RICE UNIVERSITY

Motion Planning for Physical Simulation

by

Andrew M. Ladd

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:

Lydia E. Kavraki, Professor, Chair
Computer Science

Joe Warren, Professor
Computer Science

Richard Baraniuk, Professor
Electrical and Computer Engineering

HOUSTON, TEXAS

DECEMBER 2006

ABSTRACT

Motion Planning for Physical Simulation

by

Andrew M. Ladd

Motion planning research has been successful in developing planning algorithms which are effective for solving problems with complicated geometric and kinematic constraints. Various applications in robotics and in other fields demand additional physical realism. Some progress has been made for non-holonomic systems. However systems with complex dynamics, significant drift, underactuation and discrete system changes remain challenging for existing planning techniques particularly as the dimensionality of the state space increases. This thesis develops a novel motion planning technique for the solution of problems with these challenging characteristics. The novel approach is called Path Directed Subdivision Tree Exploration algorithm (PDST-EXPLORE) and is based on sampling-based motion planning and subdivision methods. PDST-EXPLORE demonstrates how to link a planner with a physical simulator using the latter as a black box, to generate realistic solution paths for complex systems. The thesis contains experimental results with examples with simplified physics including a second order differential drive robot and a game which exemplifies characteristics of dynamical systems which are difficult for planning. The thesis also contains experimental results for systems with simulated physics, namely a weight lifting robot and a car. Both systems have a degree of physical realism which could not be incorporated into planning before. The new planner is finally shown to be probabilistically complete.

Acknowledgments

I would like to thank my advisor Lydia Kavraki who is an outstanding example of what a great mentor and a friend can be.

My deepest gratitude and love go to my wonderful wife Fumiko, my family and my friends.

My thesis committee provided constructive critique and feedback that improved this thesis significantly and helped me prepare my defense. I am grateful for this help.

The members of my group at Rice inspired me with the great things they do; they listened to and discussed my ideas. This thesis would not have been possible without them.

Special thanks go to Brian Chen for six years of mutual reassurance and friendship.

Over the years I have had many discussions with researchers in the robotics community. Thanks for the many discussions, feedback, inspiring ideas, greatness and help.

Finally, I would like to thank Dr. Gordon, Dr. Einhorn and Dr. Kim and their staff for saving my life.

Work on this thesis has been partially supported by NSF 9702288, NSF 0308237, NSF 0205671, and an FCAR fellowship to Andrew Ladd. The computational experiments were carried on equipment obtained by the above grants and by NSF EIA 0216467, CNS 0454333 and CNS 0421109 in partnership with Rice University, AMD and Cray.

Contents

Illustrations

Tables

Chapter 1

Introduction

This thesis proposes a new framework for the "motion planning" problem in robotics. The introduction explores the various interpretations of "motion planning" over the years. An attempt will be made to provide the historical and academic perspective by considering a selection of quotations from foundational texts in the fields of robotics and control theory.

1.1 Motion Planning in the Last Fifteen Years

The seminal book of Latombe "Robot Motion Planning" published in 1991 [Lat91] states in its introduction:

> "In this book we are interested in giving the robot the capability of planning its own motions, i.e., deciding automatically what motions to execute in order to achieve a task specified by initial and goal spatial arrangements of physical objects. Creating autonomous robots is a major undertaking in Robotics. It definitively requires that the ability to plan motions automatically be developed. [...] This would allow the user to specify tasks more declaratively, by stating what he/she wants done rather than how to do it. As robots become more dexterous, the need for motion planning tools will become more critical."

The final sentence of the above quote predicts the current state of affairs in robotics. In the fifteen years since it was written an enormous amount of progress has been made in robotics in terms of landmark examples of robot autonomy, such as the RHINO tour guide

robot [BBC+95], the Honda ASIMO biped [Hon], the Sony Robocup Legged League [Rob] and the DARPA Grand Challenge [TMD+06]. These examples are huge improvements in precision and sophistication of robot hardware and require the development of new and effective motion planning algorithms. The progress in autonomous robotics has been largely driven by advances in real-time sub-systems and integration: namely, in computer vision, map and model building, localization and motion generated by feedback control. Motion planning in these examples is typically limited to 2-D planning for obstacle avoidance or is simply not necessary. Progress in the development of accurate, capable and dexterous robot hardware has proceeded very quickly. Huge improvements in servo motors, batteries, sensors, embedded computers and other "everyday" devices have led to robots that have enormous potential for autonomy and complicated environmental interaction. These possibilities are currently not realized since the need for automatically generated motion for dexterous systems prevents using new robot hardware to reach new landmarks in autonomy. In an introductory quote from Principles of Robot Motion [CBH+05] written in 2005, fifteen years after Latombe's book, little has changed.

> "Some of the most significant challenges confronting autonomous robotics lie the area of automatic motion planning. The goal is to be able specify a task in a high-level language and have the robot automatically compile this specification into a set of of low-level motion primitives, or feedback controllers, to accomplish the task."

