluated because it is assumed to be satisfied in this setting by construction.



(a) Raw ratio data (b) Ratio histogram

Figure 5.2.1: **Empirical ratio metrics**. The figure compares two different state lattice search spaces and indicates that, on average, one of the search spaces was as much as 40% more efficient at computing approximately the same quality of solutions (cost ratio peaks at 1.0).

*Choosing a finite radius in this manner is justified by the third assumption in Section 1.2.3. The value of this radius is chosen as large as possible given the available computation capacity. Note that a search space developed with a finite R still can be extended indefinitely via repeated application of the control set, so the finite reachability tree used for search space design does not sacrifice the global nature of the resulting motion planner. The sacrifice here is in terms of the *quality* of the representation of the reachability, hence the desire to choose R as large as possible.

Once the three quantities above are computed for a set of motion planning queries, the same queries are repeated with the second planner, based on the alternative search space, $\hat{\mathcal{U}}_2$. Once the process is completed, two ratios for each planning query (Optimality and Runtime) and one ratio for the entire set of queries (Completeness) are computed. Performing statistical analysis on these data reveals the relative performance of $\hat{\mathcal{U}}_1$ with respect to $\hat{\mathcal{U}}_2$. It is used as the basis of computing $\tilde{\rho}(\cdot)$. For example, Figures 5.2.1 a) and b) plot the data in raw and histogram form, respectively. The histogram reveals that, among the two different state lattice search spaces compared, on average, one of the search spaces was as much as 40% more efficient at computing approximately the same quality of solutions (cost ratio peaks at 1.0). Similarly, coverage results can be represented as a histogram over $\hat{\mathcal{V}}_R$, a subspace of X that was used in the evaluation. Figure 8.5.3 shows an example of such a histogram: since the domain of the histogram is 2D, pseudo-color is utilized to visualize the coverage in terms of the number of different (discrete) heading values that are reachable at each (x, y) cell.

Certainly, there are certain drawbacks to a purely empirical method of comparing search spaces, most notably the involved computation effort, however it is beneficial that the method directly measures the qualities that matter in applying the resulting motion planner.

At the outset, the goal was to compute a scalar value to represent the comparison of two search space candidates. At this point, we have obtained histograms for the three quantities of interest. Any method of reducing a datapoint distribution in such histograms to a scalar may be used. One example is Kullback-Leibler divergence; an even simpler method is a sum of histogram bar heights weighted by the value the bar represents. As Example 5.2.1 illustrates, by formulating the horizontal axis, e.g. in Figure 5.2.1b) on a *log* scale, we obtain the signed pseudo-metric that was motivated above: swapping the arguments to $\tilde{\rho}(\cdot)$ negates its value.

5.2.2 Quality Measure

The set of vertices in the roadmap $\hat{\mathcal{V}}$ is, in general, countably infinite. The regular structure of the state lattice makes it possible to enumerate any R-bounded subset $\hat{\mathcal{V}}_R \subset \hat{\mathcal{V}}$. Similarly, the power set of edges in the roadmap, $\hat{\mathcal{U}}$, defined in Section 5.1, is similarly countably infinite, and its R-bounded subset $\hat{\mathcal{E}}_R \subset \hat{\mathcal{E}}$ is finite, albeit very large. Nevertheless, $\hat{\mathcal{E}}_R$ is an important construct – it is the most expressive set of motion alternatives in the R-ball that the state lattice is capable of

representing.

As the reader may recall, an objective function $\mathcal{F}(\hat{\mathcal{U}})$ over search spaces was introduced earlier in this Chapter, and it played a key role in formulating search space design as an optimization problem (5.1.2). It acts as a *measure* in that it estimates the quality of the given search space. The pseudo-metric in Section 5.2 can be transformed into this measure using $\hat{\mathcal{E}}$ reviewed above:

$$\mathcal{F}(\hat{\mathcal{U}}) = \tilde{\rho}(\hat{\mathcal{U}}, \hat{\mathcal{E}}) \tag{5.2.1}$$

In practical computation, R-ball variants of $\hat{\mathcal{U}}$ and $\hat{\mathcal{E}}$ would be utilized, resulting in an approximate estimate of this quality measure.

Example 5.2.1 (Search Space Quality Comparison)**.** *In this example, the empirical formulation of the search space pseudo-metric (Section 5.2.1) is applied to the quality comparison of two search spaces: a state lattice and a Barraquand-Latombe search space [11]. As elsewhere in this thesis, a car-like system model is used for its simplicity (1.2.2).*

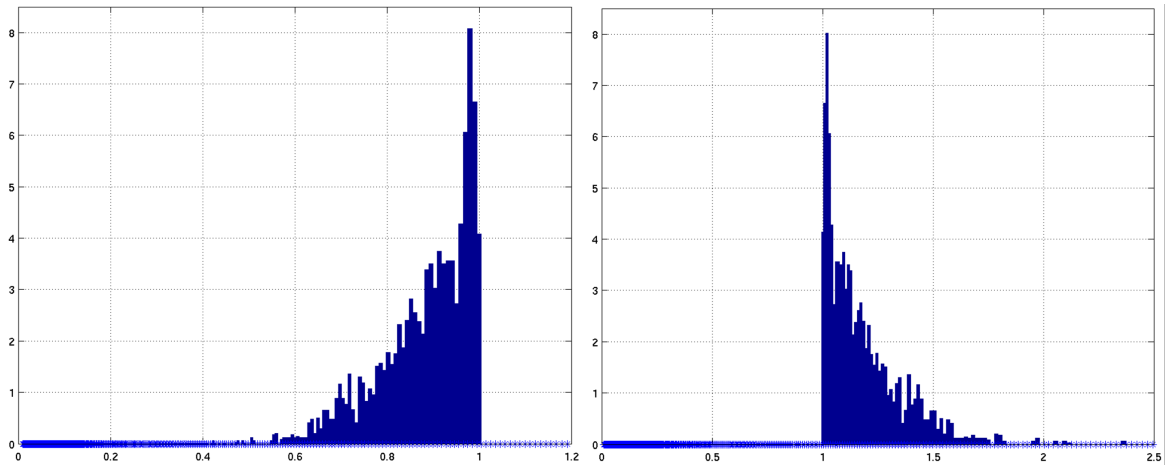
Barraquand-Latombe search space, $\hat{\mathcal{U}}_{BL}$, includes three motions emanating from any state: a left and right turn at the maximum value of steering angle and a straight motion, for a certain duration Δt. The duration is set in such a way as to ensure that each of the motions escapes the state cell associated with the vehicle's current state. State sampling is three-dimensional, including 2D position and orientation. Position sampling cell size was set to 0.2 *while the vehicle minimum turning radius was set to* 1.0*. Heading dimension was discretized into* 16 *values. Note that concatenations of such primitives results in discontinuous curvature across connections – the robot would have to stop at each transition point between the primitives in order to re-orient its wheels.*

State lattice search space, $\hat{\mathcal{U}}_{SL}$, was designed to match the above search space as closely as possible: the same position and heading sampling was used. However, the available controller for constructing state lattice motion primitives allowed setting steering angle at trajectory endpoints [63] – this capacity was utilized in order to allow continuity of curvature across nodes. The steering angle was set to 0 at the transitions between motion primitives.

The empirical pseudo-metric was evaluated on the two search spaces by computing planning queries for every element of $\hat{\mathcal{V}}_R$, using a large state space region, $R = 20$, in order to capture the maneuverability of the car-like robot as fully as possible.

Figure 5.2.2 shows the results of comparing the cost of computed motions: the state lattice search space results in about 14% greater path length, largely due to the imposed constraint that curvature is continuous across transition points. In addition, the figure demonstrates the effect of representing the cost ratio data as histograms, generated against the horizontal axis in log *scale: swapping the order of the arguments to* $\tilde{\rho}$ *reflects the histogram across the vertical axis (compare Figure 5.2.2 a). and b).), and negates the value of the pseudo-metric:*

$$\tilde{\rho}(\hat{\mathcal{U}}_{BL}, \hat{\mathcal{U}}_{SL}) = -\tilde{\rho}(\hat{\mathcal{U}}_{SL}, \hat{\mathcal{U}}_{BL}) \tag{5.2.2}$$



(a) $\tilde{\rho}(\hat{\mathcal{U}}_{BL}, \hat{\mathcal{U}}_{SL}) = -0.142$ (b) $\tilde{\rho}(\hat{\mathcal{U}}_{SL}, \hat{\mathcal{U}}_{BL}) = 0.142$

Figure 5.2.2: **Barraquand-Latombe vs. state lattice cost ratios**. The histogram demonstrates a comparison of the cost ratios (Optimality criterion) of two search spaces: Barraquand-Latombe [11] and the state lattice. Both search spaces were configured as similar as possible, however Barraquand-Latombe does not allow continuity of steering angle across motion primitive transitions, while state lattice does, which results in a 14% path length advantage of the former w.r.t. the latter. The histogram method, when expressed on the horizontal *log* scale, allows inverting the pseudo-metric, which results in reflecting the histogram across the vertical axis and negating its residual value.

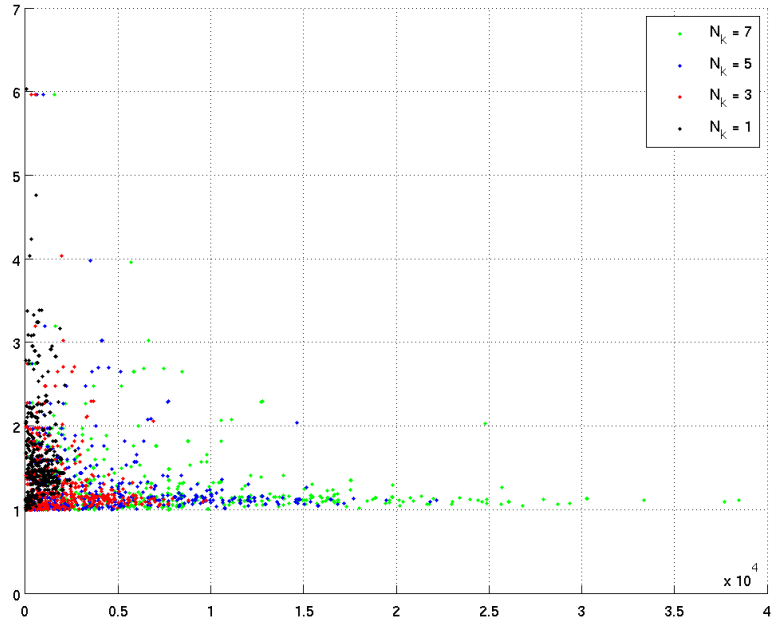
An alternative method of visualizing search pseudo-metric data is presented in the Figures 5.2.3 and 5.2.4. Here, cost ratios are plotted against runtime ratios. Hyperbolic shape of the plots is due to the majority of the motion planning queries with the two search spaces having similar

relative cost and runtime (i.e., being close to the point $(1,1)$ *in the plots).*

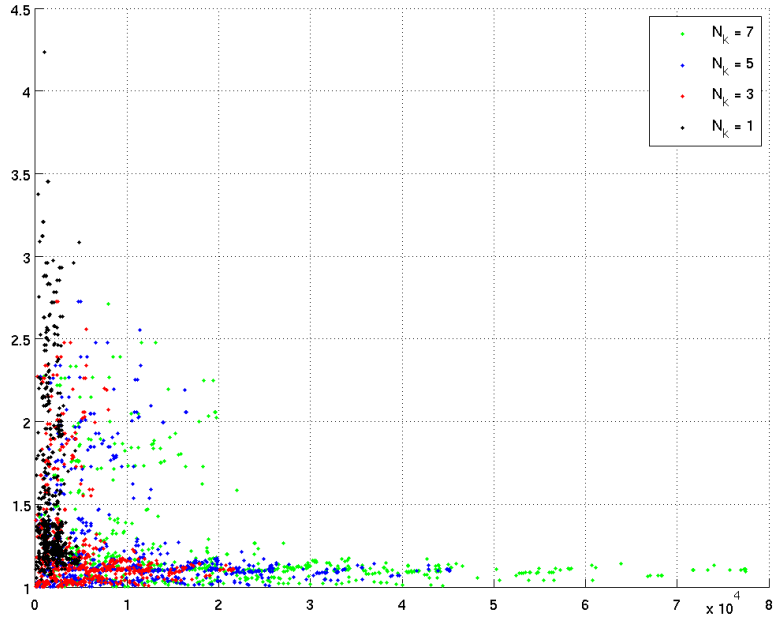
The data in these plots are from the experiments that are very similar to the one represented in Figure 5.2.2, which was repeated at four different resolutions of heading discretization: values $N_\theta = 8, 16, 32, 80$*, and four different resolutions of steering angle, values* $N_\kappa = 1, 3, 5, 7$ *(color-coded in the Figures). An example of the search space with* $N_\theta = 8$ *and* $N_\kappa = 1$ *is shown in Figure 3.2.2. These plots are helpful in exploring the* resolution space *of state lattice designs, the space of alternative state lattice designs by varying the sampling resolution of the state space dimensions, in this case the resolution of the heading and steering angle dimensions for the two search spaces being evaluated.*

□

This chapter began the discussion on the principles of the design of state lattice motion primitives by setting up the metrics for comparing alternative search space designs in an effort to arrive at the optimal one with respect to the design criteria in Section 1.2.2. A data-driven approach was proposed for establishing a fair comparison of two alternative state lattice search space designs in a way that is meaningful with respect to the design criteria.

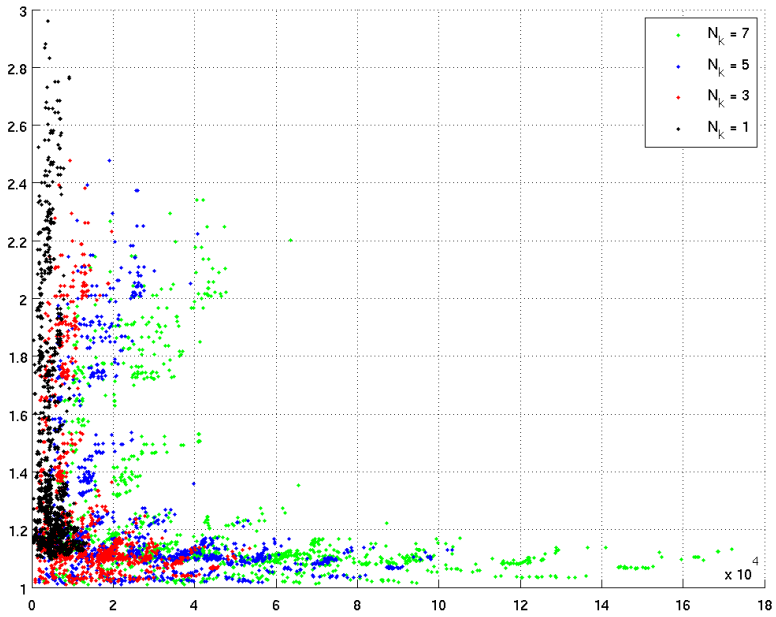


(a) $N_\theta = 8$

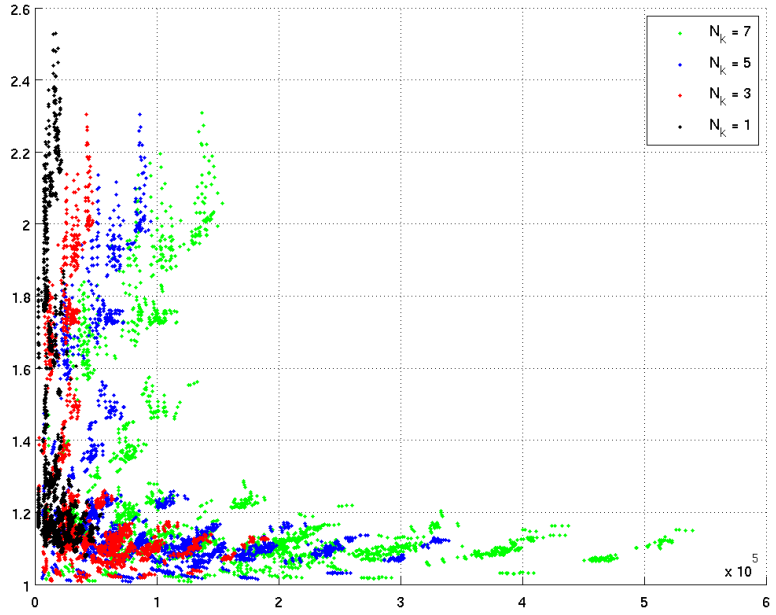


(b) $N_\theta = 16$

Figure 5.2.3: **Resolution-space analysis**. The resolution space of heading and steering angle dimensions, in the context of the search space design Example 5.2.1, is explored here. The plots represent cost vs. computation ratios plotted against each other.



(a) $N_\theta = 32$



(b) $N_\theta = 80$

Figure 5.2.4: **Resolution-space analysis (cont.)**. The resolution space of heading and steering angle dimensions is continued here for heading discretizations with 32 and 80 values.

CHAPTER 6

STATE SPACE SAMPLING

The state lattice motion primitives, proposed in this thesis, have been motivated in Chapters 3 and 4 and found to be attractive with respect to addressing the principal challenges of motion planning with differential constraints (Section 1.2.2). Chapter 5 began an inquiry into the design principles for such motion primitives by formulating the search space design as an optimization problem and providing a utility function to be maximized in this process. The motivation for such a formulation was to enable the design of motion primitives to be governed by an algorithm that would lend itself well to being encoded as a computer program, leading to automated search space design. As suggested in Section 1.4, automated design of motion primitives is one of the contributions of this thesis.

Now that the framework for search space design was fixed in Chapter 5, this Chapter begins the presentation of the design principles for constructing candidate search spaces to be evaluated and optimized in the above framework.

6.1 DIMENSIONALITY REDUCTION

In general, the problem of selecting the minimal number of dimensions that adequately represent the planning problem is quite challenging. In the case of designing lattice primitives, the chosen BVP solver would typically fix the choice of dimensions. If solvers for the same problem are available at different dimensionalities, an iterative dimensionality reduction process may be undertaken as part of the automated search space design procedure. In particular, once a set of lattice primitives is designed at the highest dimensionality, it may be repeated with one of the dimensions removed. The process iterates until the loss of representation quality exceeds desired application tolerances. This straight-forward iterative process, though computationally intensive, would be able to reduce the dimensionality of representation for the given system.

6.2 DESIGN PRINCIPLES

Once state dimensionality is fixed, the design principles for choosing a good state sampling rule in state lattice context are given here. The principles are purposefully broad: each application imposes specific requirements on state sampling. As suggested in Section 1.2.2, no optimal design is attempted here because the basis for this optimality is dependent on a particular application. Instead, the presented principles attempt to arrive at a design that addresses the challenges, outlined in that section, as well as possible.

Design Principle 6.2.1 (Workspace sampling)**.** *The sampling of workspace dimensions (2D or 3D position) is dictated by the problem specification. Some of the criteria that significantly influence it include:*

- *Robot's perception resolution*
- *Vehicle footprint*
- *Control accuracy*

These criteria are to be adhered to during motion planner design since none of these criteria are design freedoms.

□

In other words, when discretizing position variables, it is important to take into account the specifics of the vehicle. For example, it is generally helpful to match the resolution of planner's position discretization to robot's perception resolution. Making the former much finer would be somewhat inefficient, since the collision detection and cost evaluation is only able to function at the perception resolution. Similarly, planning maneuvers at a much higher resolution than the vehicle footprint and control accuracy may be wasteful since the vehicle is not capable of executing such intricate maneuvers in the first place.

Once position sampling is set, the sampling rules for other dimensions are derived from it. The following principles are designed to guide the development of these sampling rules.

Design Principle 6.2.2 (Incremental sampling)**.** *In an effort to minimize the sampling density, a design with the most coarse sampling rule is to be attempted first. The resolution is to be increased only if the resulting search space is determined to require it.*

□

This principle is related to incremental dimensionality reduction described in Section 6.1. Fortunately, due to the automatic nature of the motion primitive design, the incremental sampling procedure can be performed with minimal manual effort.

Certain dimensions that are frequently used in modeling the mobility of robots have a finite range, for example steering angle, velocity, acceleration, etc. The following design principle suggests the minimal sampling of these dimensions.

Design Principle 6.2.3 (Extremal values)**.** *The minimal sampling of a dimension involves two values: the extremals of the range of the dimension.*

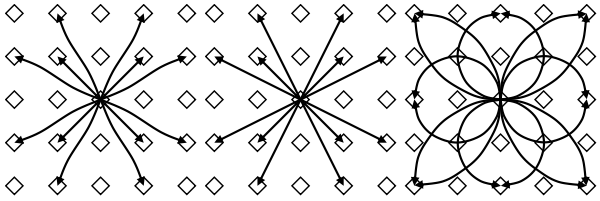
□

This principle is similar to the *bang-bang* approach in optimal control [126]. It also appears frequently in motion planning literature [11; 28; 30; 115].

6.3 Natural Sampling Rule

Among similarly performing search spaces, those with more coarse sampling are preferred. This is an Occam's razor statement: a simpler approach is likely to lead to a solution that is easier to

develop, test and maintain. If there are motions that require the least amount of control effort for the system to follow, then these motions are somehow *natural* for it. For example, a car-like robot moving on a flat surface in a straight line would typically require a low controller effort, and so a straight line – constant heading – is a natural trajectory for this system. Similarly, moving along an arc – maintaining constant trajectory curvature – is also a natural motion, since the steering angle joint once again does not need to be actuated (ignoring trajectory following issues for a moment). Examples of these natural motions are shown in Figure 6.3.1.



(a) Ad-hoc state sampling (b) Natural heading (c) Natural curvature

Figure 6.3.1: **Motivating natural sampling rule**. Part a): ad-hoc state sampling may result in overly complex controls. Straight lines b) and arcs c) are natural for a car-like system because they do not require actuation of the steering angle.

Since controls are induced by state sampling in this setting, it is beneficial to choose state samples that reduce the cost of controls, e.g. lead to a greater number of low-cost, natural motions.

Design Principle 6.3.1 (Natural sampling rule)**.** *A beneficial sampling rule is the one that maximizes the occurrence of natural motions in the resulting control set.*

□

One of the methods of applying this principle is as follows. Given the position sampling, as specified by the Principle 6.2.1 and the rudimentary sampling of other dimensions, per Principle 6.2.3, the resulting search space is evaluated. Assuming it is desired to increase the sampling further, a list of the dimensions to undergo this increase is sorted in terms of derivatives: for example, car-like steering angle is related to curvature of its trajectory, and therefore it is processed after heading; similarly acceleration is processed after velocity. With this ordering, the dimension undergoing sampling density increase is taken to be a freedom, and the reachability of the system is generated with an emphasis on using simplest, natural motions. Finally, statistics are collected

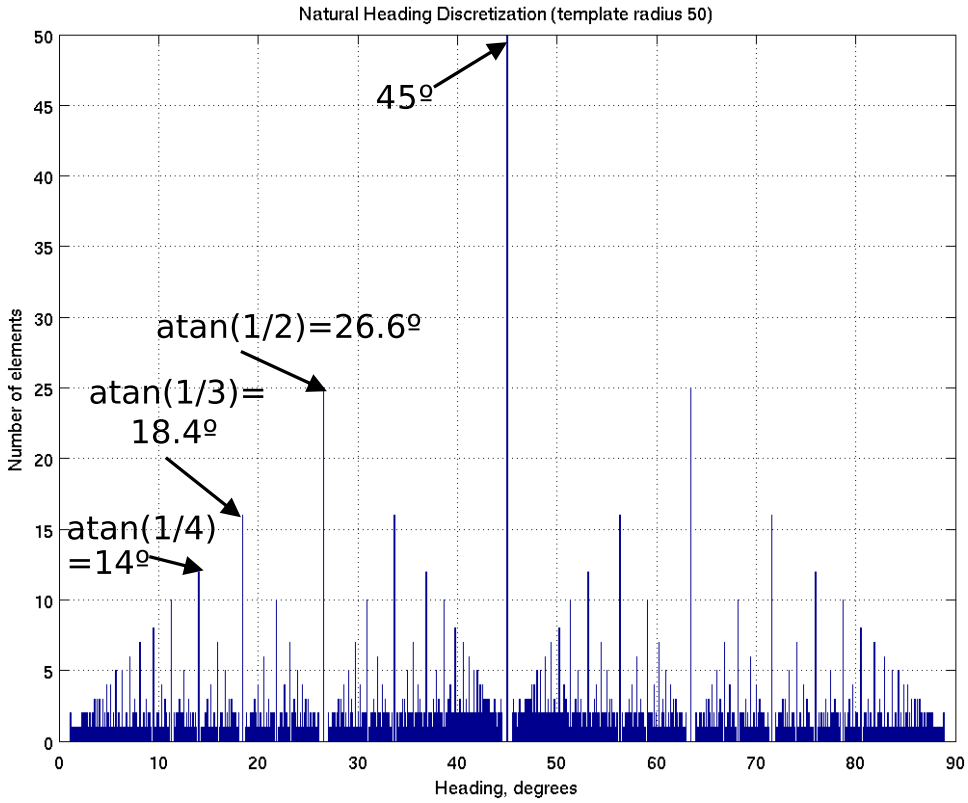


Figure 6.3.2: **Natural heading discretization**. The heading values that correspond to spikes in the histogram are shared by a greater number of natural motions in this example, and therefore are good candidates for inclusion in sampling the heading dimension, in the order of their frequency.

on the occurrence of the values in the dimension in question; most frequently occurring values are good candidates to serve as new samples of that dimension. Example 6.3.1 illustrates this procedure.

Example 6.3.1 (Natural sampling rule)**.** *As most of the examples in this thesis, the example uses the car-like system model. Among the well-known systems with differential constraints, it is one of the simplest mathematically and quite intuitive.*

As described above, position sampling is chosen first. For simplicity, a regular grid in 2D (x,y)-space is chosen. According to (1.2.2), the remaining state dimensions to sample are heading and steering angle.

First, heading sampling is chosen using the procedure above: keeping this dimension a freedom, the position samples were connected with most natural motions, straight lines (no differential constraints are considered since heading is free, and curvature is being ignored by dimension ordering). This process is similar to Figure 6.3.1b). The procedure is applied to position samples in a 2D ball around the origin that is large enough to capture the reachability of the system. After all of the line motions were generated, the statistics on the values of their slopes (world-frame heading) are computed in order to obtain the values that occur most often. Figure 6.3.2 shows a histogram of these values. The peaks of the histogram indicate that, in the open interval $(0^\circ, 90^\circ)$*, the most frequently occurring values of heading are* arctan *of the ratios* $\frac{1}{1}$, $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$*, etc., corresponding to* 45°, 26.565°, 18.435°, 14.036°*, etc. This observation makes sense geometrically; in fact, all values in the histogram must be* arctan *of integer ratios. The ones that correspond to spikes in the histogram are the most frequent ones, and therefore are good candidates for inclusion in sampling this dimension, in the order of their frequency. Selecting the desired number of the heading values (as determined by testing the resultant search space in relevant planning scenarios) concludes sampling the heading dimension.*

Once heading dimension is sampled, the process is repeated for the steering angle dimension. Once again, a set of points, now three-dimensional (x, y, θ) *since heading* θ *is included, in a region around the origin is chosen. For every "oriented" point in this set, a constant-curvature arc is computed, and the statistics of the resulting curvatures is once again analyzed with a histogram, shown in Figure 6.3.3.*

The following three figures present alternative ways of visualizing the occurrence frequency of arc curvature values. Figure 6.3.4 presents a similar curvature histogram as before, but it color-codes the contribution of the arcs at various distances from the vehicle. This reveals, perhaps unsurprisingly, that the arcs in the immediate vicinity of the vehicle contribute most of the higher values of curvature (including the maximum curvature, the value that corresponds to the minimum turning radius of the vehicle).

Figure 6.3.5 sorts the curvature and arc-length values of the arcs with respect to each other. The most frequent values of either quantity manifest themselves as the longest horizontal line segments in the figure.

Lastly, a 2D pseudo-color plot in Figure 6.3.6 captures both the relationship of curvature to arc-length, while using the color to represent frequency of each value. It confirms a peak at

the curvature value 0.4581 *that was also noted in Figure 6.3.3. However, this Figure reveals that this curvature value is not only abundant in 8 different arc-lengths, but one curvature-length combination is especially dominant. It appears to be a good curvature sample to keep because it also has a relatively low arc length.*

□

In conclusion, it is noted that the sequence of design principles presented in this chapter represents a top-down design methodology. It also first caters to the variables that are most constrained by the other aspects of robot design, such as perception and controller characteristics, and leaves the quantities with the greatest flexibility to be settled last.

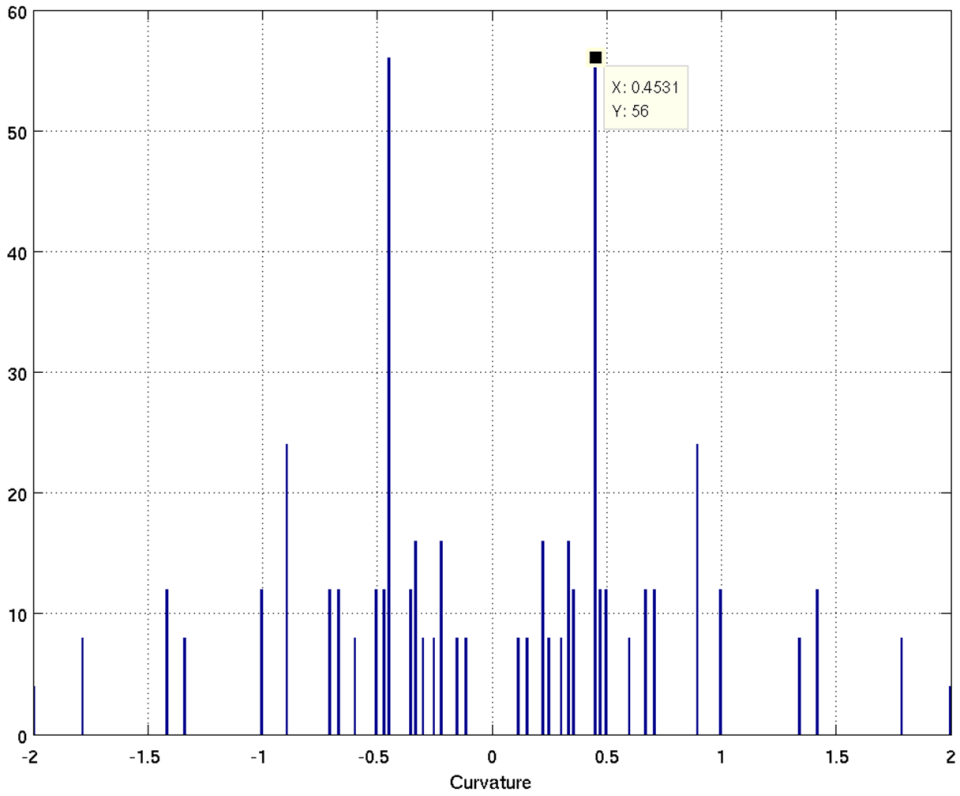


Figure 6.3.3: **Arc curvature histogram**. In determining steering angle discretization for the car-like system example, a constant-curvature arc is computed for each (x, y, θ)-point in a region around the vehicle. The curvatures of the resulting arcs are plotted in this histogram. The expected symmetry is apparent. Similar to Figure 6.3.2, the values of steering angle discretization are chosen from the peaks of the histogram, in descending order.

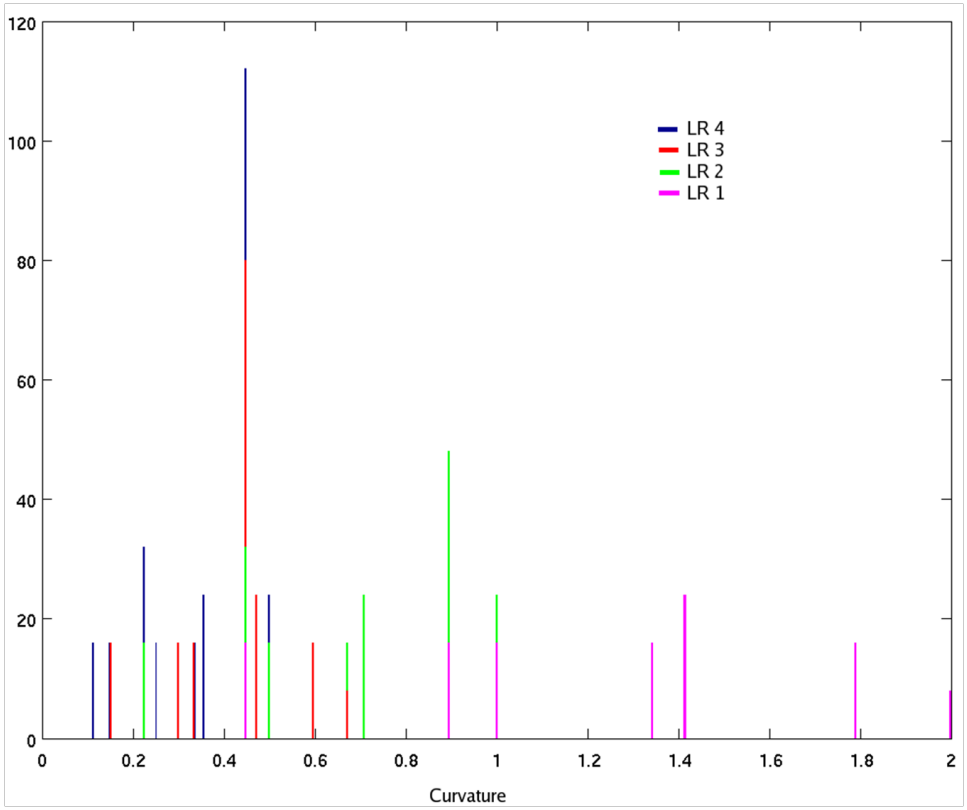
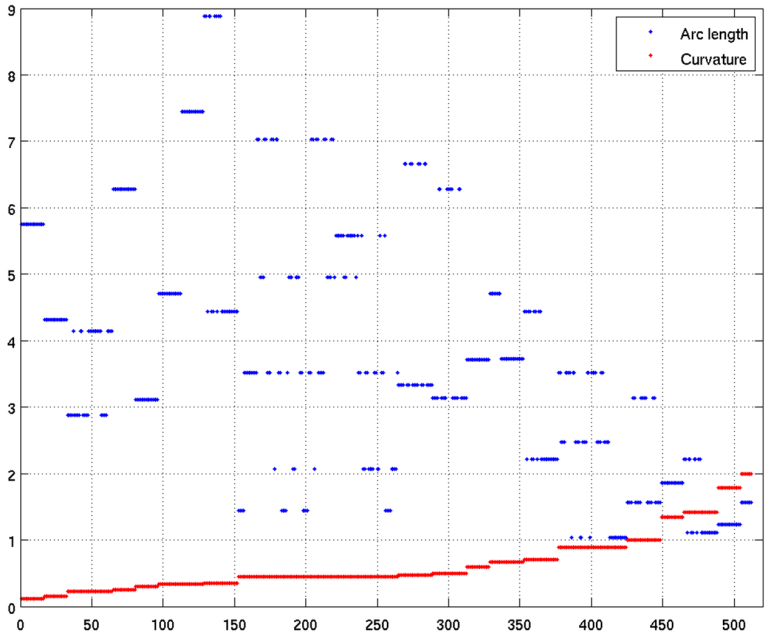
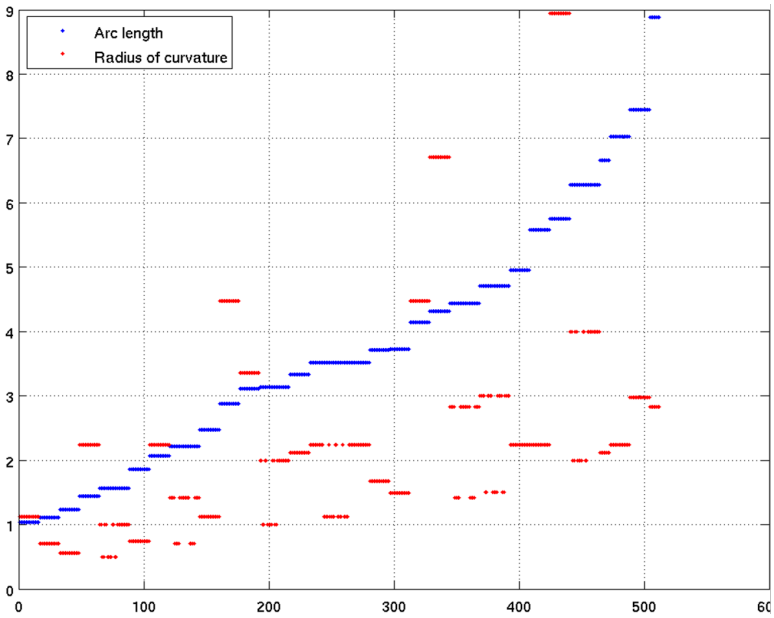


Figure 6.3.4: **Distance-sorted curvatures**. The histogram shows the number of constant-curvature trajectories for each curvature value. Curvature values are sparse because the study in this example only computes arcs connecting lattice nodes exactly. The vicinity of the connected nodes is represented by Lattice Radius (LR), equivalent to L_1-norm in 2D. Note that arcs from origin to 8 immediate neighbor nodes (magenta bars) provide all curvature values above 50% of max. curvature.



(a) Curvature sort



(b) Arc-length sort

Figure 6.3.5: **Arc length vs. curvature**. The curvature and arc-length values of the arcs are sorted with respect to each other. The most frequent values of either quantity manifest themselves as the longest horizontal line segments in the figure.

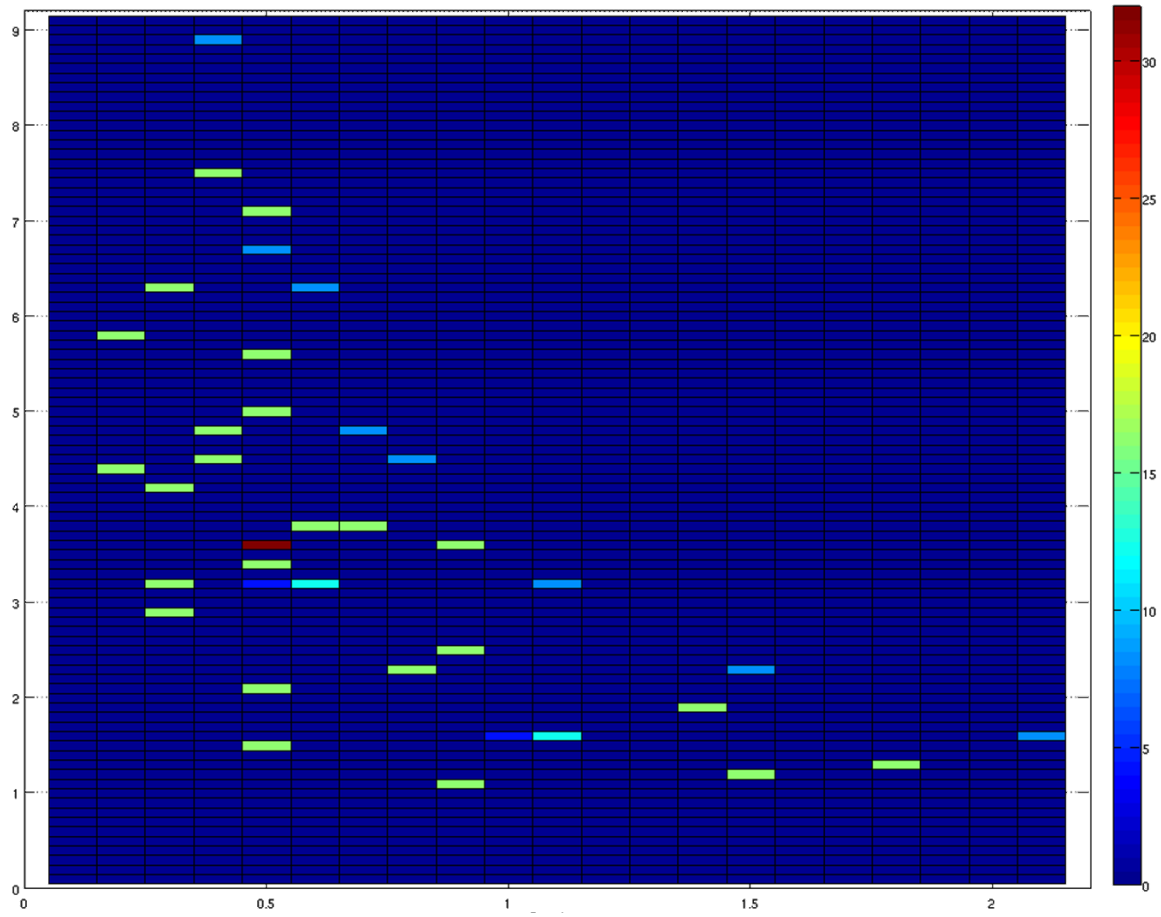


Figure 6.3.6: **Arc-length pseudo-color**. The red cell in the histogram indicates the most frequent arc curvature-length combination. It corresponds to the peak in the 1D histogram in Figure 6.3.3. Besides its high frequency of occurrence, this value of curvature appears to be a good sample to keep in the discretization because it corresponds to a relatively low arc length.

CHAPTER 7

CONTROL SET GENERATION

Previous chapters have set up the context of motion planning with differential constraints using state lattice motion primitives. Chapter 5 began the discussion of automatic generation of these motion primitives by formulating the problem of search space design as optimization and setting up methods to estimate the measure of search space quality, the objective function in this optimization. With that structure in place, Chapter 6 began actual design of search spaces by specifying design principles for choosing good state sampling. Since, in state lattice setting, that sampling induces the sampling of controls, the resulting search spaces can already be used in planning, although they suffer from excessively large size and oversampling. This Chapter concludes the discussion on automatic search space design by providing principles for reducing this oversampling down to a manageable level, while retaining guarantees on quality of the resulting search space.