A key difference in language in the above definition for motion planning compared to Latombe's words from fifteen years ago is the explicit mention of the intuitive notion of "low-level motion primitives" or "feedback controllers". Low-level motion primitives such as geodesics or 0 acceleration curves play an important role in the construction of popular

and effective sampling-based motion planning algorithms, a class of motion planning algorithms that is currently widely used [Ov94, Kav95, Ov95, KvLO96, Šve97, KL98, KL00, LK01c, HLM99, HKLR00b, HKLR00a]. The mention of feedback control belies the view that the role of a motion planner in a full robot system is to provide open-loop trajectories that can be fed into a feedback controller, or sequence of feedback controllers, to physically realize the plan in hardware. Quoting again from Latombe's book [Lat91]

> "A still widespread view is that motion planning is essentially some sort of simple collision checking. Motion planning is much more than that. As we will see in this chapter, it presents an unexpected variety of facets, the simplest of which raise provably difficult computational issues. In fact, the kind of operative intelligence that people use unconsciously to interact with their environment, which is needed for perception and motion planning, turns out to be extremely difficult to duplicate in a computer program."

It is a well-known fact in robotics that finding collision-free paths in a world consisting only of geometric and kinematic constraints can be a very hard problem, both in practice and theory. Additional constraints such as kinodynamics, multiple agents, parallel chains and contact increase complexity sharply [Lat91, CBH$^+$05]. However, many automatic motion generation problems deriving from more ambitious and capable dexterous robots do not reduce to the simple problem of finding collision-free paths. More precisely, reconstructing a feasible trajectory that satisfies additional physical constraints from a collision-free path often goes beyond what can be done with traditional control tools. The next quotation comes from classic text on traditional control theory by Stengel originally published in 1986 [Ste94].

> "Designing control logic that commands a dynamic system to a desired output

> or that augments the system's stability is a common objective of many technical fields, ranging from decision making for economic and social systems through trajectory control of robots and vehicles to "knowledge-based engineering" in the application of "artificial intelligence" concepts. If the control objective can be expressed as a quantitative criterion, then optimization of this criterion establishes a feasible design structure for the control logic."

Traditional control theory shares a deep connection with the methods of mathematical optimization. Concepts such trajectory following and stability can be formulated in terms of minimizing an objective in the presence of noise in a particular space. There are many problems that cannot be solved using these tools and control theorists have looked to predictive methods and open-loop trajectory generation to solve hard control problems that arise in real applications. Quoting from Choset et al [CBH$^+$05]

> "Motion planning for nonholonomic and underactuated systems has been the subject of a great deal of recent research, and the results could easily fill several books [...]. Motion planning approaches with roots in control theory tend to apply to systems with particular structure and no obstacles, while approaches based on search algorithms are computationally intensive and are suited to finding collision-free trajectories among obstacles."

An accurate, but broad, definition of motion planning is the computational construction of inputs to a physical system designed to produce a desired observable output. This definition is virtually identical to the objectives of control theory, however certain differences exist. Motion planning methods are typically "offline" methods that produce solutions for realizing a task. This is sometimes referred to as trajectory generation. Control theoretic methods deal primarily with the "online" problem, i.e., how to generate inputs in response

to observed outputs. Motion planning tends to use algorithmic analysis techniques and is model driven. Control theory leans more towards tools from optimization and convergence analysis and is not necessarily model driven. Most specifically, motion planning and control theory can be distinguished by the class of constraints they typically consider. Motion planning algorithms focus primarily on global constraints and controllers tend to deal with local constraints. In fact, the two methodologies are generally weak at duplicating the strengths of the other.

A principle goal of this work is to develop algorithmic techniques that exploit the strengths of traditional motion planning algorithms and make use of the strengths of control theory to solve increasingly physically realistic problems. The contributions in this work advance the state-of-the-art in this area by a significant step. However, the final quotation in this section speaks to a further goal which is to find algorithms that can exploit the power of methods from motion planning, control and artificial intelligence to realize the goals articulated fifteen years ago: to go from a high-level description of a task to realized and robust motion on a robot. Quoting from LaValle's 2006 book "Planning Algorithms" [LaV06]

> "Due to many exciting developments in the fields of robotics, artificial intelligence, and control theory, three topics that were once quite distinct are presently on a collision course. In robotics, motion planning was originally concerned with problems such as how to move a piano from one room to another in a house without hitting anything. The field has grown, however, to include complications such as uncertainties, multiple bodies, and dynamics. In artificial intelligence, planning originally meant a search for a sequence of logical operators or actions that transform an initial world state into a desired goal state. Presently planning extends beyond this to include many decision-theoretic ideas such as Markov decision processes, imperfect state informa-

> tion, and game-theoretic equilibria. Although control theory has traditionally been concerned with issues such as stability, feedback, and optimality, there has been growing interest in designing algorithms that find feasible open-loop trajectories for nonlinear systems. In some of this work, the term "motion planning" has been applied, with a different interpretation from its use in robotics. Thus, even though each originally considered different problems, the fields of robotics, artificial intelligence, and control theory have expanded their scope to share an interesting common ground."

This thesis extends the current state-of-the-art in motion planning by coupling algorithmic techniques with control theory and physical simulation in an effort to generate planners that are directly applicable to complex physical systems.

1.2 Physical Simulation

Research in physical simulation deals with an entirely different set of objectives. In the realm of simulation, computation is used to predict the output of an inputless physical system. The design objectives for physical simulators are to reduce computational cost and to increase the accuracy of the reproduction. Frequently physical simulators are subject to rigorous validation against actual measured results or other, known accurate simulators. Design trade-offs balance expressiveness, computational cost, scalability, accuracy and the complexity of parameter estimation. Significant progress is often driven by the discovery of clever approximations, often domain specific, and implementation improvements. Research in physical simulation developed in concert with the modern computer, has empowered an enormous amount of progress in every aspect of science and engineering, and is still a powerful motivation for efforts in computer science. In the last decade, new phys-

ical approximations, fast geometric algorithms, faster optimization techniques, massive improvements in low-cost floating point processing and cheap, fast memory have led to a new class of simulators that accurately model and simulate rigid body physics with contact and friction at order of magnitude faster than realtime on low cost consumer hardware. Additionally, driven by the gaming industry, special-purpose physics processing hardware to further extend the capabilities of these simulators is beginning to enter the consumer market.

1.3 The Model Gap

In the context of practical system engineering for many applied robotics implementations, both motion planning and control theory are used. It is typical, though, for the model of the robot employed by the motion planner to be significantly simpler than the one used by the controller. Typically, the controller uses, either implicitly or explicitly, a richer and more expressive robot model that is a closer approximation to reality. Often model error is estimated by having to estimate model parameters or by unobservable noise inputs. Additionally, since controllers are often local methods, models that only are valid locally such as a linearizations of non-linear differential systems around a given point can be used. Motion planners often use grossly simplified robot models to side-step the thorny difficulties posed by increased complexity. The resulting model gap between the planner and controller can sometimes be modeled as system error and fixed on the fly by the controller. However, planners rarely deal directly with physical dynamics, and consequently the resulting model gap can be fatal. As robotics applications become increasingly ambitious and move away from the domain of statics and geometry towards physical dynamics and environmental interaction, much of the current motion planning research is almost totally inadequate. For this reason, research at the forefront of hardware robotics rarely makes use of motion plan-

ning, foregoes the advantages of algorithmic design, automation and provable guarantees and instead often employs labor intensive, ad hoc solutions to trajectory design.

1.4 Opportunities in Robotics

The impact of recent developments in physical simulation on robotics has already begun. The ease of use, accuracy and speed of existing simulation software packages has made physical simulation an invaluable tool in mechanism and robot design, and provides a way to test controllers without the risk of damaging expensive hardware. Perhaps most important, a relatively small of amount of effort can be expended learning or determining the physical parameters for the simulator and then the simulator can be leveraged to nearly totally automate parameter tuning for the controller, which is often very labor intensive because of the time overhead in setting up experiments on a physical platform. Simulation also provides an effective abstraction barrier to divide labor between hardware designers and programmers. Superficially, the evolution of motion planners reasoning about coarse models with little or no physics towards direct reasoning on physical simulations seems obvious. The advantages are motivating: the problem of model gap between the planner and the controller can be reduced to the problem of validating the predictiveness of the simulation. Since controller design in a simulator and subsequent validation on a physical platform is a tested and successful research and development paradigm, it is reasonable to think that trajectory design in simulator will also be very effective. An additional strength is the capability of the planner to directly incorporate the controller by simulating the operation of the controller. Since controllers often provide excellent methods to reduce the number of input parameters and increase the usefulness of generated motions in complex systems, this strength may provide enormous computational efficiency advantages as well being an effective marriage of the advantages of control theory and motion planning.

1.5 Contributions

As stated earlier, a principle goal of this work is to develop an algorithmic framework that exploits the strengths of traditional motion planning algorithms and makes use of the strengths of control theory to solve increasingly physically realistic problems.

This goal is achieved through

- The design of a new algorithm for motion planning, and
- The tight coupling of the above algorithm with physical simulators, which are used as black boxes by the planner.

As it will be shown in this thesis, the proposed approach can handle problems of complexity that has not been dealt with in the past, mainly because of the accurate modeling and physical realism of the robots involved. Although similar problems had been solved by hand, it is the first time that it is demonstrated how they can be attacked in a fully automated way.

An important point that will be iterated multiple times in this thesis is that physical simulation is used by the proposed planner as a black box. This is reminiscent of the way collision checking has been used by sampling-based motion planners in the past [CBH$^+$05] (see discussion in Chapter 2). Sampling-based planners benefited enormously by advances in collision checking which were driven by academic disciplines other than robotics (e.g., computational geometry) and industries other than robotics (e.g., games and entertainment). The faster collision checking became, the faster sampling-based planners became. The proposed planner will benefit from advances in physical simulation, which are again driven by powerful industries, in the same way that sampling-based motion planners benefited from advances in collision checking.

1.6 Organization

Chapter 2 gives some background on sampling-based motion planners which provided an inspiration to the proposed planner. It also includes a short discussion on physical simulation. Chapter 3 defines in detail the motion planning problem and illustrates the principles and design decisions that will govern the design of the new planner. Chapter 4 presents the new planner. Experiments with simplified physics are given in Chapter 5, while Chapter 6 shows experiments with simulated physics. The probabilistic completeness of the proposed planner is shown in Chapter 7. This thesis ends with a discussion in Chapter 8.