7.1 Approximate Off-line Controller via Reachability Analysis

As was suggested above, given state sampling rule, the state lattice is constructed by running a controller to compute controls to steer the system between all the pairs of vertices in the state lattice roadmap. It may be a challenge to construct such a controller for some systems, especially those with complex kinematics and dynamics. Fortunately, this controller need not execute in real time on the system. This section suggests one such controller design that may require extensive computation, yet is quite general in its applicability.

The suggestion here is based on conventional forward-simulation control sampling techniques [11; 46; 60]. First, a dense reachability tree of the system was generated using high-resolution, regular sampling in control space. A state space reachability pruning technique, as in [11] is utilized to maintain computation at manageable level. Since the state resolution of this pruning is a multiple of the desired state sampling resolution, the endpoints of the primitives can be selected to be close to their respective state cell centers. Those that are not sufficiently close are improved via gradient descent optimization by treating the durations of control space samples, comprising the trajectory, as variables and the distance of the end-point to cell center as the objective. This may be a significant computation, since gradient estimation involves repeated execution of the physics simulation. The experimental results presented in Section 8.4 utilized this technique, which led to the automated construction of an efficient search space for a complex dynamics system.

7.2 Generation Approaches

With a choice of state sampling and a method of generating steering functions between state samples, the capacity to generate roadmap edges is obtained. The concept of a search space as a collection of all feasible connections between roadmaps, introduced as a theoretical concept, $\hat{\mathcal{U}}$, in Chapter 5, is made concrete here. In applications, a finite subset is considered, so it is understood to be a set of all feasible roadmap edges in a state space R-ball. Radius R is chosen to be as large as possible given the capacity of the computing hardware in order to obtain the best possible representation of vehicle mobility.

A search space generated in this manner is a complete roadmap design and is capable of addressing motion planning queries. However, the corresponding $\hat{\mathcal{U}}$ is generated via a computa-

tionally extensive off-line process, and its cardinality may be prohibitively large for fast, on-line planning. With this motivation, we embark on a concluding discussion in this thesis on reducing this cardinality while minimizing loss of search space quality.

To this end, two types of control set generation methods are proposed: *progression* and *regression*. These methods are further described below.

7.2.1 PROGRESSION METHODS

This type of search space generation forms a progression of computing new edges and adding them to the control set being developed. Such control set generation methods start out with an empty set. Motion primitives are generated and added to the control set until it reaches the desired size.

An example of this method is a technique referred to as *shortest edges*, presented in Algorithm 7.1. The benefit of this method is its relative simplicity: unlike the regression type, it can be developed without elaborating and analyzing large datasets of state samples and motion primitives. However, that comes at a cost of an inability to provide any quality guarantees on generated search spaces. It is still possible to apply the methods of Chapter 5 to validate control sets for motion planning and to ascertain that they would produce reasonable results. However, no statements or guarantees can be made with respect to the optimization formulation in (5.1.1) because $\mathcal{F}(\hat{\mathcal{U}})$ is never being evaluated.

Input: representation radius, R
Output: approximating control set E_a
$E_a = \emptyset$;
foreach $r \in [0, R-1]$ **do**
  foreach $x_i \in X_{r+1} \setminus X_r$, *where* X_r *is an R-ball in* X **do**
    generate $u(t)$ s.t. $u(t_I) = O$, $u(t_F) = x_i$;
    $E_a = E_a \cup u(t)$;
  end
end

Algorithm 7.1: Shortest Edges progression

7.2.2 REGRESSION METHODS

The regression methods are significantly more involved, but they are preferred because they systematically attempt to retain as much representation quality as is possible within the state lattice framework. These methods start out from a fully connected roadmap and attempt to detect and prune away the edges that are redundant or otherwise do not significantly contribute to the quality of representation. Justification for such methods is given below, and a design succession of three specific algorithms is presented in the following sections, culminating in the algorithm, in Section 7.4.2, that was the basis of most of the results given in Chapter 8.

By the given lattice state sampling rule, the set $\hat{V}_l$ is known. Theorem 7.2.1 develops a primitive set E_O that generates $\hat{E}_l$, a superset of $\hat{U}_l$, when used as the Dijkstra's vertex expansion; free space is assumed below, unless otherwise noted. More precisely, E_O is a set of primitive sets defined for all possible trajectory initial states in $\hat{V}_l$, up to the invariant dimensions (e.g. position). For example, different values of heading in Figure 3.2.2 would require different vertex expansions; E_O can be viewed as the union of the corresponding primitive sets.

Theorem 7.2.1. *Suppose origin vertices $O \subset \hat{V}_l$ are chosen (up to invariant dimensions). For every vertex $v_i \in \hat{V}_l$, an edge from each element of O to v_i is computed using the BVP solver and added to E_O (initially empty). When used as the Dijkstra's vertex expansion, E_O will search (equivalently, generate) a $\hat{G}_l$ such that $\hat{E}_l \supseteq \hat{U}_l$.*

Proof. First, by construction, we conclude that E_O contains $\hat{V}_l$ as endpoints of its primitives. Using E_O as the Dijkstra's vertex expansion amounts to replicating its edges at every $v_i \in \hat{V}_l$. If, per connectivity of $\hat{U}_l$, a certain v_i connects to a set of vertices $\{v_j\}$, then E_O, when replicated at v_i, will connect it to at least the same vertices, since $\{v_j\} \subseteq \hat{V}_l$. Thus, the process of replicating E_O at $v_i \in \hat{V}_l$ generates at least the edges present in $\hat{U}_l$, and therefore the induced set of edges $\hat{E}_l \supseteq \hat{U}_l$. □

Using E_O as the vertex expansion represents an extreme of quality-complexity trade-off. The cost of the capacity to explore at least all of $\hat{U}_l$ during search is a very large branching factor, $|E_O|$. Next we discuss an approach to manage this trade-off by computing an approximation primitive set $E_a \subset E_O$ of much smaller cardinality, while guaranteeing bounded suboptimality of computable motions w.r.t. $\hat{U}_l$ in terms of arbitrary notion of cost.

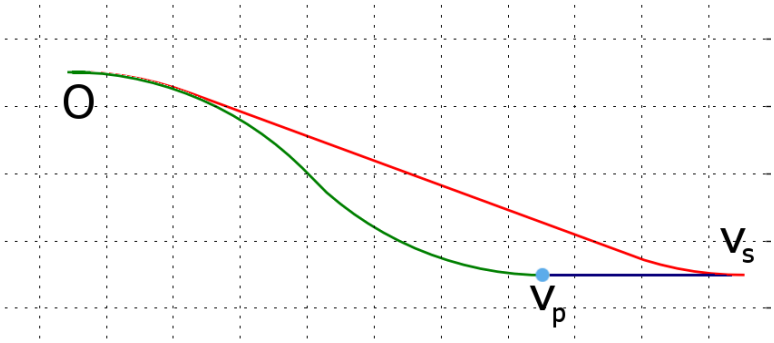


Figure 7.2.1: **Motion decomposition**. A motion is approximated by a concatenation of two shorter motions. The ratio of their combined cost to the original cost is constrained not to exceed a user-specified threshold.

This process attempts to decompose each motion in $\hat{E}_l$ into two or more other motions that are also in $\hat{E}_l$. Decomposition (Figure 7.2.1) is allowed only if the concatenation of the components is within a user-specified threshold on cost increase, defined as a cost ratio $c_t > 1$ of the concatenated motion vs. the original one:

$$g(v_p) + c(v_p, v_s) \leq c_t g(v_s) \tag{7.2.1}$$

The component motions that can be reused to generate other motions are collected into E_a. E_a is designed to be capable of generating every one of the motions in $\hat{E}_l$, within the specified cost increase threshold.

A brute-force approach to decomposing $\hat{E}_l$ by enumerating all possible motion decompositions would be exponential in $\left|\hat{E}_l\right|$ and therefore prohibitively expensive. We propose two greedy algorithms that produce near-minimal E_a, given $\hat{E}_l$ and c_t.

Grounds for eliminating a path:

- Spatial similarity: a trajectory that is indistinct spatially from the other trajectory alternatives (that share the same start and end states) is removed;

- Relative cost similarity: a trajectory that is indistinct from other alternative trajectories in terms of its cost is removed. This condition is often faster and easier to test than the previous one.

7.3 SPATIAL SIMILARITY REGRESSION

An observation that many paths in the system reachability tree are very similar leads to the idea that there is no need to represent them explicitly in the control set. A straight-forward way to remove the redundancy in the reachability tree is to eliminate the paths that are very similar. A simple measure of similarity is path cross-track difference. An idea of path equivalence classes was proposed, where two paths were equivalent iff their cross-track error (along the whole path) did not exceed a certain value. An initial approach to control set generation replaced all equivalent paths in a reachability tree with a unique (somewhat arbitrary) representative path.

It was noted that, in the context of state lattices, it is important to connect discrete state values using paths in the control set. Similar to an equivalence class notion, a path could be considered to hit a discrete state value, if it passes within ε of it. This relaxation would introduce small state discontinuity, but a judicious choice of ε would preserve constraint satisfaction. Path Decomposition was proposed, where any path in the reachability graph would be broken up into component subpaths at the points that are within ε of state values of the lattice. This by itself would not address the infinite length of paths; however, it was shown that, as approximation, only the first segment of such decomposition could be placed into the control set. Path Decomposition control sets were the first usable automatic control sets. Control Set outdegree was controlled via the ε parameter. Using too high ε would lead to degenerate control sets with less-than-useable outdegree. They showed good quality of resulting plans (running A* in the induced state lattice), but large runtime (due to high min. outdegree achievable).

Even though the representable set of controls U_l is infinite, the reachability of many systems of practical interest can be captured well by analyzing a finite, albeit very large, subset $\hat{U}_l$. For example, for car-like robots, we could define $\hat{U}_l$ as the set of motions that are contained in a region (centered around the robot) that is much larger in extent than the robot's minimum turning radius. This motion set will include many maneuvers that the robot is capable of executing, including multi-point turns in close quarters. Next we develop an explicit and exact representation of $\hat{U}_l$ as a graph $\hat{G}_l = \hat{V}_l \cup \hat{E}_l$.

7.3.1 Analysis of Path Proximity to Nodes

Once we reached some conclusions about the convenient discretization of the C-space, we wanted to verify our intuition that the longer the paths are, the more likely they are to cross other nodes in the control set. We employed the following algorithm.

Input: Finite control set $\hat{E}_l$, cost threshold c_t
Output: Approximating control set E_a
$E_a = \emptyset$;
Run Dijkstra's search with $\hat{E}_l$, starting at O;
foreach *path in the control set* **do**
    Find the closest node, record its coordinates and distance;
    Move along the path, calculate the new distance;
    if *it is smaller* **then**
        Continue iterating;
    end
    else
        We are departing this node, use previous distance as the minimum distance;
        Repeat move;
    end
    Repeat for other nodes along the path;
    Keep a minimum of such distance to any node;
end

Algorithm 7.2: Spatial Similarity Decomposition.

Consider a path whose minimum distance to any node is 0. This means that the path is perfectly aligned to a node along its way. In this case it is reasonable to eliminate this path from the lattice because it is composed from two sub-paths, that must already exist in the lattice because they are smaller than the original path and by construction the control set includes all paths that start and end on the control set nodes and are within the control set radius.

With these principles, a comprehensive study of such minimal distances to any node has been undertaken for paths ending on control set perimeter. The results for a control set of angle radius 2 are summarized in the graph in Figure 7.3.2. In particular, we see that for lattice radii of over 10 all paths come to within 20% of control set node spacing. In other words, if we set the node tolerance to 20% (i.e. the path "hits the node" by approaching it inside this tolerance), then all of the

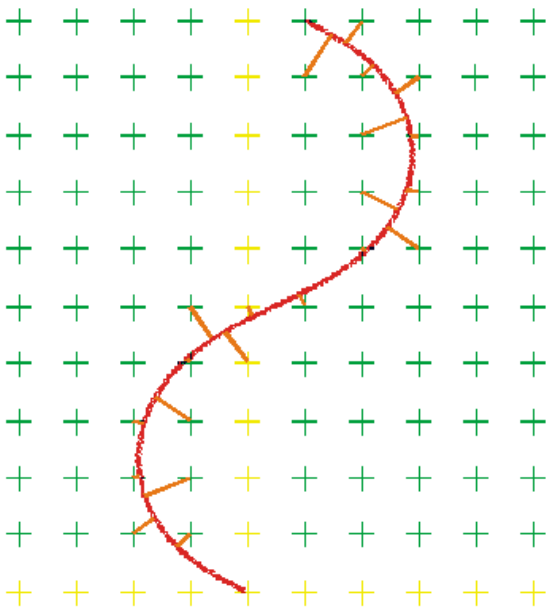


Figure 7.3.1: **Minimum distance to any node**. Illustration of the algorithm for calculating the path's minimum distance to any node. Orange highlights denote Euclidean distances to the nodes along the path. The minimum such distance is sought.

perimeter paths in the control set can be eliminated. It means that if we adopt the above tolerance, then the control set horizon we have been seeking is only 10. We can copy the control set of this size at every node in the lattice and thereby obtain any possible motion through workspace.

Also, a non-zero tolerance would suffice and we would still be able to eliminate all perimeter paths if we were willing to insert special connection nodes by choosing convenient for such nodes. They would be the ending points of the first sub-paths and the starting points of the second sub-paths. However, inserting extra nodes violates the regularity of the control set and invalidates all of its useful properties.

However, it may still be possible to eliminate some perimeter paths under the assumption of a non- zero tolerance and without inserting extra nodes. We may get lucky that for a perimeter path, there exist two sub-paths in the control set that, when concatenated, will generate another path that has the same initial and goal poses as the original path and that will be similar, or not spatially distinct from the original path. The Lebesgue measure is utilized in determining the

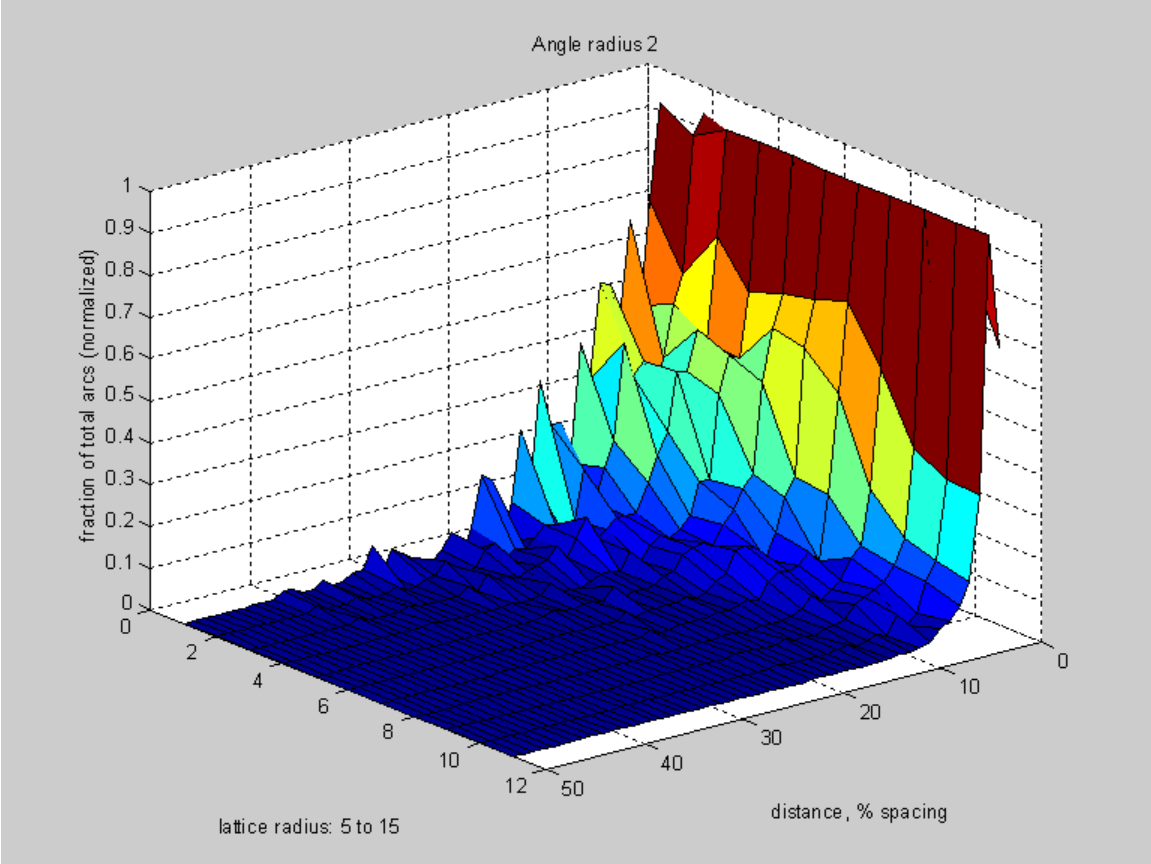


Figure 7.3.2: **Node distance histogram**. Summary of the minimum distance to any node study. This graph shows the percentage of the paths that have a particular value of minimum distance to any node, expressed as a percentage of 2D spacing between control set nodes, given a particular control set radius (here same as lattice radius).

spatial distinctness [84], or in other words the measure of distinctness in the unsigned area between the two paths. However, unlike in the first two path elimination scenarios, here the existence of a pair of eliminating sub-paths is not guaranteed.

Yet another approach is possible through understanding the scale dependence of the above discussion of tolerances. Consider an example where the size of the vehicle is 100cm, ; then 20% of that is only 4cm. Intuitively, this tolerance seems to fall well within the precision of mechanical vehicle control. Thus, if the path ideally takes the vehicle 4cm off from the node, then almost

certainly the vehicle will be able to drive exactly through the node (it may end up there anyhow due to inaccuracy of control). The only exception is when the ideal path already utilizes the maximum curvature of the vehicle, in which case if we still split the original path at the node in question, then the vehicle most likely will end up slightly off from the goal node. As long as this final error is within an acceptable tolerance (or the same mapcell), this maneuver is successful. The only caution is that in concatenating infinitely many such "inaccurate" paths, the terminal error will grow without bound.

7.3.2 Obtaining a Minimal Set of Paths

We obtained the minimal set of spatially-distinct paths by eliminating the paths that were not distinct. In some fairly large control sets, we were consistently able to eliminate over 98% of paths (i.e. thousands of paths). Figure 7.3.3 shows some successful examples of such path elimination.

However, as stated above, the existence of convenient sub-paths is not guaranteed, and so it occurred sometimes that paths could not be eliminated with the above approach. All of these cases seemed to indicate that the inverse-solution trajectory generator somehow lacked the capability to generate the required sub-paths. Sometimes it was related to the fact that the original path already operated at the limits of the maximum allowable value of curvature. Later it was determined that by allowing non-zero curvature at the connection point, it was possible to achieve many more eliminations. However, there always remained at least one or two perimeter paths (out of thousands!) that still could not be eliminated.

7.4 Cost Similarity Regression

7.4.1 Leave-one-out Regression

This approach involves elaborating a reachability tree of the system and attempting to reproduce each of its edges within a certain relative optimality limit. First, the path in question is removed from the reachability tree. The remaining reachability tree is used as a large control set to try to reconstruct the original path. If a reconstruction is possible, the path is taken out of the control set, the next path is picked and the algorithm proceeds. In order to make sure that parent vertices are treated last, the vertices are processed in descending order of their costs.

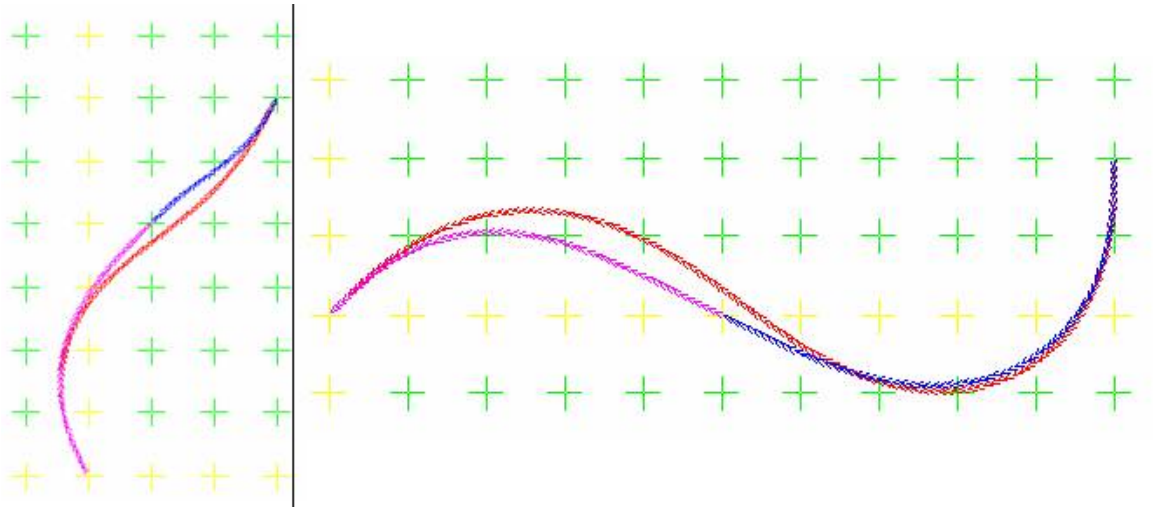


Figure 7.3.3: **Motion decomposition examples**. The original path is red, the two eliminating sub-paths are magenta and blue. In either case, two sub-paths happened to exist in the control set so that a). they could be concatenated to produce a path with the same initial and final pose as the original, and b). the produced path was not spatially distinct from the original.