Chapter 2

Background

There have been many alternative approaches that address aspects of the motion planning problem since its introduction almost thirty years ago. The following discussion provides an overview of important milestones in the field and focuses on advances that have inspired the proposed planner.

2.1 The General Motion Planning Problem

Motion planning asks for valid actions that move a system from an initial to a goal spatial arrangement [Lat91]. A specification of a robot's spatial arrangement is called a *configuration* and all configurations define the configuration space. The general mover's problem is posed for polyhedral robots in a polyhedral workspace and is `PSPACE-hard` [Rei79]. In two dimensions this problem is also known as the Sofa mover's problem, while in three dimensions it is known as the Piano mover's problem. Early on, a single exponential algorithm was proposed that uses a cylindrical algebraic decomposition of the semi-algebraic descriptions of the configuration space to solve it [MBOR86]. Following this line of research, a series of theoretical results provided polynomial time algorithms for problems with fixed dimensions. For example an algorithm for the Sofa mover's problem has complexity $O(n^5)$ [SS83a], for moving a fixed number of discs the complexity is $O(n^3)$ [SS83b], for moving a 2D star-shaped robot with k arms [SAS84] it is $O(n^{k+4})$ and for moving a 3D rod shaped robot [SS84] the complexity is $O(n^{15})$. These

results, however, also suggested an exponential dependence on the dimensionality of the problem. Eventually a PSPACE algorithm [Can88] was constructed that proved that the problem is PSPACE-complete. The same algorithm introduced the roadmap, a 1D subspace of the configuration space that captures its connectivity. The algorithm, however, is impractical both computationally and implementation wise. Other approaches attempted to approximate the structure of the configuration space but they were also impractical [BP83, Per83, KD86].

A different paradigm based on potential fields and obstacle avoidance [Kha86] was also developed [HA88,Kod89,BL91,HA92]. Nevertheless, the construction of general complete potential field approaches methods was proved difficult [Kod89, RK92]. A potential-field based method that solved difficult problems was the RPP planner [BL91, LL96]. It took a stochastic approach to avoid local minima and later was proved complete [LL96]. Sequential search with backtracking was also explored [GG95].

2.2 Probabilistic Roadmap Planners

While the deterministic and exact planners faced difficulties in addressing many interesting instances of the general motion planning problem, advances in modeling provided accurate CAD models of spaces and robots that required efficient planners for solving planning queries. This development led to the introduction of a new series of algorithms that took advantage of the advances in collision detection and modeling [LC91,Qui94,LM91,MC95, LM97,GLM96,KPLM98,EL00] and used sampling to improve performance. Examples of these planners are the Ariadne's Clew algorithm [ATBM92, BATM94, AGM98] and the Probabilistic Roadmap Method (PRM) [KL94,Ov94,Kav95,Ov95,KvLO96,Šve97,KL98]. Especially the latter, PRM, proved to be very successful in problems with complex geometries by utilizing a learning phase. It is also easier to implement than its algebraic

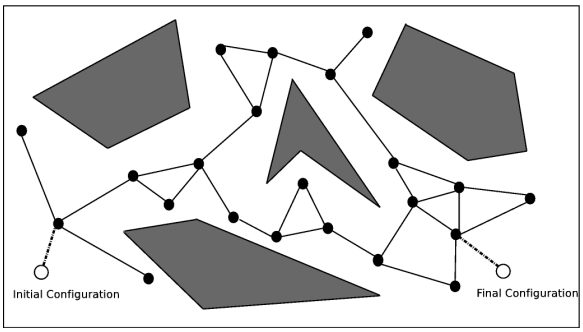

Figure 2.1 : A simple PRM roadmap. The gray areas are obstacles. The dots are sampled configurations. Edges denote collision-free paths

counterpart.

PRM splits planning in two phases, a learning and a querying phase. During learning, PRM samples collision-free configurations and connects them with simple paths to build a roadmap. The initial implementation used uniform sampling and straight line paths [KvLO96]. Given a roadmap, multiple queries can be answered quickly. First the initial and goal configurations are connected to the roadmap and then planning amounts to solving the corresponding graph search problem. An illustration of a PRM roadmap is given in Figure 2.1.

The efficiency of PRM depends on how well the sampling strategy can capture the connectivity of the free space. A major factor that affects PRM performance is the presence of "narrow passages" [ABD+98, LK02, HKL+98] in the space. In order to solve problems with "narrow passages", PRM must select samples from very small sets in order to connect the roadmap. A plethora of biased sampling techniques were introduced to improve the effectiveness of PRM. Example include quasi-random sampling [LB02], or a variety of techniques that focused sampling on subsets of the configuration space. These subsets could be either parts of the space with low connectivity [KvLO96], or parts of the space close to the obstacle boundaries [ABD+98, AW96, BOvdS99] or the medial-axis of the free

space [OY85, GHK99, WAS99, HK00].

The benefits of sampling-based planning with PRM come at the cost of sacrificing completeness. That is, path non-existence cannot be proven with sampling-based planners. A looser guarantee, termed *probabilistic completeness* is provided instead. If an algorithm is probabilistic complete and a path exists, then the planner will find it eventually [KvLO96, KLMR96, KKL96, Šve97, LK02]. This was proved for the PRM algorithm originally for k-dimensional manifolds [KLMR96, KKL96, KvLO96, Šve97], then for non-holonomic robots [Šve97] and recently for a broad class of problems [LK02].

A main emphasis of the PRM planner was the use of collision checking both for selecting the samples and for creating connections among the samples. The planner benefited enormously from advances in collision checking [LC91, Qui94, LM91, MC95, LM97, GLM96, KPLM98, EL00]. In retrospect, using collision checking as a black box was a powerful primitive of PRM that contributed to its performance, ease of implementation and subsequent widespread use.

2.3 Single-Query Sampling-based Planners

PRM takes advantage of the roadmap constructed during the learning phase to efficiently answer multiple queries. In many cases however only single query problems have to be solved. The solution to a single query problem can be often obtained more efficiently by focusing the exploration only on certain parts of the configuration space, without the need to construct a roadmap that captures the connectivity of the entire configuration space. This observation motivated the development of a series of approaches that focus on answering single queries [BK00, SL01, Boh01, LB02, LL06, LL05, HKLR02, SK06, SK05, YJSL05, HBHL06, BB05].

One approach to single-query motion planning is to speed up computation by doing

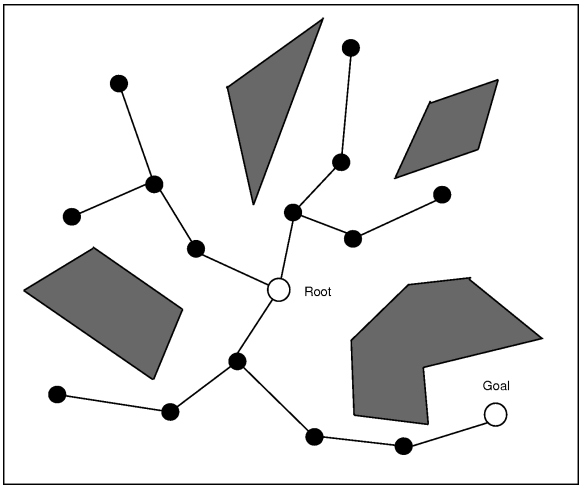

Figure 2.2 : A generic tree

collision checking only when it is necessary. The lazy PRM variant [BK00, SL01, Boh01] constructs a roadmap whose nodes and edges are initially assumed to be collision-free. The roadmap can even be computed implicitly on a grid [Boh01, LB02]. As in the PRM formulation, lazy PRM searches the roadmap graph to find a path that connects the query configurations. The computed path is then checked for collisions. The path is discarded if it is found in collision and the violating nodes and edges are removed from the roadmap. Lazy PRM continues to search the roadmap for alternative paths in this manner until a collision-free path is found.

An alternative approach to roadmap-based planners yielded in a family of planners known as tree-based planners [BK00, SL01, Boh01, LB02, LL06, HKLR02, SK06, SK05, YJSL05, HBHL06, BB05]. In contrast to roadmap-based approaches, the objective of tree-based planners is to bias the exploration toward those parts of the configuration space that facilitate the construction of a path that connects the given initial and goal configurations. The idea here is to start the exploration by rooting a tree at the initial configuration and incrementally explore the relevant parts of the configuration space by advancing the tree

toward the goal configuration. Figure 2.2 provides an illustration. The tree-based planners have been successfully used to solve not only geometric motion planning problems, but also motion planning problems with kinematic and dynamic constraints, as it will be explained later in this chapter.

The success of tree-based planners depends on the strategy employed to expand the exploration tree. Typically the tree is expanded by adding new configurations and edges. One popular and successful approach is the Rapidly-Exploring Random Tree (`RRT`) [LK99]. At each iteration a configuration, q_{rand}, is sampled at random from some probability distribution. The nearest configuration in the tree, q_{near}, is then computed and attempts are made to extend q_{near} toward q_{new}. A new configuration, q_{new}, is obtained by moving q_{near} for a small distance along the straight-line defined by q_{near} and q_{rand} in the configuration space. This process is repeated until q_{near} reaches q_{rand} or the extension is no longer possible due to collision with obstacles. Several or all the collision-free intermediate configurations q_{new} are added to the tree. The tree is pulled toward the goal by often generating random configurations near the goal configuration.

The Expansive Spaces Tree (`EST`) approach is another successful tree-based planner [HLM99,HKLR00b,HKLR00a]. `EST` attempts to advance the tree in unexplored regions of the configuration space. At each iteration, an existing node in the tree is selected according to some probability distribution and extended for a short distance in a random direction. Nodes in the tree that are in dense areas are selected less frequently than nodes in sparse areas. The tree thus explores new regions of the configuration space that could lead to the construction of a path from the initial to the goal configuration.

Many other tree-based planners have been proposed. The Single-Query Bidirectional Lazy `SBL` planner combines the idea of collision checking with tree expansion [SL01]. Some instances of tree-based planners build two trees, one rooted at the initial configu-

ration and another one rooted at the goal configuration, and grow the trees toward each-other [HLM99, LK00, SL01]. The Sampling-based Roadmap of Trees (SRT) leverages the benefits of PRM and tree-based planners. It tries to expand multiple trees from various initial seed points in the configuration space. Neighboring trees are then grown toward each-other giving rise to a roadmap of trees that ultimately captures the connectivity of the configuration space. SRT has proven effective both for multiple-query and single-query planning while it can also be parallelized very efficiently [BCL+03, ABC+05, PBC+05, PK05].