This algorithm decomposes the motions in $\hat{E}_l$ in decreasing order of their cost. This implements a heuristic on managing the dependencies of component motions, namely that the higher-cost motions are likely to be decomposed by lower-cost motions. Each motion e_s to be decomposed is selected and removed from $\hat{E}_l$. Next, optimal A* search is used to compute a motion from O to the endpoint of e_s using vertex expansion $\hat{E}_l \setminus \{e_s\}$. If the least cost motion represents a cost increase greater than c_t, the e_s is reinserted into $\hat{E}_l$, since it cannot be decomposed satisfactorily. Otherwise, it is left out of $\hat{E}_l$, and the algorithm repeats by selecting the next motion to decompose. Very large values of c_t will lead to a degeneracy where almost all motions are decomposed. This condition can be detected by observing the resulting E_a.

7.4.2 D* DECOMPOSITION

This method involves generating a discretized, bounded reachability tree. Both discretization and bounding cause this reachability tree to be an approximation of the continuum. They are used in order to make this process tractable. The resulting reachability tree is used as a large control set in growing another reachability tree of similar size using Dijkstra's (best-first) search. This allows

generation of multiple ways to attain the nodes in the second reachability tree. The different ways of attaining each node are analyzed and only one way is selected such that it is still within the desired relative optimality and the size of the resulting control set is minimized. Ensuring the satisfaction of the relative optimality limit is done by explicitly considering predecessor's costs using the formula:

$$\beta = \min(\beta_d, \frac{a^* - c}{a - c})$$

where β_d is the desired (maximum) relative optimality threshold, β is the updated threshold of the predecessor vertex, a^* is the known optimal cost of a vertex, a is the cost of the path that goes through the given predecessor vertex, and c is the cost of the edge between the predecessor and the given vertex.

The previous algorithm has a disadvantage in complexity: it requires running A* "from scratch" $\left|\hat{E}_l\right|$ times, in graphs with large (albeit reducing) branching factors. Inspired by the capacity of incremental search algorithms, e.g. D* Lite [71], to prevent repetitive "from scratch" search by virtue of reusing computation, we propose an alternative with much better runtime (Algorithm 7.3). It does a single Dijkstra's elaboration of E_O vertex expansion to explore the extent of $\hat{U}_l$ (line 2) and then performs the decomposition analysis from Section 7.4.1 while reusing the collected vertex predecessors, similar to evaluating the *rhs*-value (4.2.1). In this manner, unlike the traditional application in planning, the principles of the D* algorithm are used here for search space design.

Once the predecessors are computed, only those that can possibly yield c_t-decompositions are retained (lines 3-7). If a state has only a single predecessor, it implies by construction that the one and only motion that connects it to the origin is the original edge in $\hat{E}_l$, and motion decomposition cannot succeed. This edge is entered into E_a. Since the motions that form decompositions may be further decomposed themselves, we keep track of per-vertex cost thresholds with a database db; thresholds for all vertices are initially set to c_t on line 5.

Line 8 follows the cost-sorting heuristic discussed in Section 7.4.1. The next line selects edges in decreasing order of cost of their destination endpoints. Lines 10-12 re-evaluate the number of admissible predecessors. Line 13 enumerates the options for decomposing the edge e_s leading to v_s. The algorithm attempts to select the decomposition that is likely to minimize the final E_a. To this end, it uses two other heuristics. The first one is sorting potential decompositions in increasing

Input: finite control set $\hat{E}_l$, cost threshold c_t
Output: approximating control set E_a

1 $E_a = \emptyset$;
2 Run Dijkstra's search with $\hat{E}_l$, starting at O;
3 **foreach** $e_s \in \hat{E}_l$ **do**
4 $\quad Pred_{c_t}(v_s) = \{v_p, \text{s.t. } g(v_p) + c(v_p, v_s) \leq c_t g(v_s)\}$;
5 $\quad db(v_s) = c_t$;
6 $\quad$ **if** $|Pred_{c_t}(v_s)| = 1$ **then**
7 $\quad\quad E_a = E_a \cup e_s$;
8 $\quad$ **end**
9 **end**
10 $E_{sort} = \text{sort}(\hat{E}_l)$;
11 **foreach** $e_s \in E_{sort}$ **do**
12 $\quad$ **if** $|Pred_{c_t}(v_s)| = 1$ **then**
13 $\quad\quad E_a = E_a \cup e_s$;
14 $\quad\quad$ continue;
15 $\quad$ **end**
16 $\quad$ **foreach** $v_p \in sort(Pred_{c_t}(v_s))$ **do**
17 $\quad\quad$ **if** $e_{v_p,v_s} \in E_a$ **then**
18 $\quad\quad\quad$ adjust_threshold(v_p, v_s);
19 $\quad\quad\quad$ goto 9;
20 $\quad\quad$ **end**
21 $\quad$ **end**
22 $\quad v_p^* = \underset{v_p \in Pred_{c_t}(v_s)}{\text{argmin}} \; g(v_p) + c(v_p, v_s)$;
23 $\quad$ adjust_threshold(v_p^*, v_s);
24 $\quad E_a = E_a \cup e_{v_p^*,v_s}$;
25 **end**

Algorithm 7.3: D* Decomposition.

order of cost (sort on line 13), in an attempt to choose a decomposition with cumulative cost that is as close as possible to the original motion. The second heuristic is preferring a decomposition, in which one of the segments is already found in E_a (lines 14-16), in an attempt to avoid adding new elements to E_a.

Once an edge is decomposed into two segments connecting at vertex v_p, the segment leading

Input: vertices v_p and v_s
Output: $db(v_p)$, $Pred_{c_t}(v_p)$, and E_{sort} modified
1 $\alpha = (g(v_p) + c(v_p, v_s))/g(v_s)$;
2 $c_t' = \frac{db(v_s)g(v_s) - c(v_p, v_s)}{\alpha g(v_s) - c(v_p, v_s)}$;
3 **if** $c_t' < db(v_p)$ **then**
4 $\quad db(v_p) = c_t'$;
5 $\quad Pred_{c_t}(v_p) = \{v_p'$, s.t. $g(v_p') + c(v_p', v_p) \leq db(v_p)g(v_p)\}$;
6 $\quad$ **if** $g(v_p) \geq g(v_s)$ **then**
7 $\quad\quad E_{sort} = E_{sort} \cup e_p$;
8 $\quad$ **end**
9 **end**

Algorithm 7.4: adjust_threshold function.

to v_p may in turn be decomposed; however the cost threshold for its decomposition may have to be different from c_t. To see this reasoning, suppose path length is taken as cost (Figure 7.2.1). Using the cost sorting heuristic, the longer, original motion (terminating in v_s) will be decomposed before the first segment of decomposition (terminating in v_p). When the latter is in turn selected for decomposition, we rewrite (7.2.1) as:

$$c_t' g(v_p) + c(v_p, v_s) \leq c_t g(v_s) \tag{7.4.1}$$

Depending on the value of $c(v_p, v_s)$, the required cost increase threshold c_t' for the recursive decomposition of v_p may be $c_t' < c_t$. Thus, as soon as a decomposition relationship is established between the vertices, the necessary reduction of cost threshold of the predecessor vertex is computed with the function *adjust_threshold*. If the algorithm gets to line 17, no decomposition opportunities have been selected by the heuristics. This line selects a decomposition with the least-cost predecessor; it is added to E_a on line 19.

Algorithm 7.4 presents the *adjust_threshold* function. In line 2, it computes c_t', the new cost threshold for the predecessor vertex v_p. If this value is less than the previous one, it is recorded on line 4, and the list of admissible predecessors is reviewed on line 5. Lines 6-7 address the case where the cost sorting heuristic is incorrect, and the predecessor v_p has equal or greater cost than v_s. In this case, any changes done to v_p need to be propagated to its potential predecessors, so the next iteration of Algorithm 7.3 must select v_p's edge, e_p (line 3).

CHAPTER 8

EXPERIMENTAL RESULTS

8.1 VALIDATING STATE LATTICE MOTION PRIMITIVES

A set of experiments was conducted in order to assess the performance of state lattice motion primitives in comparison with popular types of motion primitives in motion planning with differential constraints. These experiments invoke the planner, based on such primitives, with a number of queries in a variety of configurations. Here we compare our planner with other algorithms: the basic grid-based planner (on 4-, 8-, and 16-connected grids) and the well-known Barraquand and Latombe (BL) nonholonomic planner [9]. In doing so, we attempted to level the playing field for fair comparison. Therefore, the same implementation of A* was used for all algorithms. The primary differences were in the node expansions, and the cost and heuristic estimates. For the former, we have adopted a notion of a generalized control set: for the grid search it is simply a grid-based node expansion (straight paths), whereas for the BL planner it is the node expansion as described in

that work. The first set of tests looks at how the sample lattice control set performs in comparison to other types of generalized control sets. Next the impact of the heuristic look-up table on state lattice performance is examined.

8.1.1 EXPERIMENTAL SETUP

A representative lattice control set was used in all tests. Its state space consisted of the 2D translational coordinates, heading and curvature (x, y, θ, κ). For the sake of simplicity, curvature was constrained to be zero at each discrete state. This control set is depicted in Figure 8.1.1.

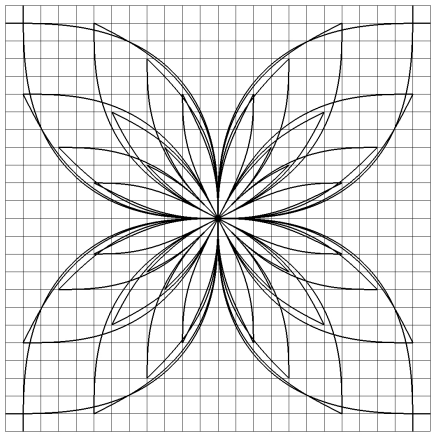


Figure 8.1.1: **Experiment control sets**. The state lattice control set selected for testing has 16 discrete headings, a maximum curvature of 1/8 of cell, and an average outdegree of 12, for a total of 192 curves. The straight edges cannot be seen because they are obscured by the longer curved edges.

For these tests, a list of 10,000 random queries was generated, consisting of an initial and final state, also expressed as position, heading, and curvature. The set of queries was generated with the intent to require the planner to produce paths ranging from simple (nearly straight paths) to complex (e.g. parallel parking or *n*-point turn maneuvers) among obstacles. Each query was tested with a wide variety of configurations, including different obstacle fields, alternative control sets such as a grid or Barraquand-Latombe, and A* heuristic functions.

Start and goal states were produced with a random number generator that provides evenly

distributed real values in a requested range. Initial positions were selected at random from the free space in order to produce the maximum possible variety of queries. The goal position was then specified by a randomly selected radius and angle specified in polar coordinates with respect to the start, repeating this step as necessary to ensure that the goal is also in the free space. Initial and final headings were randomly selected, and curvature at end-points was constrained to be zero. Goal states were constrained to be no more than ten minimum turning radii (80 cells) from their corresponding start states when projected onto x, y space.

Each query was tested in two worlds. In both worlds, cost to traverse free space was held constant at 1 unit per cell of free space. Hence, path cost was equal to the distance traveled. The baseline case was an obstacle-free world, meaning that the HLUT gave exact solutions for cost. Results were also obtained using a world with randomly placed point obstacles, shown in Figure 8.1.2 with paths generated by two different planners. These obstacles are the size of one map cell, which in this control set corresponds to 1/8th of the minimum turning radius. Points were generated with uniform distribution and 5% density in the plane.

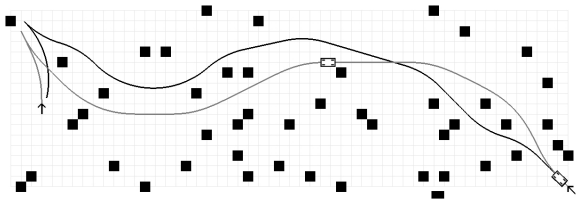


Figure 8.1.2: **World with Obstacles**. A portion of the world with point obstacles used in the experiments is shown here. Two plans are shown which solve the same query. The black line shows the plan generated by the state lattice, while the grey line traces the path returned from the BL planner.

Metrics for performance included time, memory consumption, and the quality of the resulting plan. In analyzing the performance of the planner, it is necessary to distinguish among the spectrum of queries ranging from simple to complex, as was suggested above. In the case of a grid, Euclidean distance would be an appropriate measure of difficulty. When planning in full state space, however, the length of the path followed by a nonholonomic vehicle can vary widely even in cases where Euclidean distance between the end-points is held constant.

In order to quantify the complexity of a particular query, the distinction is made between ab-

solute difficulty, which is proportional to path length, and relative difficulty, which reflects how much the resulting path deviates from a straight line. Figure 8.1.3 illustrates the two concepts. Both factors contribute to the overall resource requirements for a particular planning problem. Absolute difficulty is measured here by the path length. In the case of relative difficulty, the ratio of Euclidean distance / nonholonomic distance was used. In this scale, values near one mean that the resulting path is nearly a straight line, while those values nearest to zero indicate that much maneuvering is necessary to reach the final pose. In the analyzes below, results are shown for queries with absolute difficulty of 40 cells. This number was chosen arbitrarily for clarity of presentation. Any other absolute difficulty would have conveyed similar results.

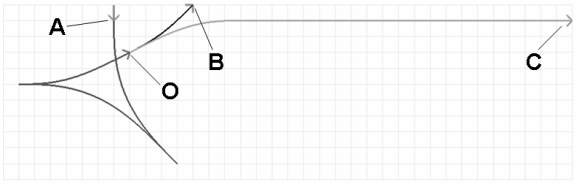


Figure 8.1.3: **Absolute and Relative Query Difficulty**. The difficulty of a query can be quantified in two dimensions. Each path, A, B, and C, starts at O. Query A is high in absolute difficulty as well as relative difficulty because it is long and has multiple cusps. B is simple in both measures. Query C has the same absolute difficulty as A (same length), but the same relative difficulty as B (nearly a straight line).

In the sequel, a variety of alternative control sets are considered. In each set of cases, the same framework is used, including an A* planner, heuristics, and world representation. The only distinction is the method by which neighboring states are generated during the expansion step. A standard lattice control set, depicted in Figure 8.1.1, was used across all tests. An overview of all the control sets which are considered here is presented in Table 8.1. Conceptually, a control set with longer edges requires fewer expansion steps to reach the goal but a greater total length to integrate, proportional to the outdegree. Correspondingly, a higher outdegree increases memory requirements for the planner and computation in the expansion step, but also improves the chance of finding a more direct route in fewer plan steps. Below, the best performing of several combinations is empirically determined.

Control Set	Total Edges	Ave. Edge Length	Ave. Outdegree
original control set	192	8.72	12
ctrl set w/ turn-in-place	224	7.47	14
BL	96	4.00	6
grid-4	4	1.00	4
grid-8	8	1.21	8
grid-16	16	1.72	16

Table 8.1: **A quantitative overview of control sets**. Parameters of a control set have a strong influence on how the planner will perform while using them.

8.1.2 Comparison to a Grid Planner

The basic grid search method of planning does not consider heading, which leads to several drawbacks. First, it is not clear what the heading state is unless two adjacent states on the path are compared and even in this case, the heading can only have 4 values. Second, it is impossible to plan feasible paths for vehicles incapable of a point turn. Third, heading continuity constraints cannot be enforced. A grid has nonetheless been widely used due to its simplicity and efficiency.

But is this approach inherently faster than searching in a full-dimensional state space? For this test, the planner was run with four different control sets, in each case using a heuristic function which always returned the exact distance to the goal. The basic state lattice was matched with a large HLUT. Three grid control sets were tested, in which each state is connected to its 4, 8, and 16 nearest neighbors (Figure 8.1.4), using a perfect heuristic for each level of connectivity.

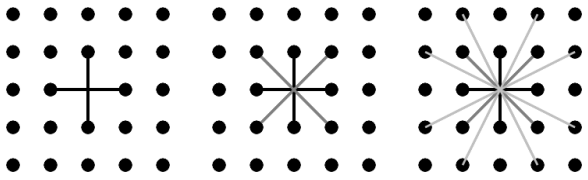


Figure 8.1.4: **Grid Control Sets**. Three different grid control sets were tested. A point is connected to the 4, 8, or 16 nearest neighbors that have unique headings.

Performance of the control sets in the absence of obstacles is shown in Figure 8.1.5. In order to normalize the results, only queries of absolute difficulty equal to 40 cells were used. The data are

presented across a range of relative difficulty. In the absence of obstacles, very interestingly, the lattice performs on par with basic grid search. Only on most difficult queries, the lattice consumes more CPU time than a grid control set because the nonholonomic path solution diverges more dramatically from a straight line solution. Of course, for more difficult problems, the answer returned by the grid is increasingly infeasible to execute on a real vehicle - and the path is also unlikely to arrive at the correct heading.

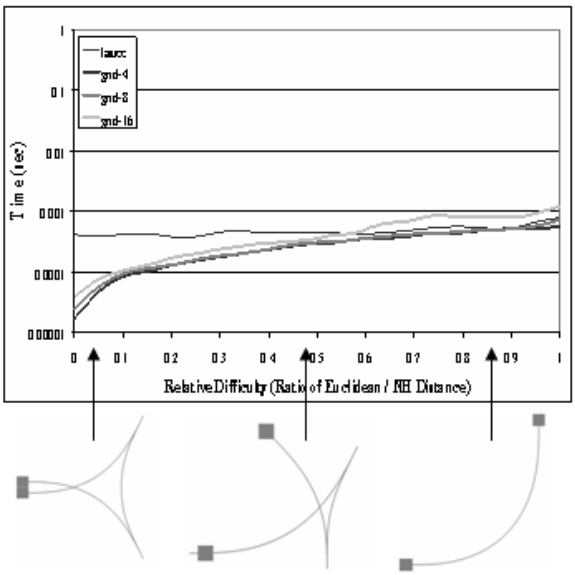


Figure 8.1.5: **Lattice vs. Grid without Obstacles**. Queries with an absolute difficulty of 40 cells are shown. The lattice performs similarly to the grid planners overall. Only for the greatest relative difficulty (nearest zero) does the lattice require more CPU cycles. Of course, the benefit of this extra computation is a plan which is feasible.

Results in the presence of obstacles are shown in Figure 8.1.6. It is intuitive that the grid should outperform the lattice in this obstacle field for any class of query. If an obstacle appears in the path of a grid plan, that plan is often displaced by only a few cells in order to circumvent the obstacle. In the case of the lattice under test, however, only smooth continuous paths with a maximum curvature of 1/8 = 0.125 are considered. These requirements substantially limit the planner's options, making it more difficult to find a satisfactory path through an obstacle field as shown in Figure 8.1.7. So while the grid path deviates slightly in the cluttered environment, paths generated by the state

lattice are often much more sophisticated in order to plan around obstacles. Thus, the difficulty experienced by the lattice planner reflects the true mobility limits of the vehicle. In plain terms, a grid planner produces faster answers, but they are usually wrong. Despite the increased search requirements, the lattice remains consistently only one order of magnitude slower than the grid planner, returning on average in less than 0.1 second for all classes of query.

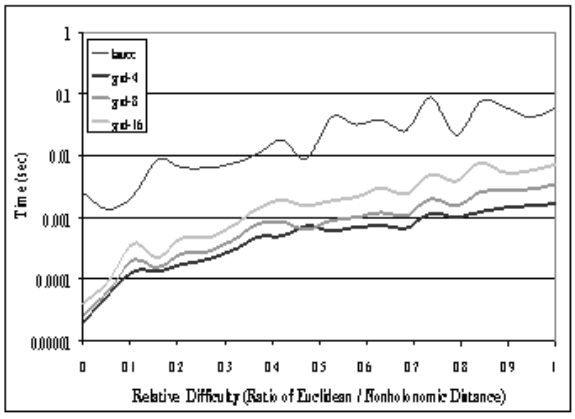


Figure 8.1.6: **Lattice vs. Grid with 5% Obstacle Density**. Queries of length 40 cells are shown. The grid outperforms the lattice, but it usually returns infeasible solutions in a dense obstacle field. Lattice planner runtime is still acceptably fast.

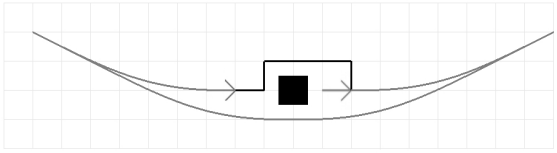


Figure 8.1.7: **Obstacle Avoidance with Two Control Sets**. Grid planners can easily avoid small isolated obstacles, as shown by the black path. By contrast, the grey path has limited curvature and so can be feasibly traversed by a constrained vehicle.

8.1.3 COMPARISON TO FULL *C* SPACE AND NONHOLONOMIC PLANNERS

The comparison between the state lattice and grid-based planner is not entirely fair, since the two planners search different spaces, and the grid results cannot be traversed by nonholonomically constrained vehicles without post-processing. The Barraquand and Latombe (BL) planner is well known and has been a popular nonholonomic planner for over a decade. It has also been used in a variety of real mobile robot applications [7], [96]. In this section, several different implementations of that planner are examined. In all cases, the identical A* algorithm was applied, but the heuristic cost estimate was altered to produce a fair comparison. In the first case, the algorithm was run as documented in [9], with a zero heuristic applied. In the second case, Euclidean distance was used as heuristic. Finally, an attempt was made to produce a perfect heuristic by generating an HLUT for the BL planner, for a fair comparison with an HLUT-enabled lattice planner.

In order to make the BL control set work in a fashion compatible with the state lattice test framework, an adjustment was necessary. Barraquand and Latombe [9] describe the edge lengths as being the L^1 diameter of a cell and assert that their planner requires the discretization of the search parameters to be fine enough. The state lattice's discretization settings were experimentally adjusted to allow proper operation of the BL planner: the edge lengths were set to four cell widths. The resulting reachability tree is shown in Figure 8.1.8 along with the standard state lattice tree. Reverse edges were omitted from both trees for clarity.

The performance of BL with three different heuristics is shown in Figure 8.1.9 along with the state lattice using two different heuristics. The heuristic which enforces optimality for the BL planner is zero, as proposed by authors, but the computational time required makes this impractical in most situations. In a fair match-up using the Euclidean distance heuristic, the two planners perform comparably, but only the lattice retains optimality (Figure 8.1.10). However, note that the Euclidean heuristic was used for the lattice planner only for the fairness of the comparison. The lattice planner using the HLUT significantly outperforms our implementation of BL, both with and without its own HLUT, as shown below.

The HLUT generation algorithm was run using a BL control set in an attempt to produce a perfect BL heuristic. However, in A* runs which used this heuristic, many false leads were still explored. The HLUT generation algorithms discussed above make a fundamental assumption that the cost between two points in the graph depends only on their relative states because the lattice is

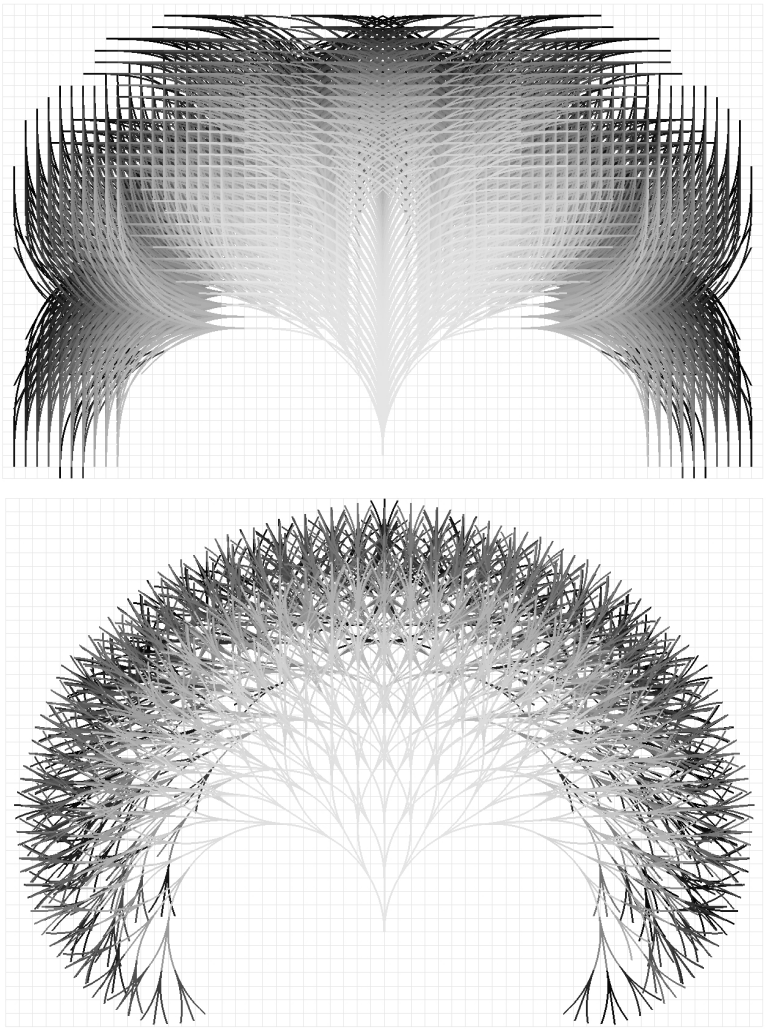


Figure 8.1.8: **Reachability Trees for the Lattice and BL**. At top, the first 1400 expansions of the basic state lattice control set in best-first order. At bottom, the first 1400 expansions of the BL control set were generated using the same algorithm. Lighter edges are older in both images.

regular in translational discretization. The BL algorithm, however, does not operate on a regular search space. As edges grow outward, other edges are excluded, such that the path between two points depends on the order in which states were expanded to produce each edge between the two points. So while using the HLUT resulted in the best BL performance, this heuristic cannot be

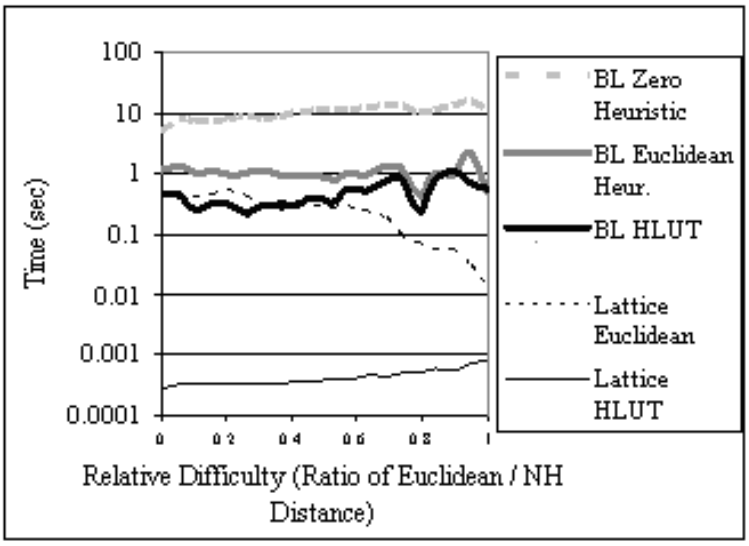


Figure 8.1.9: **Lattice vs. BL without Obstacles**. Queries with an absolute difficulty of 40 cells are shown. Various heuristics are considered for each planner. Only when the slow Zero heuristic is used does BL return optimal paths.

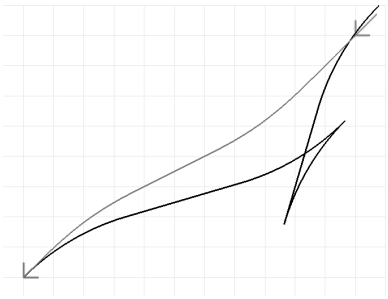


Figure 8.1.10: **BL sub-optimality example**. In black, the BL planner with Euclidean heuristic is used. In grey, the standard lattice control set with Euclidean heuristic returns an optimal path. BL does not because it does not permit revisiting of states.

guaranteed to be admissible for that algorithm. Note, however, that with any non-zero heuristics the planner does not guarantee optimality anyway.

An alternative lattice was also tested. This control set has the same controls as the basic lattice,

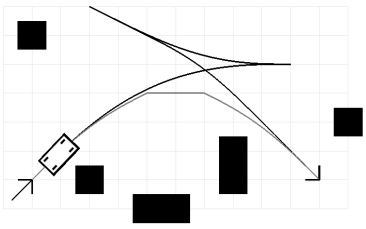


Figure 8.1.11: **Control sets with and without turn-in-place**. In black, a control set with a minimum turning radius of 8 cells is used. In grey, the same control set is augmented by two additional controls which allow the vehicle to turn in place at a cost equal to traversing 5 cells. This plan performs the turn-in-place maneuver twice in order to execute a sharp turn.

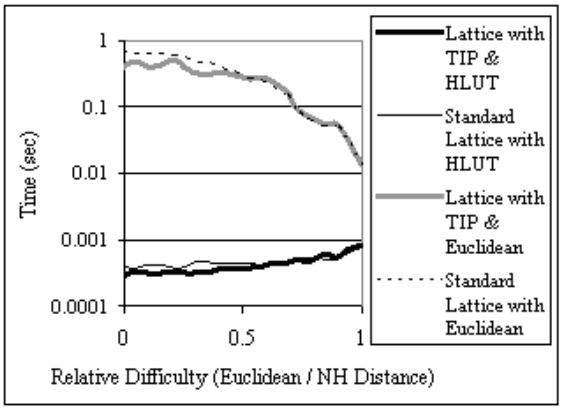


Figure 8.1.12: **Lattice vs. enhanced lattice without obstacles**. Queries with an absolute difficulty of 40 cells are shown. Performance for the two lattices is comparable.

except that extra zero-length edges were added so that the vehicle is permitted to turn in place. The cost of these additional edges was computed based on the aggregate distance of travel of each of the wheels in the vehicle. The effect of this capability on resulting plans is depicted in Figure 8.1.11, where the path is substantially shortened by avoiding circuitous maneuvering. These additional controls have no significant effect on overall planner performance (Figure 8.1.12), but they provide valuable added flexibility in negotiating dense obstacle fields. Moreover, for practical applications, the lattice planner algorithm allows tuning the robot's preference for performing point turns versus smooth maneuvers by assigning appropriate costs to the zero-length edges.

8.2 GRADUATED FIDELITY

In order to evaluate the effects of graduated fidelity, we performed simulated and field experiments with mobile robots. In all experiments, a planetary rover prototype roughly 0.8m by 1.0m in size was used. It was assumed to be a nonholonomic vehicle capable of moving forward and back with minimum turning radius 0.5m. The vehicle was capable of point-turns, although such maneuvers were considered costly and used as a last resort. The search space consisted of two subgraphs: a high fidelity subgraph that moved with the vehicle and a low fidelity subgraph elsewhere. The (x, y) resolution of both subgraphs featured square cells of 20 cm. The average outdegree of the 4D high fidelity subgraph was 45 (a state lattice). The outdegree of the 2D low fidelity subgraph was 8 (grid connectivity). The dimensions of the state lattice included the translational coordinates (x, y), discretized as a grid, as well as heading and curvature. The planner used the Euclidean distance heuristic, and an improvement in performance can be expected with better informed heuristics.

The set of experiments below attempts to quantify the effects of the size of the high fidelity subgraph around the vehicle on both runtime performance and the quality (cost) of the solution. The second set of simulated and real vehicle experiments demonstrates the performance of the planner in a realistic setting.

The plot in Figure 8.2.1 reports the results of the first type of experiments. The planner used an environment with random single map cell obstacles independently and identically distributed with probability of 3%. This is the same environment as used later in the second set of simulated experiments, and it is illustrated in 8.3.1. To avoid confounding the results with simulated perception, the cost map was fully known *a priori*. For each experiment, the robot was placed at a random location in its environment, and the goal was chosen approximately 100 cells away. The robot then navigated toward the goal, while periodically replanning after traversing approximately two cells. The replanning runtime and the costs of traversed edges were accumulated. The runtime was computed as (real-time) seconds elapsed for each planning computation. The experiment was repeated using a different size of the high fidelity subgraph around the robot. Both the runtime and cost measurements were normalized by dividing by the respective quantity, featured by the lattice planner without graduated fidelity (high fidelity throughout). The plot suggests that using the high fidelity subgraph of a smaller size appears to offer a significant performance improvement with a very small increase in traversal cost. For larger sizes of the subgraph, the runtime increases

significantly. This is expected in the given scenario, where the subgraph around the vehicle grows radially outward. Larger values on the x-axis represent larger diameters of this subgraph. The increased circumference causes significantly larger numbers of perimeter vertices to be processed to reconnect the subgraphs. The plot demonstrates that beyond a certain size of the movable subgraph (here about 20% of the distance to the goal), the runtime costs of modifying the search space exceed its runtime benefits. In this application, the approach is mostly helpful for smaller sizes of the subgraph: for a size of 10% of the distance to the goal, the runtime benefit is almost an order of magnitude. Note that in most realistic settings, the perception horizon (which would typically prompt high-fidelity representation) is typically much less than 10%, and the motion goal of the robot may be very far, perhaps kilometers away. Differentially constrained planning at high fidelity would not be feasible at such scale.

The graduated fidelity trials are somewhat at a disadvantage in the study above, since a robot using high fidelity throughout requires no replanning, given the assumed known environment. In a realistic setting, replanning may still be required due to perception updates. If we choose the settings from the simulated robot navigation experiments, where the perception is limited to a realistic radius around the vehicle, the benefits of varying fidelity are more pronounced, as shown in Figure 8.2.2. For extremely large sizes of the high fidelity subgraph (over 40% of the distance to the goal), the runtime cost is significant due to repairing connections for a large number of perimeter vertices. However, as argued above, much smaller sizes of high fidelity subgraphs are meaningful for real applications.

8.3 AUTONOMOUS NAVIGATION

Here we present the results of experiments, simulated and real, that demonstrate planner performance in a realistic setting, where the robot moves in a challenging environment toward a distant target. In the simulated experiment, the robot has a perception region limited to 21x21 cells, centered around the robot. No perception information is available outside this horizon. The size of the high fidelity region is the same size as the perception region. Otherwise, the setup is the same as above. For clarity, Figure 8.3.1 shows a 40-meter subset of a 500-meter path, traversed in this setting. Grey cells are obstacles that have not yet come into view of the robot and are unknown to it. Black cells (and red cells in the insets) are obstacles that were seen by the robot. The dark-gray

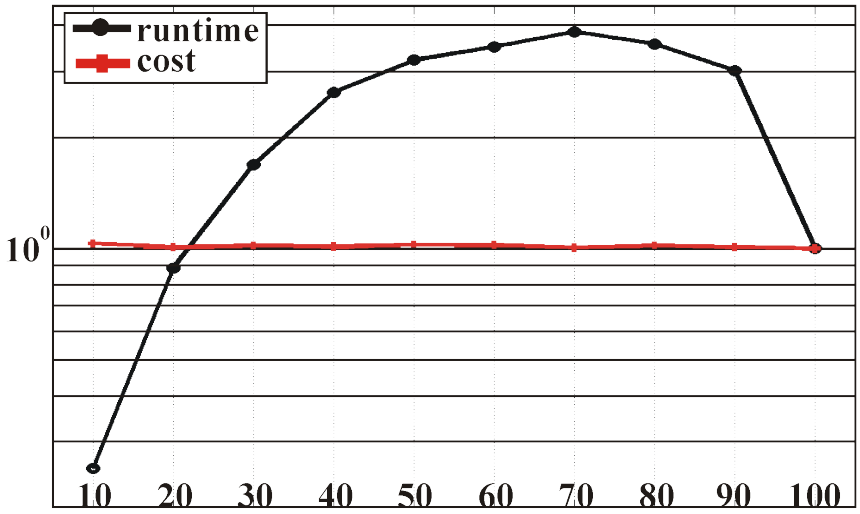


Figure 8.2.1: **Simulated robot traversal**. The results of the simulated robot traversal using graduated fidelity. The horizontal axis is the percentage of the search space (between robot and goal) occupied by the high fidelity subgraph. For example, 10% denotes a high fidelity subgraph of radius 10 cells around the robot, given that distance to the goal is 100 cells. The vertical axis is the ratio of cumulative runtime and traversal cost at the given subgraph size to the corresponding quantities without graduated fidelity (high fidelity throughout). Averages over 10 independent trials are presented.

line is the path the robot traveled. Note that it explored many cul-de-sacs due to the limited perception, and the planner was effective at guiding the robot out of all of them, while leveraging robot's maneuverability. The replanning due to obstacle discovery and fidelity modification in the search space was occurring continuously. This experiment was performed on a conventional laptop computer with 2GHz CPU and 2GB of RAM. Notice the two peaks in the middle of the runtime plot, bottom of Figure 8.3.1: they correspond to two replans #39 and #53, when significant replanning was required due to cul-de-sacs.

The graduated fidelity motion planner was integrated with research prototype rovers at the Jet Propulsion Laboratory (JPL). It enabled rovers to navigate in rough rocky terrain set up in the JPL Mars Yard. Figure 8.3.2 shows the results of the FIDO rover running the graduated fidelity state lattice planner on-board to navigate autonomously amid dense rocks. In this experiment, the robot featured a single 1.6GHz CPU and 512MB of RAM, shared among all processes of the rover. The actual memory usage of the planner was less than 100MB. The rover used a high fidelity region of

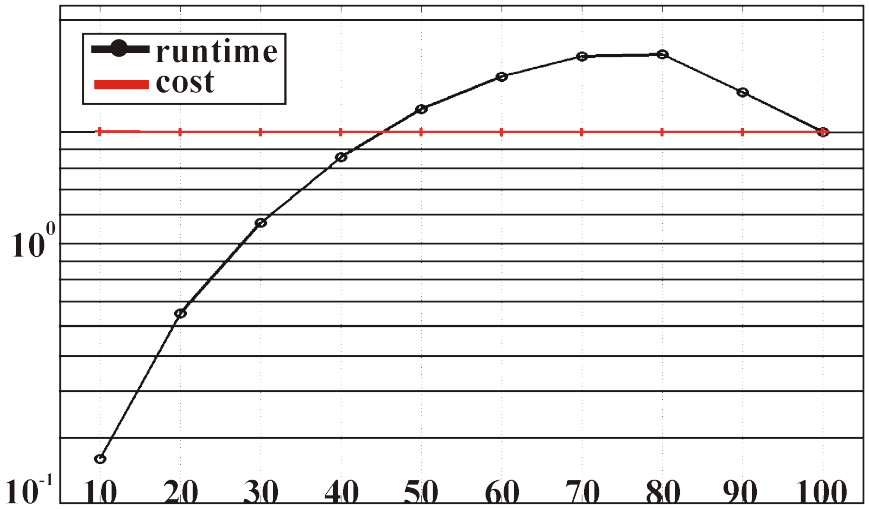


Figure 8.2.2: **Simulated robot traversal: limited perception**. The results of the simulated robot traversal using graduated fidelity as in Figure 8.2.1, but with the a limited perception horizon. Averages over 10 independent trials are presented.

the same size as above, 21x21 map cells, and a perception region (via stereo cameras) 41x41 map cells, both centered around the rover. The top part of Figure 8.3.2 shows the approximate path the rover traveled, and the bottom part shows the plot of the runtimes of the corresponding replans, averaging at approximately 10Hz.

8.4 KINODYNAMIC PLANNING

We present experimental validation results of kinodynamic planning using two examples: a double integrator system inspired by [29] and a car-like system with complex dynamics. We also describe a multi-query BVP approach (Section 7.1) that is helpful for implementing the second example, presented in Section 8.4.2. Both examples were developed by applying the design principles in Chapters 5 – 7. The primitives, developed with Algorithm 7.3, are then used to perform incremental search by utilizing the unmodified D* Lite algorithm [71] (Figure 8.4.3).

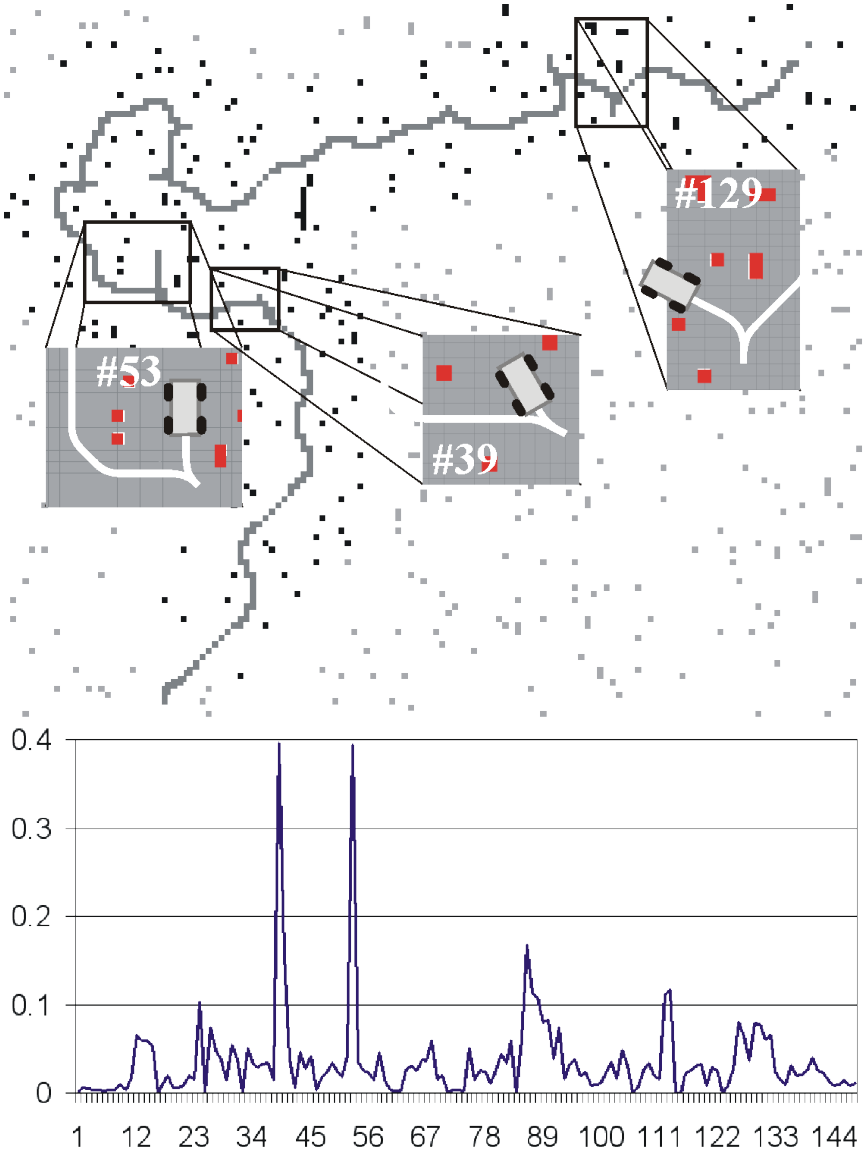


Figure 8.3.1: **Long-distance simulated forest traversal**. A simulated experiment of traversing about 500 meters among previously unknown obstacles. Top: the first 40 meters of the path are shown for clarity. Note that all motions generated by the planner were globally feasible, and backing up maneuvers were generated automatically when necessary. Bottom: plot of planner runtime (vertical axis is runtime in seconds, and the horizontal axis is replan cycle number).