The planner proposed in this thesis is a tree-based planner. However it displays several key differences with the planners above as discussed in the next chapter.

2.4 Dynamic Constraints in Motion

As mentioned in the introduction, there is an important model gap between control theory and motion planning. The role of a controller is to approximate reality as close as possible and take into account the real constraints and capabilities of a system. A controller is also responsible to adjust the behavior of the system in case of errors or unexpected events. Nevertheless, controllers can rarely address global planning problems. On the other hand, motion planners, which are built to solve global queries, often apply simplistic approximations to avoid modeling the full system's complexity. Often this gap in the representation of a solution can be fatal because the planner does not take into account the dynamic constraints. For example, a kinematic path, returned by a a planner such as PRM may not be executable due to bounds on velocity, acceleration and applied forces [DLOS98]. To alleviate this problem motion planners must incorporate a higher level of physical realism. The typical approach to address this problem is to augment the configuration space by including the parameters related to the motion constraints and define the system's state space [CBH+05]. The state of a system does not only characterize its spatial arrangement

but is a full description of its status in a way that if a control input and the state of a system is provided, then the progress of a system in the future can be simulated.

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

A hierarchical approach exists in techniques that use PRM. The planner can be used to produce a roadmap of kinematic paths, which are later adapted to the dynamic constraints of the robot [SvLO98, SA01]. Such techniques must compute a valid trajectory to connect two given states, a difficult sub-problem in general, known as the steering problem [LFV04] and hence the approach is not generally applicable. On the other hand, tree-based planners apply forward integration of controls instead of steering. This is a simple and fast primitive, which naturally simulates the underlying propagation model of a system. In this way, tree-based planners have become the norm in kinodynamic planning [CBH$^+$05, LaV06]. Examples include but are not limited to [HKLR00a, LK99, LK01c, LK01b, CSL01, CL03, CL02, CFL03, KVdP06, HBHL06].

2.5 Physical Simulation

Simulation studies "the design of a model of an actual or theoretical physical system, the execution of the model on a digital computer, and the analysis of the execution output" [Fis94]. This abstract definition emphasizes the applicability of simulation to a wide variety of applications, varying from factory simulation, computer networks, flight dynamics to operation research and imaginary worlds [Hol04, PC06]. In each of these areas, the use of simulation has resulted in procedures that are more cost effective, less dangerous, faster, or otherwise more practical than experimenting with a real system. The idea of using a model to formulate sequences of actions is also central to planning and, given a sequence of actions, a robot can use the model to simulate the future as it would occur if the actions were carried out [DW91].

Often, assumptions are made about this system and mathematical algorithms and relationships are derived to describe these assumptions. This constitutes a "model" that can reveal how the system works. If the system is simple, the model may be represented and solved analytically. However, many problems of interest in the real world are usually much more complex and a simple mathematical model cannot be constructed to represent them. For example, in some problems in robotics, a state transition equation might not be available. Nevertheless, it is still possible to compute future states given a current state and an action trajectory by using simulation as a black box [LaV06], since simulators can also work internally with implicit differential constraints. In computer graphics applications, simulations can even arise from motion capture data [RPE+05].

The focus in this thesis is on physical simulators of multi-body dynamics [Smi06, Hav, MSO94], especially for rigid and potentially articulated bodies, so as to plan for the motions of such bodies. The field of rigid body dynamics and more generally of multi-body dynamics designs mathematical models and algorithms to predict the motions of bodies and

the contact forces, including friction, that arise between them when they interact [ST96]. Multi-body contact systems are involved in many engineering applications such as robotic manipulation, fixture loading, graphic simulation and haptic display that can be used for planning. Rigid bodies are a convenient approximation to reality for these applications. Although it is inevitable that the bodies will deform when a contact occurs, the deformation can be neglected, considering the purpose of these applications. For computational purposes, the underlying theory for a rigid-body system is much simpler than for deformable bodies, and computation of the rigid-body system is more reliable and efficient [Ste00]. Simulation of deformable objects is also receiving a lot of attention recently [BSB+01] and motion planners for deformable objects [LK01a, MK06] should benefit from the increasing realism of such physical simulators.

2.5.1 Rigid Body Dynamics and `ODE`

A realistic simulation must both respect the laws of Newtonian dynamics and not allow two rigid bodies to inter-penetrate [Bar89]. The simulator must calculate what forces would actually arise in nature to prevent bodies from inter-penetrating and then use these forces to derive the action motion of the bodies. Lotstedr [Lot84] represents the first attempt to compute friction forces in an analytical setting, by using quadratic programming to compute friction forces based on a simplification of the Coulomb friction mode. Baraff also proposed analytical methods for dealing with friction forces and presents algorithms that deal both with static friction [Bar93] and dynamic or sliding friction [Bar94].

The Open Dynamics Engine (`ODE`) [Smi06] is a popular, successful example of a physics-based simulator that has been used in this work. It is an open source, high performance library for simulating rigid body dynamics. It is a fully featured, stable software platform that includes modeling of advanced joint types and integrated collision detec-

tion with friction. ODE is useful for simulating vehicles, articulated objects and objects in virtual reality environments. There are many other physical simulator available both proprietary [Hav] and open-source ones [MSO94].

ODE implements a time-stepping algorithm similar to the approaches by Stewart and Trinkle [ST96] and Anitescu et al. [APS99], but with reduced accuracy in order to increase frame rates for applications. These methods contrast with the more traditional approaches, which involve formulating a system of equations, or complementarity problem, at each time-step to be solved for the forces, which are then used as input for a differential equations or differential-algebraic equations solver [HWY86, Lot82, PG96]. The problem with traditional approaches is that the systems of equations or complementarity problems to be solved at each instant in time may not have a solution. The point of the new time-stepping techniques is that they avoid this problem by implicitly allowing impulsive forces at any time during contact, not just at the instant of impact. The solver used by ODE is based on Baraff's pivoting method [Bar92], while collisions are handled following Mirtich's method [Mir97].

Chapter 3

Problem Statement

The focus of this thesis is to describe how motion planning for systems with a high degree of physical realism can be solved in practice. The specific target is to consider models that can express rigid body dynamics in 3-D under a rich variety of constraints such as force limits, serial and parallel kinematics, motor dynamics, friction and contact. To do this correctly, it is necessary to consider the possibility that discrete changes in the set of constraints can occur. Additionally, it is convenient to assume very little about specific simulation techniques. Finally, extensions to more advanced physical systems such as deformable materials or fluid dynamics should be theoretically straightforward. Figure 3.1.

3.1 Preliminaries and Definitions

In order to introduce the algorithm developed in this thesis a simple 2-D robot motion planning problem will be used. This problem is depicted in Figure 3.1. This example will be used as an analogy to simplify exposition but will be rigorously generalized in the formal treatment.

3.1.1 States and State Space

A state consists of the information required to represent the entire physical system at a given instant of time. The intention of the abstraction is that a state consists of information that is, together with the relevant external input, sufficient to run the simulation without history.

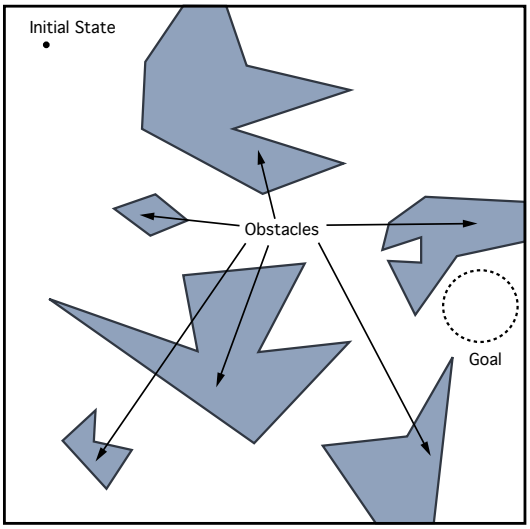

Figure 3.1 : A motion planning problem

The set of all states is called the state space and is written $\mathcal{Q}$. The notion of state only requires that the representation be sufficient to run the simulation without history and does not demand any kind of minimalism. Although it is desirable to avoid considering states that are unreachable in the simulation or even totally unreasonable, it can be very difficult to characterize these properties and consequently to design minimalistic state spaces.

3.1.2 Obstacle Constraints

In the problem shown in Figure 3.1, the state space is $\mathcal{Q} = [0, 1]^2$, the unit square. A state consists of the pair $(x, y) \in [0, 1]^2$ and is the point robot's position in the square. Intuitively from the diagram, it is clear that some of the states are not reasonable, insofar that robot is inside the obstacles. The obstacles shown in Figure 3.1 form a polygonal subset $\mathcal{O} \subset \mathcal{Q} = [0, 1]^2$. A region of state space that is an obstacle or is forbidden is one important kind of constraint that is often used in motion planning.

3.1.3 Paths

A path in the state space is a function $\pi : [0, T] \rightarrow \mathcal{Q}$, where $T \geq 0$. The quantity T is the duration of the path and meant to represent the amount of time the path takes. The intention of the path abstraction is that the path be feasible motion of the robot in the state space. In the example in Figure 3.1, a feasible path is any continuous map $\pi : [0, T] \rightarrow [0, 1]^2$ such that $\pi(t) \notin \mathcal{O}$. In general, the notion of path in this treatment need not be continuous.

3.1.4 Initial State and Goal Region

An initial state and a goal region are shown in the diagram in Figure 3.1. These are used to define the motion planning problem. In the case of the example in Figure 3.1, the motion planning problem is to construct a continuous path $\pi : [0, T] \rightarrow \mathcal{Q}$ such that the path begins at $\pi(0) = q_{\text{initial}}$, the initial state, and ends in the goal region, $\pi(T) \in G \subset \mathcal{Q}$.

3.1.5 Controls and Control Space

In 2-D point robot example from Figure 3.1, there are many ways of representing a solution path. However, in a general setting, paths are generated and represented by a sequence of inputs or controls. A control is an input to the represented system and the set of all controls is written $\mathcal{U}$. The critical detail is the way the control inputs are interpreted and determines how the modeled system behaves.