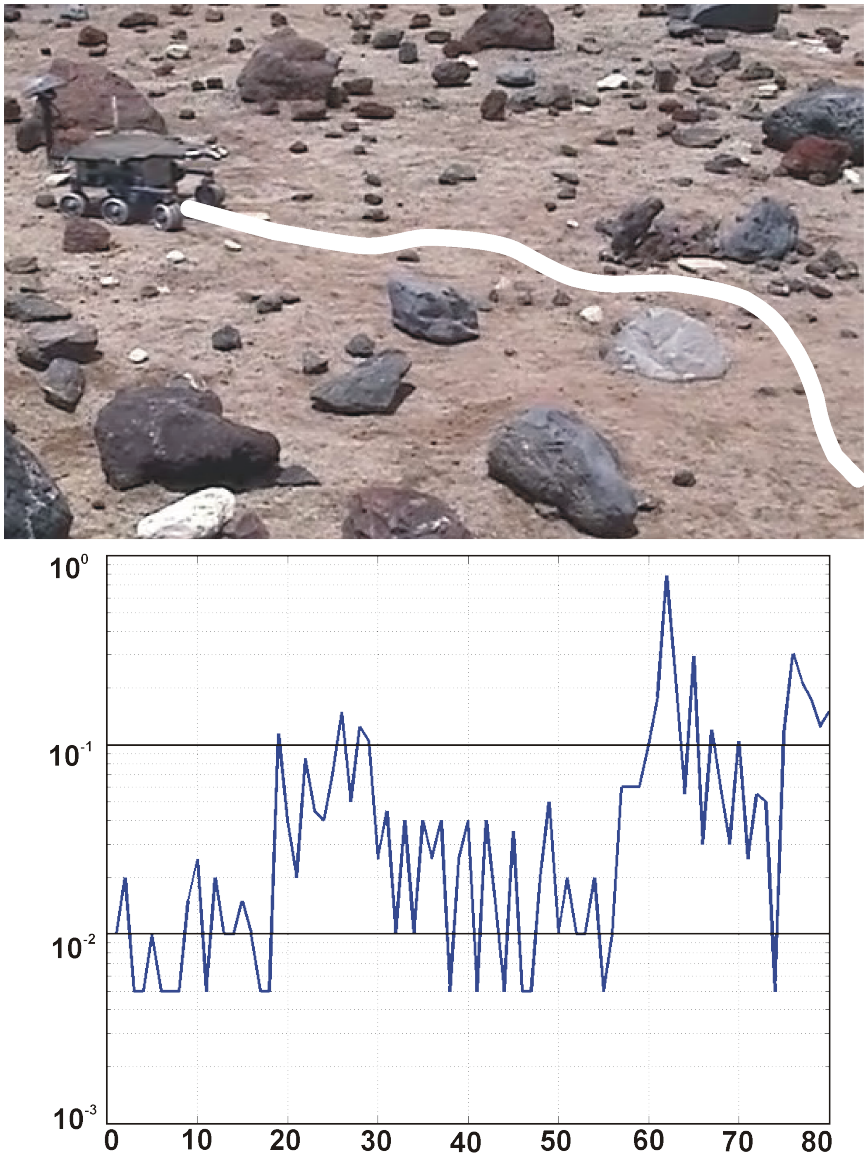


Figure 8.3.2: **JPL Mars Yard experiment**. FIDO rover navigated autonomously among previously unknown obstacles, while running the graduated fidelity lattice planner on-board. Top: approximate path the rover followed. Bottom: a plot of planner runtime (vertical axis is runtime in seconds, and the horizontal axis is replan cycle number).

8.4.1 DOUBLE INTEGRATOR

The *double integrator* is a simple dynamics system, where the acceleration is controlled directly. Formally, it is governed by a second-order differential equation $\ddot{x} = u$, where $x, u \in \mathbb{R}$. We look at a one-dimensional example here, although more dimensions, similarly defined, can be added easily due to their independence. The state transition equation $\dot{x} = f(x,u)$ can be written as $(\dot{x}_1, \dot{x}_2) = (x_2, u)$. Thus, the terminal states of a trajectory, x_{init} and x_{goal} are specified by the corresponding positions and velocities. A discrete-time model with controls given by $u = \{-1, 0, 1\}$ has reachability tree trapped on a lattice (Figure 8.4.1a).

To illustrate the application of Algorithm 7.3 to this system, consider an example in Figure 8.4.1b. The system starts at zero position and zero velocity and attempts to reach a specified position value with zero terminal velocity. The trajectory in this example is the sequence of black arrows in part b) of the figure. Part c) shows a few of the elements of the set $\hat{E}_l$ for this example; the whole set is prohibitively large to illustrate. Given this input, the algorithm does Dijkstra's (line 2), which generates the sets of admissible predecessors of the vertices. This set for the goal state, $Pred_{c_t}(x_{goal})$ is shown in gray in Figure 8.4.1b. Since the edges between all the states in this set come from the original selection of edges allowed in this problem setup, we observe that the solution E_a contains the same set of edges we started with (Figure 8.4.1a). In the case of this simple problem, the algorithm is able to find the optimal solution (that is, resulting c_t ratio is 1.0), however it is usually not the case for more realistic problems, as suggested by the following sections. The solution path (black line in part b) of the figure) can be constructed with the replanning search algorithms we consider here.

8.4.2 CAR-LIKE ROBOT WITH DYNAMICS

The system in this example is a wheeled robot with three driven wheels, one of which steers, as shown in Figure 8.4.3b. The system is simulated using the Open Dynamics Engine^{TM}software; the system model is not available in closed form. The vehicle has significant mass, is capable of achieving high speeds and is placed on a very slippery flat surface to highlight the effects of dynamics, such as significant drift, sliding sideways, etc.

A comparative study of lattice and high-diversity primitives [20; 32], based on A* search, showed that the difference in performance in terms of runtime, solution quality and completeness

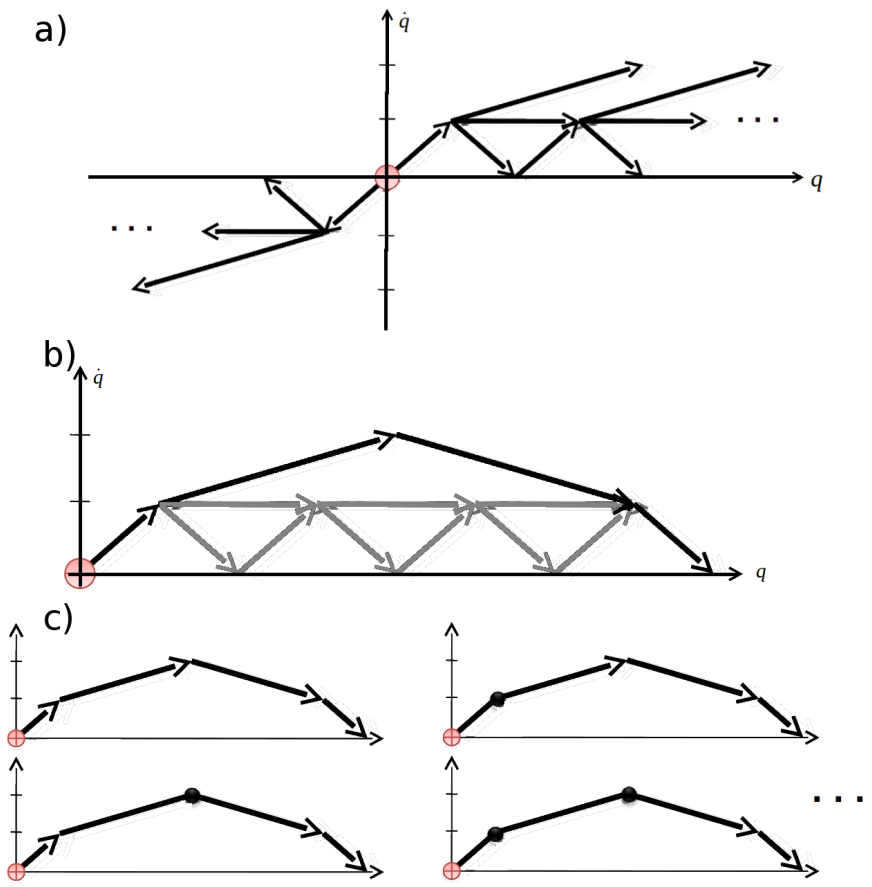


Figure 8.4.1: Double integrator system example.

of both types of primitives is less than statistically significant.

The benefit of the principled approach here is that the length of primitives is selected automatically, while it is fixed for path diversity primitives.

Next we describe experimental validation of incremental search in this context, although other planning approaches may benefit from such primitives. Figure 8.4.3 illustrates an example where a new obstacle invalidated a segment of the previously computed trajectory. D* modified the plan efficiently by limiting vertex expansions to a small neighborhood. The initial plan was computed in 1.42 seconds (on commodity hardware), and was repaired in 0.35 seconds – a nearly 4-fold speedup with respect to re-planning from scratch.

8.5 ON-DEMAND EXPANSION

This section evaluates on-demand expansion technique in the context of state lattices. Figure 8.5.2 demonstrates path alternatives that have the same initial and final state as the canonical path (notice that all path end-points coincide). In many applications relevant to rover navigation, the vehicle's minimum turning radius is large compared to the desired position discretization of the state lattice. The former is defined by the maneuverability of the vehicle's mobility system, while the latter is influenced by a number of factors, most notably the resolution of the available perception system. While the maneuverability of planetary rovers is fairly high, the computation constraints inherently present in this domain disallow very fine perception resolution. In this setting, it is not hard to visualize that the set difference between the canonical path and its alternatives (number of black squares in Figure 8.5.2 will be either small or zero). For example, observe that the lengths of the majority of motions in top and bottom of Figure 8.5.2 are of the same order of magnitude as the minimum turning radius, 2.5 cell widths. This observation explains one of the "lessons-learned" of this effort: retaining the identical endpoints of canonical and alternative paths does not allow on-demand expansion to yield as significant impact on performance as allowing the path endpoints to vary. The only criterion for including paths in a set of alternatives was to retain the same values of final state other than the position dimensions. In particular, we constrained this set to have paths with the same final heading as the canonical path.

Figures 8.5.1, 8.5.3 and 8.5.4 demonstrate experimental results of the algorithm described in the previous section. Following the procedures proposed in Section 5.2.1, 100 different worlds (of size 100x100 cells) with varying single-point obstacle density (5% to 35% density) were generated. Three different planners were used to generate motions to every cell in each of the worlds (resulting in approximately one million motion solutions per density level):

1. A planner featuring standard, car-like set of motion primitives with the graph outdegree that is sufficiently small to enable the planner to run on-board a planetary rover;

2. A planner with an artificially high outdegree (comprising all motions with end-points in a large region around the rover, 13 by 13 cells);

3. A planner enabled with on-demand expansion, where canonical motions were taken from the primitives of Planner 1, and the sets of path alternatives were derived from Planner 2.

Figure 8.5.1 presents plots of relative runtime and cost of Planners 2 and 3, presented as ratios to the respective quantities of Planner 1. The significant result is that the coverage and cost ratios of Planner 3 are very similar to Planner 2, while its computation runtime is over two times lower.

Figure 8.5.4 makes a similar presentation of comparing coverage, number of attainable states, used as a measure of completeness. Figure 8.5.3 is a visual representation of the improved coverage by using on-demand expansion. Blue cells are obstacles, and shades of red represent the number of different headings that are attainable at each cell (dark means greater number of headings).

This research demonstrated that it is possible to generate paths in a recombinant state lattice using a control set whose expressiveness is dependent on the local obstacle density of the environment. This provides a method of exploiting computational complexity of coarse search spaces while gaining some the expressiveness of denser search spaces. In the domain of space/planetary robotics (where risk minimization and resource efficiency are important), such techniques could provide the capability to plan longer distance maneuvers on flight-analog computers.

8.6 MOBILE MANIPULATION

The presented approach was evaluated in the context of base placement with differential constraints in mobile manipulation. This is recognized as a hard problem; the present Section attempts to demonstrate that state lattice motion primitives offer an attractive solution in this context. We present results of three types of experiments. In the first two, the mobile manipulator robot is tasked to perform "point-at" operations: the target pose of the arm's end-effector is given. The third type of experiments illustrates the "pick-and-place" task, where object coordinates are specified (in $SE(3)$), and the planner must compute the arm's end-effector pose that would allow it to grasp the object. Figures 8.6.2 through 8.6.5 describe the experimental setup. Each of the experiment types was repeated 20 times (the only changed variable was the robot's initial pose, selected randomly). The base planner in all experiments was a state lattice planner running A* [51], configured according to base mobility in each experiment. It was enabled with a heuristic look-up table (HLUT) – a well-informed heuristic [109] that used pre-computed free-space costs of all (discretized) motions in the area that spanned the workspace of each experiment.The first two experiment types used the Resolution-Optimal termination condition of the base planner, while the third type used Minimal-Runtime termination condition.

The first two experiment types did not consider arm obstacles (analytical IK was used), while the third experiment type used Dijkstra's search operating in five dimensions of the arm. It is expected that the application of randomized search techniques [13; 14; 87] would yield faster computation time for arm planning in this setting. The experiments were executed on conventional computing hardware: a laptop computer with a dual-core 2GHz CPU and 1GB of RAM. The runtime statistics of all trials are presented in Tables 8.2 and 8.3, followed by a summary discussion.

8.6.1 2D POINTING

This simplified experiment is intended largely as an illustration of the concepts presented earlier. The setup is inspired by the JPL Rocky8 rover, pictured on the left of Figure 8.6.2 [103]. A mobile robot base has six wheels and is capable of point turns. The rover features a 5-DOF PUMA-class arm, although it is simplified in this section as a 2-DOF planar arm for consistency with earlier examples. The robot's "point-at" task is to place the arms' end-effector at the 2D coordinates of the target, indicated with a circle and a cross on the right of Figure 8.6.2. The G_d is also shown, along with workspace obstacles and the motion solution of the planner (red line with dots). The base planner was a state lattice planner with 8-connected grid control set, configured at quarter-resolution of the G_d for clarity. The red dots represent the vertices in the graph path, and red lines are the edges.

8.6.2 3D POINTING

A more realistic set of experiments in this sub-section focuses on another "point-at" task, where the arm end-effector target is specified in $SE(2)$. This type of experiments builds on that in the previous sub-section and extends it in three ways. First, the base is subjected to car-like kinematics constraints. Second, the arm is enabled with the third, wrist, joint; this arm is similar to Rocky8's, except that shoulder-yaw and wrist-yaw joints are omitted for clarity of 2D illustration. Third, the manipulation cost is defined in terms of application of a force along the specified vector in 2D. This setup models the task of drilling or riveting in construction applications, or the application of a rock abrasion tool for mineral composition experiments in planetary exploration. While the dynamics of these operations is not considered explicitly, the manipulation cost has been configured to represent a heuristic that the maximum application of force is along the longitudinal dimension of the vehicle.

This places a premium on base poses that face the end-effector target (a string of black cells leading to the target, shown as a circle and a cross in the right half of Figure 8.6.3). The obstacle in front of the target (set of white cells) is considered an impediment to the base only; the arm "goes over" the obstacle to perform its task – an acceptable solution in this setup.

8.6.3 Placing I-Beams in Autonomous Construction

The final set of experiments illustrate "pick-and-place" operation (Figures 8.6.4 and 8.6.5). The robot's task is to install I-beams at a construction site that is too large and constrained to allow the use of a fixed manipulator workcell. The arm end-effector pose is not specified explicitly; instead $SE(3)$ pose of the I-beam is given, and the robot must compute a grasp. In our experiments, we used a heuristic that the grasp should be at the mid-point of the beam, oriented along its height. Alternative grasp selection methods may be utilized [13]. The arm is a PUMA-class, 5-DOF manipulator with a 1-DOF gripper. The arm is installed on the trailer of a mobile robot with tractor-trailer mobility. The system must operate in narrow passages among obstacles.

The robot entered the area shown via a side entrance while carrying the beam in a stowed arm configuration: the shoulder (longest) link was oriented along the longitudinal axis of the vehicle, and the elbow link was folded in. The base planner computed the footprint of the vehicle carrying the beam and used it to find a collision-free path (orange dots in the figure). Once the base arrived at the chosen base placement pose, the elbow link unfolded, while the shoulder joint kept the shoulder link at sufficiently low height to allow the arm to fit between the long yellow beams and place the I-beam into its desired pose.

Table 8.2 presents the statistics of computational runtimes of the "point-at" experiments. The figures are mean runtimes over all runs (in seconds); standard deviation figures are in parentheses. Both types of experiments demonstrate fast computation due to the benefit of using free-space, analytical IK for arm planning and a lattice planner equipped with a well-informed HLUT heuristic for base planning. This result is significant because it demonstrates computing a resolution-optimal, mobile manipulator motion under nonholonomic constraints for a realistic "point-at" task, e.g. drilling or riveting in autonomous construction, among arbitrary base obstacles in only a few seconds. The result is likely to enable fielded solutions of this type.

Table 8.3 presents the statistics of "pick-and-place" experiments. We separated the runtimes for

Table 8.2: Experimental Planning Runtime: *point-at* (seconds)

2D pointing	3D pointing
0.37 (0.22)	1.76 (0.37)

Table 8.3: Experimental Planning Runtime: "pick-and-place" (seconds)

	Arm only	Base + Cost sum min.
Serial	18.31 (5.46)	2.41 (0.47)
Parallel	12.76 (4.28)	2.87 (0.39)

the Dijkstra's arm planner (first column) and the rest of the system. The total runtime of the whole system is the sum of the two numbers. The separation illustrates that nearly 90% of the runtime was due to the arm planner. This observation is significant because it identifies arm planning as the unique limiting component. Any progress in improving the runtime of this component will drive down the runtime of the whole mobile manipulation planning system.

The first row of the table presents the runtimes when the mobile manipulation was executed in a serial fashion on the single core of the CPU. The second row presents analogous figures when the computation was parallelized between the two cores of the CPU. Since the bulk of the computation was arm planning, one of the cores executed one half of the arm planning computations, while the other core executed the other half, as well as the rest of the planning system. The average runtime reduction is not two-fold due to the overhead of distributing the computation, yet it does demonstrate a substantial (over 40%) reduction in runtime. To our knowledge, this is the first result of this type in the given domain. Employing a greater number of parallel processes is likely to improve the efficiency further, thereby potentially enabling increased adaptation of mobile manipulator systems in a variety of applications.

8.6.4 DISCUSSION

In this section, we explored motion planning for mobile manipulator robots. We considered a class of mobile manipulation problems that admit decoupled mobile manipulation. We proposed task-

space decomposition that leverages the decoupled nature of this class of problems and yields solutions with greater efficiency than is attainable by considering the original problem. Furthermore, the components of the decomposition – base and arm planners – are explicitly coupled via user-defined manipulation cost, which allows attaining mobile manipulator motions that are resolution-optimal with respect to that cost. In addition, the approach admits the flexibility of employing a choice of component planners, including those that satisfy nonholonomic constraints. Application of a certain type of base planners allows parallelizing mobile manipulation planning. Experimental results in simulation demonstrate that the proposed task-space decomposition is competitive with respect to the state of the art.

In addition, it would also be interesting to validate the approach on physical robots. We would also like to explore other methods of achieving dimensionality reduction in this setting, as well as finding ways of performing a greater amount of off-line pre-computation in order to gain further efficiency in mobile manipulation planning.

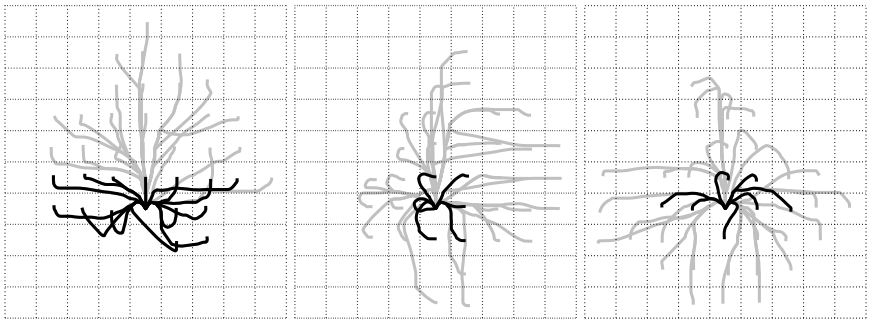


Figure 8.4.2: **Control set regression**. Several subsets of primitives, generated for the given system, are shown; abundance of high curvature is due to extreme dynamics. Top and bottom rows include primitives at low and high velocity, resp. The columns show the primitives with final headings 0°, 90° and 180° (left to right). Gray motions have been pruned, as concatenations of black motions can replicate them within specified cost increase threshold.

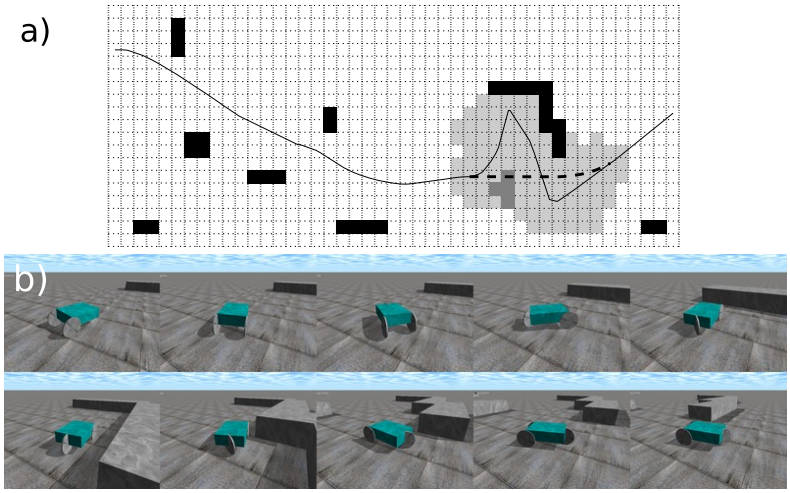
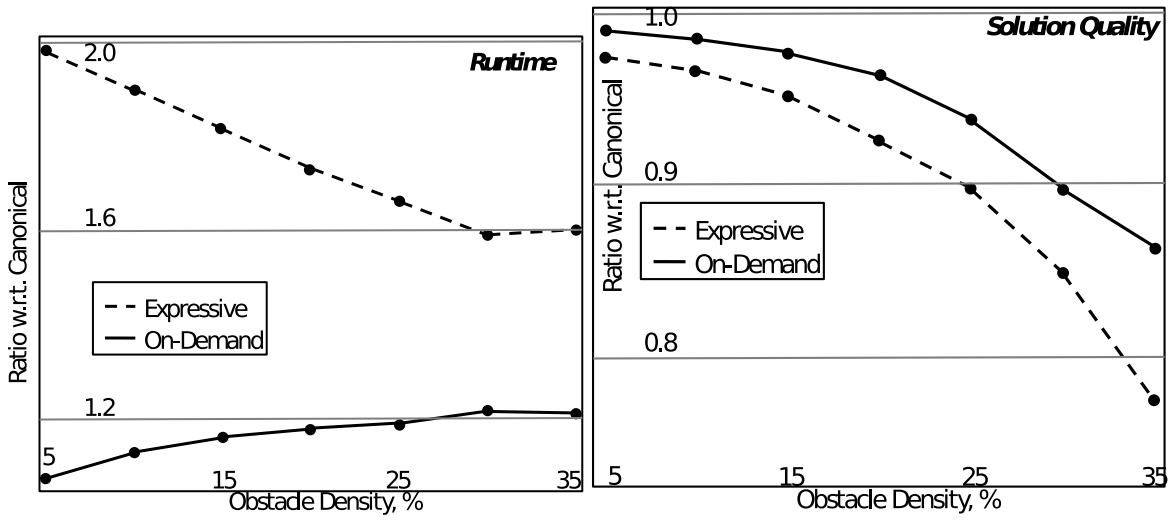
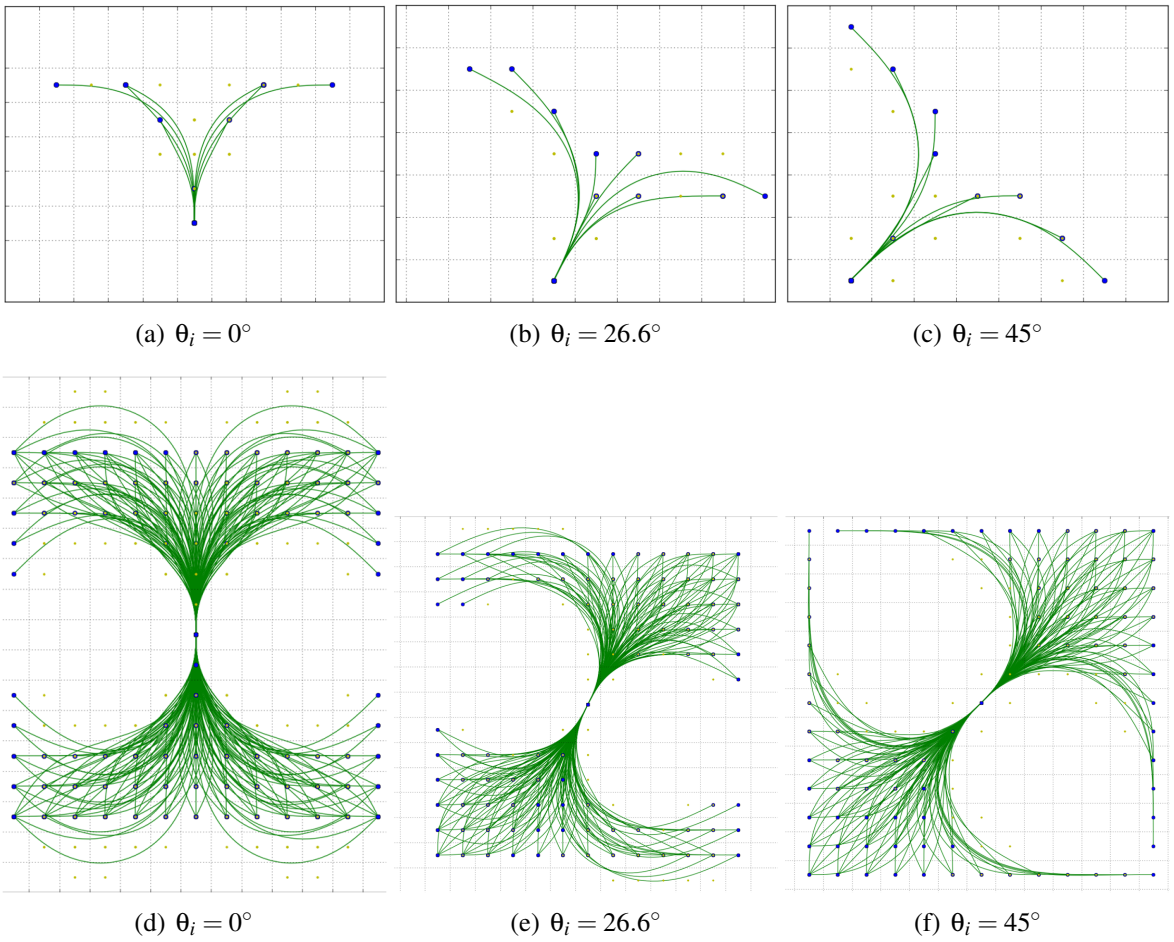


Figure 8.4.3: **Kinodynamic incremental planning**. Robot is avoiding a number of obstacles (black cells), while traveling at high speed on slippery surface. A new obstacle (dark gray cells) is discovered and invalidates a segment of the previous trajectory (dotted line). D* repairs the path by expanding states (light gray) only in the affected region.



(a) Relative Runtime (b) Relative Cost

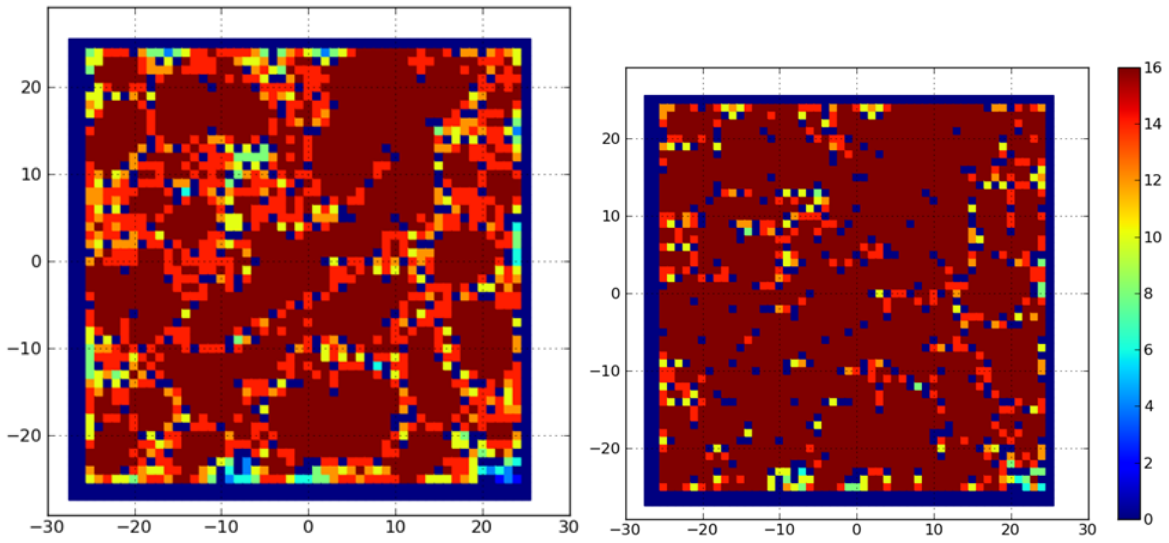
Figure 8.5.1: **On-demand expansion performance**. Simulation experiments evaluating the performance of canonical, expressive, and on-demand expansion control sets. Parts a) and b) compare the relative runtime and solution cost, respectively, of expressive and on-demand expansion-based motion sets with respect to canonical motion sets (with varying obstacle density). Although the expressive motion set unsurprisingly produces the safest paths, almost the same quality of motions is achieved by the on-demand expansion-based motion set while consuming significantly less computational resources.



(a) $\theta_i = 0°$ (b) $\theta_i = 26.6°$ (c) $\theta_i = 45°$

(d) $\theta_i = 0°$ (e) $\theta_i = 26.6°$ (f) $\theta_i = 45°$

Figure 8.5.2: **Canonical and expressive control sets**. Top row: a set of canonical motion primitives that represents the minimal expressiveness, yet yields fast planning due to its small size. Bottom row: the other end of the spectrum: high expressiveness, yet longer planning runtime.



(a) Canonical (b) On-demand Exp.

Figure 8.5.3: **Coverage pseudo-color**. The figure illustrates the capacity of the robot to move to locations that are hard to achieve due to dense obstacles. The pseudo-color represents the number of different headings that are attainable for each cell in the environment using the canonical template (left) and on-demand expansion (right). Shades of red denote the number of attainable headings (deep red represents all desired headings), while blue cells are obstacles.

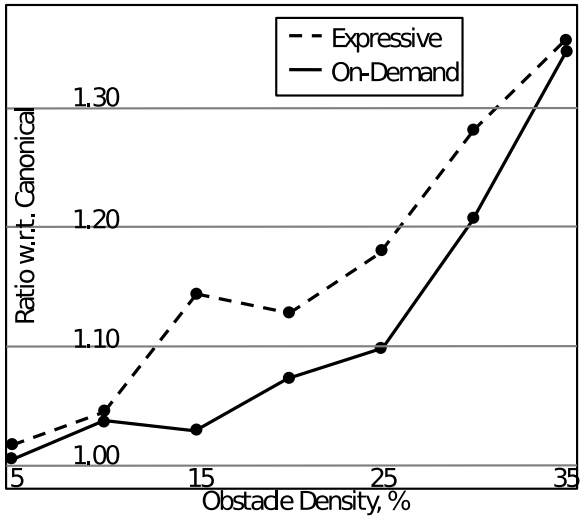


Figure 8.5.4: **Relative coverage**. A plot of the number of attainable states using Planners 2 and 3, represented as the ratios to the corresponding quantities of Planner 1. The plot demonstrates that on-demand expansion (solid line) features a similar improvement over the canonical motion set as the expressive motion set, while allowing significantly lower computation cost, as suggested in Figure 2a).

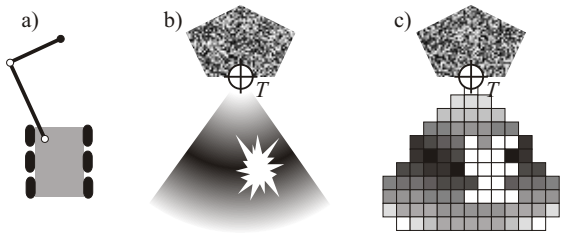


Figure 8.6.1: **Discretized goal region**. Figure a) shows a two-link planar arm installed on a mobile base. Figure b) illustrates a subset of the goal region G induced by $\rho(\bullet)$, where grayscale intensity corresponds to its value (white color indicates $\rho(\bullet) = \infty$). An odd-shaped obstacle is visible near the manipulation target T. Figure c) shows the discretized version of the goal region, G_d. The obstacle causes a set of cells to be white – infeasible poses for the base joint of the arm.

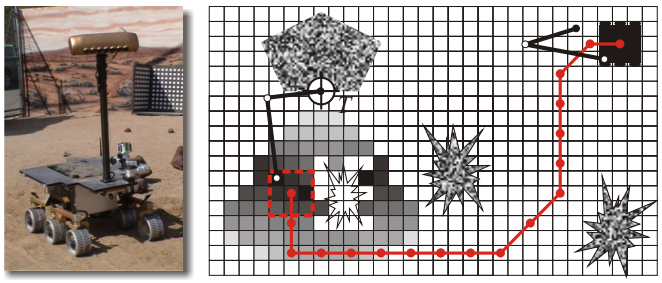


Figure 8.6.2: **2D *point-at* task**. Left subfigure shows the Rocky8 rover that inspired this and the following subsection (courtesy of JPL [103]). The figure on the right illustrates mobile manipulation in the scenario where the robot moves from top right of the figure (arm is stowed initially) to the pose that maximizes manipulability. Note that the base joint of the arm lands in the black cell, indicating minimal manipulation cost). The red line with dots is the motion of the base, computed by the planner. The dots are the vertices of the graph (elaborated by the planner), and the lines are the edges. The motion represents the resolution-optimal trade-off between path cost (length) and manipulation cost (manipulability) automatically made by the planner.

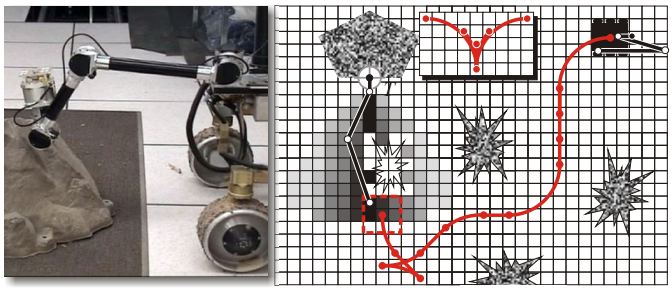


Figure 8.6.3: **3D *point-at* task**. Left subfigure is intended to motivate this class of manipulation tasks; it shows a close-up view of the Rocky8 rover's arm performing a rock imaging study, common in planetary exploration (courtesy of JPL [103]). The figure on the right illustrates a more realistic set of experiments that extend Figure 8.6.2. The initial pose of the rover is shown in top right (with arm stowed). The inset shows the control set utilized by the state lattice nonholonomic planner [109]; an identical set of controls for reverse motion is omitted for conciseness. The red line with dots represents the solution of the planner, where the dots are the vertices of the graph, elaborated by the planner, and the lines are the edges. Similar to Figure 8.6.2, the motion represents the trade-off between path cost (length) and manipulation cost (manipulability), automatically made by the planner.

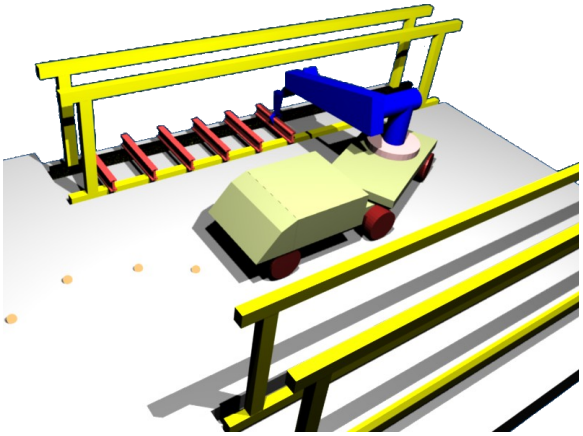


Figure 8.6.4: **Placing I-beams in autonomous construction**. The proposed approach is well-suited for "pick-and-place" and other common manipulation tasks. In this example, the robot's task is to install I-beams at a construction site that is too large and constrained to allow the use of a fixed manipulator workcell. The motion, computed by the planner, represents a resolution-optimal trade-off between path length, traversed by the tractor-trailer platform, and the subsequent manipulator operation.

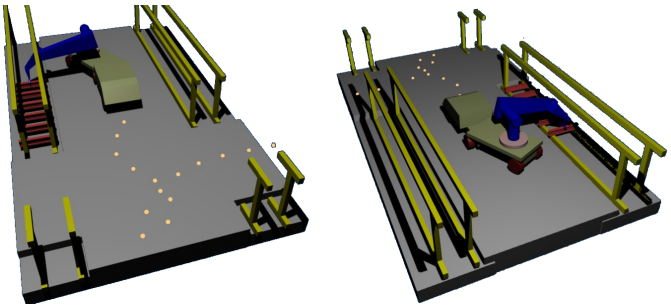


Figure 8.6.5: **A *pick-and-place* experiment**. The robot's task is to install red I-beams between long yellow beams at a construction site. A 5-DOF, PUMA-class arm with a 1-DOF gripper is installed on the trailer of a mobile robot with tractor-trailer mobility. The system must operate in narrow passages among obstacles. The figure shows the base placement of the robot computed by the planner, given the desired pose for placing the I-beam object. Orange dots show the base planner solution, the trajectory of the tractor robot. Left and right subfigures show different viewpoints of the scene.

CHAPTER 9

CONCLUSIONS

This work has been motivated by a fairly acute need to endow our field robots with sufficient understanding of their own mobility to allow them to efficiently plan correct and intricate maneuvers in response to their challenging surroundings.

This thesis proposed an approach of encapsulating the complexity of differential constraint satisfaction via a novel type of structured, pre-computed controls that serve as motion primitives. The contribution of this work is in developing a general approach to constructing such motion primitives, given a model of robot mobility.

Moreover, the proposed type of motion primitives allows an unprecedented amount of pre-computation in the area of differentially constrained motion planning, which yields a dramatic increase in planning efficiency even for systems with complex kinematics and dynamics. Lastly, the proposed motion primitives are fully compatible with a wide range of planning algorithms and allow such useful techniques as incremental planning and bi-directional search to be used in the

context of planning with differential constraints.

In conclusion, the primary contribution of this work is not a motion planner, but a principled, automated mechanism to design an efficient, precision, differentially constrained search space upon which any planner may operate. A compelling case was presented regarding the benefits of the approach with respect to the prevailing alternatives. The proposed method has been demonstrated and validated on a number of relevant systems, both in simulation and in real experiments. This work promises to enable capable and reliable motion planners with differential constraints, as encountered in many realistic robot systems with practical utility, operating efficiently in cluttered, partially known environments.

BIBLIOGRAPHY

[1] Agarwal, Aronov, and Sharir. Motion planning for a convex polygon in a polygonal environment. *GEOMETRY: Discrete and Computational Geometry*, 22, 1999.

[2] P. Agarwal, N. Amenta, B. Aronov, and M. Sharir. Largest placements and motion planning of a convex polygon. In *Proc. 2nd Annu. Workshop Algorithmic Foundations of Robotics*, Toulouse, France, 1996.

[3] H. Alt, R. Fleischer, M. Kaufmann, K. Mehlhorn, S. Naher, S. Schirra, and C. Uhrig. Approximate motion planning and the complexity of the boundary of the union of simple geometric figures. In *Proc. of the ACM Symposium on Computational Geometry*, pages 281–289, 1990.

[4] R. Alterovitz, K. Goldberg, J. Pouliot, R. Taschereau, and I-Chow Hsu. Sensorless planning for medical needle insertion procedures. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, volume 4, pages 3337–3343, 2003. doi: 10.1109/IROS.2003.1249671.

[5] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. Conf. IEEE Int Robotics and Automation*, volume 1, pages 113–120, 1996. doi: 10.1109/ROBOT.1996.503582.

[6] D. A. Anisi, J. Hamberg, and X. Hu. Nearly time-optimal paths for a ground vehicle. *Journal of Control Theory and Applications*, 2003.

[7] C. Baker, A. Morris, D. Ferguson, S. Thayer, C. Whittaker, Z. Omohundro, C. Reverte, W. Whittaker, D. Hahnel, and S. Thrun. A campaign in autonomous mine mapping. In *Proc. of the IEEE Conference on Robotics and Automation*, 2004.

[8] J. Barraquand and J.-C. Latombe. A monte-carlo algorithm for path planning with many degrees of freedom. *Proc. of the IEEE International Conference on Robotics and Automation*, pages 1712–1717, 1990.

[9] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: controllability and motion planning in the presence of obstacles. *Proc. of the IEEE International Conference on Robotics and Automation*, 1991.

[10] J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for robot path planning. In G. Giralt and G. Hirzinger, editors, *Proc. of the 7th International Symposium on Robotics Research*, pages 249–264. Springer, New York, NY., 1996.

[11] Jerome Barraquand and Jean-Claude Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2-4): 121–155, 10 1993.

[12] K. E. Bekris and L. E. Kavraki. Greedy but safe replanning under kinodynamic constraints. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 704–710, 2007. doi: 10.1109/ROBOT.2007.363069.

[13] Dmitry Berenson, Joel Chestnutt, Siddhartha S. Srinivasa, James J. Kuffner, and Satoshi Kagami. Pose-constrained whole-body planning using task space region chains. In *Humanoids*, 2009.

[14] Dominik Bertram, James Kuffner, Ruediger Dillmann, and Tamim Asfour. An integrated approach to inverse kinematics and path planning for redundant manipulators. In *ICRA*, 2006.

[15] A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. *Hybrid Systems: Computation and Control (Lecture Notes in Computer Science, no. 2993)*, pages 67–78, 2004.

[16] A. Bicchi, A. Marigo, and B. Piccoli. On the reachability of quantized control systems. *IEEE Transactions on Automatic Control*, 47(4):546–563, 2002.

[17] R. Bohlin. Path planning in practice; lazy evaluation on a multi-resolution grid. *Proc. of the IEEE/RSJ International Conference on Intelligent Robots & Systems*, 2001.

[18] R. Bohlin. Path planning in practice; lazy evaluation on a multi-resolution grid. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2001.

[19] R. Bohlin and L. Kavraki. Path planning using lazy PRM. *Proc. of the IEEE International Conference on Robotics & Automation*, 2000.

[20] M. S. Branicky, R. A. Knepper, and J. J. Kuffner. Path and trajectory diversity: Theory and algorithms. In *Proc. IEEE International Conference on Robotics and Automation ICRA 2008*, pages 1359–1364, 2008. doi: 10.1109/ROBOT.2008.4543392.

[21] M.S. Branicky, S.M. LaValle, S. Olson, and L. Yang. Quasi-randomized path planning. *Proc. of the International Conference on Robotics and Automation*, 2001.

[22] J. Canny, J. Reif, B. Donald, and P. Xavier. On the complexity of kinodynamic planning. In *Proc. th Annual Symp. Foundations of Computer Science*, pages 306–316, 1988. doi: 10.1109/SFCS.1988.21947.

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

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

[25] A. Casal. *Reconfiguration planning for modular self-reconfigurable robots*. PhD thesis, Aeronautics and Astronautics Dept., Stanford U., 2001.

[26] Peng Cheng and S. M. LaValle. Resolution complete rapidly-exploring random trees. In *Proc. IEEE Int. Conf. Robotics and Automation*, volume 1, pages 267–272, 2002. doi: 10.1109/ROBOT.2002.1013372.

[27] M. Cherif. Kinodynamic motion planning for all-terrain wheeled vehicles. In *Proc. of the IEEE International Conference on Robotics & Automation*, 1999.

[28] B. Donald and P. Xavier. Near-optimal kinodynamic planning for robots with coupled dynamics bounds. In *Proc. Symp. IEEE Int Intelligent Control*, pages 354–359, 1989. doi: 10.1109/ISIC.1989.238674.

[29] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *J. ACM*, 40(5):1048–1066, 1993. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/174147.174150.

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

[31] L. Erickson and S. LaValle. Survivability: Measuring and ensuring path diversity. In *Proc. of the IEEE International Conference on Robotics and Automation*, 2009.

[32] L. H. Erickson and S. M. LaValle. Survivability: Measuring and ensuring path diversity. In *Proc. IEEE International Conference on Robotics and Automation ICRA '09*, pages 2068–2073, 2009. doi: 10.1109/ROBOT.2009.5152773.

[33] Lawrence H. Erickson and Steven M. LaValle. Survivability: measuring and ensuring path diversity. In *ICRA'09: Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 3749–3754, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-2788-8.

[34] Xiuyi Fan, Surya Singh, Florian Oppolzer, Eric Nettleton, Ross Hennessy, Alexander Lowe, and Hugh Durrant-Whyte. Integrated planning and control of large tracked vehicles in open terrain. In *Proceedings of the International Conference on Robotics and Automation*, 2010.

[35] D. Ferguson and A. Stentz. The delayed d* algorithm for efficient path replanning. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 2045–2050, April 18–22, 2005.

[36] D. Ferguson and A. Stentz. Multi-resolution Field D*. In *Proc. International Conference on Intelligent Autonomous Systems (IAS)*, 2006.

[37] C. Fernandes, L. Gurvits, and Z. X. Li. A variational approach to optimal nonholonomic moton planning. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 680–685, 1991.

[38] Jonathan Fink, Nathan Michael, Soonkyum Kim, and Vijay Kumar. Planning and control for cooperative manipulation and transportation with aerial robots. *International Journal of Robotics Research*, 30(3):324–334, 2011.

[39] S. Fleury, P. Soueres, J.-P. Laumond, and R. Chatila. Primitives for smoothing mobile robot trajectories. *IEEE Transactions on Robotics and Automation*, 11(3):441–448, 1995. doi: 10.1109/70.388788.

[40] T. Fraichard and J.-M. Ahuactzin. Smooth path planning for cars. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3722–3727, 2001.

[41] T. Fraichard and A. Scheuer. From reeds and shepp's to continuous-curvature paths. *IEEE Transactions on Robotics*, 20(6):1025–1035, December 2004. doi: 10.1109/TRO.2004.833789.

[42] T. Fraichard and A. Scheuer. From Reeds and Shepp's to continuous-curvature paths. *IEEE Transactions on Robotics*, 20(6):1025–1035, 2004.

[43] E. Frazzoli, M.A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. In *Proc. of the American Control Conference*, 2001.

[44] Emilio Frazzoli, Munther A. Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS*, 25(1), 2002.

[45] Francois Gaillard, Michael Soulignac, Cedric Dinont, and Philippe Mathieu. Deterministic kinodynamic planning with hardware demonstrations. In *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, 2011.

[46] Jared Go, Thuc D. Vu, and James J. Kuffner. Autonomous behaviors for interactive vehicle animations. *Graph. Models*, 68(2):90–112, 2006. ISSN 1524-0703. doi: http://dx.doi.org/10.1016/j.gmod.2005.04.003.

[47] S. Goldberg, M. Maimone, and L. Matthies. Stereo vision and rover navigation software for planetary exploration. In *Proc. of IEEE Aerospace Conference*, 2002.

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

[49] Colin Green and Alonzo Kelly. Toward optimal sampling in the space of paths. In *13th International Symposium of Robotics Research*, 2007.

[50] J.H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, December 1960.

[51] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. ISSN 0536-1567. doi: 10.1109/TSSC.1968.300136.

[52] Thomas Howard. *Adaptive Model-Predictive Motion Planning for Navigation in Complex Environments*. PhD thesis, Carnegie Mellon University, August 2009.

[53] T.M. Howard and A. Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *International Journal of Robotics Research*, 26(2):141–166, 2007.

[54] D. Hsu. *Randomized single-query motion planning in expansive spaces*. PhD thesis, Computer Science Dept., Stanford University, 2000.

[55] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. Conf. IEEE Int Robotics and Automation*, volume 3, pages 2719–2726, 1997. doi: 10.1109/ROBOT.1997.619371.

[56] Terry Huntsberger and Gail Woodward. Intelligent autonomy for unmanned surface and underwater vehicles. In *Proc. OCEANS*, pages 1–10, 2011.

[57] Robert J. Webster III, Jin Seob Kim, Noah J. Cowan, Gregory S. Chirikjian, and Allison M. Okamura. Nonholonomic modeling of needle steering. *International Journal of Robotics Research*, 25(5-6):509–525, 5 2006.

[58] Leonard Jaillet, Judy Hoffman, Jur van den Berg, Pieter Abbeel, Josep M. Porta, and Ken Goldberg. Eg-rrt: Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles. In *Proc. IEEE/RSJ Int Intelligent Robots and Systems (IROS) Conf*, pages 2646–2652, 2011. doi: 10.1109/IROS.2011.6094802.

[59] F. Jean. Complexity of nonholonomic motion planning. *International Journal of Control*, 74(8):776–782, 2001.

[60] Sören Kammel, Julius Ziegler, Benjamin Pitzer, Moritz Werling, Tobias Gindele, Daniel Jagzent, Joachim Schader, Michael Thuy, Matthias Goebl, Felix von Hundelshausen, Oliver Pink, Christian Frese, and Christoph Stiller. Team annieway's autonomous system for the darpa urban challenge 2007. *Journal of Field Robotics*, 25(9):615 – 639, 8 2008.

[61] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. ISSN 1042-296X. doi: 10.1109/70.508439.

[62] L.E. Kavraki. *Random networks in configuration space for fast path planning*. PhD thesis, Computer Science Dept., Stanford University, 1994.

[63] A. Kelly and B. Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *International Journal of Robotics Research*, 22(7/8):583–601, 2002.

[64] A. Kelly, O. Amidi, M. Happold, H. Herman, T. Pilarski, P. Rander, A. Stentz, N. Vallidis, and R. Warner. Toward reliable off-road autonomous vehicle operating in challenging environments. In *Proc. of the International Symposium on Experimental Robotics*, 2004.

[65] Alonzo Kelly. *An Intelligent Predictive Control Approach to the High-Speed Cross-Country Autonomous Navigation Problem*. PhD thesis, Carnegie Mellon University, September 1995.

[66] Alonzo Kelly, Anthony Stentz, Omead Amidi, Mike Bode, David Bradley, Antonio Diaz-Calderon, Mike Happold, Herman Herman, Robert Mandelbaum, Tom Pilarski, Pete Rander, Scott Thayer, Nick Vallidis, and Randy Warner. Toward reliable off road autonomous vehicles operating in challenging environments. *Int. J. Rob. Res.*, 25(5-6):449–483, 2006. ISSN 0278-3649. doi: http://dx.doi.org/10.1177/0278364906065543.

[67] M. Khatib, H. Jaouni, R. Chatila, and J. P. Laumond. Dynamic path modification for car-like nonholonomic mobile robots. In *Proc. Conf. IEEE Int Robotics and Automation*, volume 4, pages 2920–2925, 1997. doi: 10.1109/ROBOT.1997.606730.

[68] R. Kindel. *Motion planning for free-flying robots in dynamic and uncertain environments*. PhD thesis, Aeronaut. & Astr. Dept., Stanford University, 2001.

[69] R. A. Knepper and A. Kelly. High performance state lattice planning using heuristic look-up tables. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3375–3380, 2006. doi: 10.1109/IROS.2006.282515.

[70] S. Koenig and M. Likhachev. D* Lite. In *Proceedings of the AAAI Conference of Artificial Intelligence (AAAI)*, 2002.

[71] Sven Koenig and Maxim Likhachev. D*-Lite. In *Eighteenth national conference on Artificial intelligence*, pages 476–483, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. ISBN 0-262-51129-0.

[72] J. Zico Kolter, Christian Plagemann, David T. Jackson, Andrew Y. Ng, and Sebastian Thrun. A probabilistic approach to mixed open-loop and closed-loop control, with application to extreme autonomous driving. In *Proceedings of the International Conference on Robotics and Automation*, 2010.

[73] James J. Kuffner and Steven M. LaValle. Space-filling trees: A new perspective on incremental search for motion planning. In *Proc. IEEE/RSJ Int Intelligent Robots and Systems (IROS) Conf*, pages 2199–2206, 2011. doi: 10.1109/IROS.2011.6094740.

[74] J.J. Kuffner. *Autonomous agents for real-time animation*. PhD thesis, Computer Science Dept., Stanford University, 1999.

[75] Jr. Kuffner, J. J. and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int. Conf. Robotics and Automation ICRA '00*, volume 2, pages 995–1001, 2000. doi: 10.1109/ROBOT.2000.844730.

[76] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. P. How, and G. Fiore. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009. doi: 10.1109/TCST.2008.2012116.

[77] A. Lacaze, Y. Moscovitz, N. DeClaris, and K. Murphy. Path planning for autonomous vehicles driving over rough terrain. In *Proceedings of the IEEE International Symposium on Intelligent Control*, 1998.

[78] F. Lamiraux and J.-P. Laumond. Smooth motion planning for car-like vehicles. *IEEE Transactions on Robotics and Automation*, 17(4), 2001.

[79] J.-C. Latombe. *Robot Motion Planning*. Kluwer, Boston, 1991.

[80] Manfred Lau and James J. Kuffner. Behavior planning for character animation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 271–280, New York, NY, USA, 2005. ACM. ISBN 1-7695-2270-X. doi: http://doi.acm.org/10.1145/1073368.1073408.

[81] J.-P. Laumond, S. Sekhavat, and F. Lamiraux. Guidelines in nonholonomic motion planning. *Robot motion planning and control*, 1998.

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

[83] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2001.

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

[85] S.M. LaValle, M. Branicky, and S. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *International Journal of Robotics Research*, 23(7/8): 673–692, 2004.

[86] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[87] Steven M. LaValle and Jr. James J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[88] Steven M. LaValle, Michael S. Branicky, and Stephen R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *International Journal of Robotics Research*, 23(7-8):673–692, 2004.

[89] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *Int. J. Rob. Res.*, 28(8):933–945, 2009. ISSN 0278-3649. doi: http://dx.doi.org/10.1177/0278364909340445.

[90] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *International Journal of Robotics Research*, 28(8):933–945, 8 2009.

[91] Maxim Likhachev and Anthony Stentz. R* search. In *AAAI*, 2008.

[92] S. R. Lindemann and S. M. LaValle. Multiresolution approach for motion planning under differential constraints. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 139–144, 2006. doi: 10.1109/ROBOT.2006.1641174.

[93] T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[94] T. Lozano-Perez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[95] K. Mombaur, J.-P. Laumond, and E. Yoshida. An optimal control model unifying holonomic and nonholonomic walking. In *Proc. 8th IEEE-RAS Int. Conf. Humanoid Robots Humanoids*, pages 646–653, 2008. doi: 10.1109/ICHR.2008.4756020.

[96] A. Morris, D. Silver, D. Ferguson, and S. Thayer. Towards topological exploration of abandoned mines. In *Proc. of the IEEE International Conference on Robotics*, 2005.

[97] R. M. Murray and S. S. Sastry. Steering nonholonomic systems in chained form. In *Proc. 30th IEEE Conf. Decision and Control*, pages 1121–1126, 1991. doi: 10.1109/CDC.1991.261508.

[98] R. M. Murray and S. S. Sastry. Nonholonomic motion planning: steering using sinusoids. *IEEE Transactions on Automatic Control*, 38(5):700–716, May 1993. doi: 10.1109/9.277235.

[99] R. M. Murray, D. C. Deno, K. S. J. Pister, and S. S. Sastry. Control primitives for robot systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(1):183–193, 1992. doi: 10.1109/21.141324.

[100] B. K. Natarajan. The complexity of fine motion planning. *International Journal of Robotics Research*, 7(2):36–42, 1988.

[101] Harald Niederreiter. *Quasi-Monte Carlo Methods*. SIAM, 1992.

[102] Colm Ó-Dúnlaing. Motion planning with inertial constraints. *Algorithmica*, 2:431–475, 1987.

[103] California Institute of Technology, Jet Propulsion Laboratory, and Robotics Section: http://www robotics.jpl.nasa.gov/systems/system.cfm?System=3, 2012. URL `http://www-robotics.jpl.nasa.gov/systems/system.cfm?System=3`.

[104] N. Nilsson P. Hart and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[105] D.K. Pai and L.-M. Reissell. Multiresolution rough terrain motion planning. *IEEE Transactions on Robotics and Automation*, 14(1):19–33, 1998.

[106] S. Pancanti, L. Pallottino, and A. Bicchi. Motion planning through symbols and lattices. In *Proc. of the Int. Conf. on Robotics and Automation*, 2004.

[107] S. Pancanti, L. Pallottino, D. Salvadorini, and A. Bicchi. Motion planning through symbols and lattices. In *Proc. IEEE Int. Conf. Robotics and Automation ICRA '04*, volume 4, pages 3914–3919, 2004.

[108] J. Pearl. *Heuristics*. Addison Wesley, Boston, MA, 1984.

[109] Mihail Pivtoraiko and Alonzo Kelly. Constrained motion planning in discrete state spaces. In *Field and Service Robotics*, pages 269–280, 2005.

[110] Mihail Pivtoraiko and Alonzo Kelly. Generating near-minimal spanning control sets for constrained motion planning in discrete state spaces. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3231–3237, August 2005. doi: 10.1109/IROS.2005.1545046.

[111] Mihail Pivtoraiko and Alonzo Kelly. Differentially constrained motion replanning using state lattices with graduated fidelity. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2611–2616, September 2008. doi: 10.1109/IROS.2008.4651220.

[112] Mihail Pivtoraiko and Alonzo Kelly. Kinodynamic motion planning with state lattice motion primitives. In *Proc. IEEE/RSJ Int Intelligent Robots and Systems (IROS) Conf*, pages 2172–2179, 2011. doi: 10.1109/IROS.2011.6094900.

[113] Mihail Pivtoraiko, Thomas Howard, Issa A.D. Nesnas, and Alonzo Kelly. Field experiments in rover navigation via model-based trajectory generation and nonholonomic motion planning in state lattices. In *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 2008.

[114] Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009. ISSN 1556-4959. doi: 10.1002/rob.v26:3.

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

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

[117] H. Royden. *Real Analysis*. Collier-MacMillman Limited, London, 1988.

[118] Martin Rufli and Roland Siegwart. On the design of deformable input- / state-lattice graphs. In *Proceedings of the International Conference on Robotics and Automation*, 2010.

[119] Martin Rufli, Dave Ferguson, and Roland Siegwart. Smooth path planning in constrained environments. In *Proc. IEEE Int. Conf. Robotics and Automation ICRA '09*, pages 3780–3785, 2009. doi: 10.1109/ROBOT.2009.5152506.

[120] G. Sanchez and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Proc. of International Symposium on Robotics Research*, 2001.

[121] G. Sanchez and J.-C. Latombe. On delaying collision checking in prm planning: Application to multi-robot coordination. *International Journal of Robotics Research*, 21(1):5–26, 2002.

[122] A. Scheuer and Th. Fraichard. Continuous-curvature path planning for car-like vehicles. In *Proc. IEEE/RSJ Int Intelligent Robots and Systems IROS '97. Conf*, volume 2, pages 997–1003, 1997. doi: 10.1109/IROS.1997.655130.

[123] A. Scheuer and Ch. Laugier. Planning sub-optimal and continuous-curvature paths for car-like robots. In *Proc. of the International Conference on Robotics and Automation*, 1998.

[124] J. Scholz, S. Chitta, B. Marthi, and M. Likhachev. Cart pushing with a mobile manipulation system: Towards navigation with moveable objects. In *Proc. IEEE Int Robotics and Automation*, pages 6115–6120, 2011. doi: 10.1109/ICRA.2011.5980288.

[125] I.H. Sloan and S. Joe. *Lattice Methods for Multiple Integration*. Clarendon Press, 1994.

[126] L.M. Sonneborn and F.S. Van Vleck. The bang-bang principle for linear control systems. *J. Soc. Indus. and Appl. Math.*, 2(2):151–159, 1965.

[127] O. J. Sordalen. Conversion of the kinematics of a car with n trailers into a chained form. In *Proc. Conf. IEEE Int Robotics and Automation*, pages 382–387, 1993. doi: 10.1109/ROBOT.1993.292011.

[128] P. Soueres and J.D. Boissonnat. *Robot Motion Planning and Control*, chapter Optimal Trajectories for Nonholonomic Mobile Robots, pages 93–169. Springer, 1998.

[129] A. Stentz and M. Hebert. A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots*, 2(2):127–145, 1995.

[130] Anthony Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, August 1995.

[131] Yuval Tassa, Tom Erez, and William D. Smart. Receding horizon differential dynamic programming. In *Proceedings of the Neural Information Processing Systems Conference*, 2007.

[132] Shu Tezuka. *Uniform Random Numbers: Theory and Practice*. Kluwer Academic Publishers, 1995.

[133] Paul Tompkins. *Mission-Directed Path Planning for Planetary Rover Exploration*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2005.

[134] A. Yahja, A. Stentz, S. Singh, and B. L. Brumitt. Framed-quadtree path planning for mobile robots operating in sparse environments. In *Proc. of the IEEE International Conference on Robotics and Automation*, volume 1, pages 650–655, 1998. doi: 10.1109/ROBOT.1998.677046.

[135] Dmitry Yershov and Steven LaValle. Sufficient conditions for the existence of resolution complete planning algorithms. *ALGORITHMIC FOUNDATIONS OF ROBOTICS IX*, 68: 303–320, 2011.

[136] Jun young Kwak, Mihail Pivtoraiko, and Reid Simmons. Combining cost and reliability for rough terrain navigation. In *9th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2008.