To begin this description, consider the augmented state space $\overline{\mathcal{Q}} = \mathcal{Q} \cup \{\perp\}$. The additional state, $\perp$, is used to represent constraint violation or universal failure condition. The system behavior is governed by a function called the transit function

$$F : \overline{\mathcal{Q}} \times \mathcal{U} \times \mathbb{R}^{\geq 0} \rightarrow \overline{\mathcal{Q}}.$$

The interpretation of the transit function is as follows: $F(q, u, t)$ is the state that results

from applying control input u from state q for time t.

The control space for the example in Figure 3.1 is $\mathcal{U} = [-v_{\text{max}}, v_{\text{max}}]^2$ where $v_{\text{max}} > 0$. Each $u = (v_x, v_y) \in \mathcal{U}$ is interpreted as a velocity for the point robot. The transit function for the example in Figure 3.1 is given precisely by the following expression:

$$F(q, u, t) = \begin{cases} \perp & \text{if there exists } t' \in [0, t] \text{ such that } q + ut' \in \mathcal{O} \\ q + ut & \text{otherwise.} \end{cases}$$

3.1.6 Solutions

In the example from Figure 3.1, the state space is $\mathcal{Q} = [0, 1]^2$. The constraint on the motion of the robot is that there is a part of the state space that is forbidden. This part is polygonal subset $\mathcal{O} \subset \mathcal{Q}$. The motion planning problem in this domain is specified by the initial state q_{initial} and the goal region G. A solution to this problem consists of a feasible path $\pi : [0, T] \to \mathcal{Q}$ such that $\pi(0) = q_{\text{initial}}$ and $\pi(T) \in G$, i.e. the path π begins at the initial state, ends in the goal region and does not intersect the obstacles.

The generalized model used in this work represents motion in terms of controls and times. In the point robot example, the controls are the velocities of the robot. In this way, a path is given by a state q, an integer $n > 0$, a sequence of controls $u_1, ..., u_n$ and a sequence of times $t_1, ..., t_n$, where $t_i > 0$. The path defined by q, $u_1, ..., u_n$ and $t_1, ..., t_n$ is written $\texttt{path}(q, u_1, ..., u_n, t_1, ..., t_n)$. Recursively defining $T_0 = 0$, $T_i = T_{i-1} + t_i$, the path $\pi = \texttt{path}(q, u_1, ..., u_n, t_1, ..., u_n)$ is given explicitly as

$$\pi(t) = \begin{cases} q & \text{when } t = 0 \\ F(\pi(T_i), u_i, t - T_i) & \text{when } t \in [T_i, T_{i+1}] \end{cases}.$$

A solution path represented in this way is given an integer $n > 0$, controls $u_1, ..., u_n$ and times $t_1, ..., t_n$ and is the path $\pi = \texttt{path}(q_{\text{initial}}, u_1, ..., u_n, t_1, ..., t_n)$. To be a solution,

the end state of π should be in the goal region, $\pi(|\pi|) \in G$. A motion planning algorithm would then need to search for such a path.

3.2 General Motion Planning Problem

Section 3.1 described a point robot motion planning problem in two dimensions. This section generalizes the treatment and provides formal definitions for the concepts of Section 3.1. The section begins by defining a motion planning problem instance.

Definition 3.2.1 (Motion Planning Problem Instance (`MPP`)). *An instance of motion planning problem is a tuple*

$$(\mathcal{Q}, \mathcal{U}, F, q_{initial}, G)$$

with

- $\mathcal{Q}$ *is a probability space called the state space.*
- $\mathcal{U}$ *is a probability space called the control space.*
- F *is a measurable function*

  $$F : \overline{\mathcal{Q}} \times \mathcal{U} \times \mathbb{R}^{\geq 0} \to \overline{\mathcal{Q}}$$

  where $\overline{\mathcal{Q}}$ *is the augmented state space* $\overline{\mathcal{Q}} = \mathcal{Q} \cup \{\perp\}$ *where the special state* $\perp$ *has been added. The state* $\perp$ *is meant to represent a universal failure condition or constraint violation. The function* F *is called the transit function and must satisfy Property 3.2.2.*
- $q_{initial} \in \mathcal{Q}$ *is a state called the initial state.*
- $G \subset \mathcal{Q}$ *is a measurable subset of the state space called the goal set.*

Definition 3.2.1 restates the definition of the transit function used in Section 3.1 in more formal terms and refers to necessary conditions a transit function must satisfy in Property 3.2.2, which will now be stated. These properties are designed to express the intention of the transit function as a rule for defining the behavior of a broad class of physical systems.

Property 3.2.2 (Transit Function Properties). *A transit function $F : \overline{\mathcal{Q}} \times \mathcal{U} \times \mathbb{R}^{\geq 0} \to \overline{\mathcal{Q}}$ defined over state space $\mathcal{Q}$ and control space $\mathcal{U}$ should satisfy the following properties:*

1. *For every $q \in \overline{\mathcal{Q}}$, $u \in \mathcal{U}$*

$$F(q, u, 0) = q.$$

   This requires that applying control u from state q for 0 time leaves the system unchanged.

2. *For every $q \in \overline{\mathcal{Q}}$, $u \in \mathcal{U}$, t_1, t_2,*

$$F(q, u, t_1 + t_2) = F(F(q, u, t_1), u, t_2).$$

   This requires that F is transitive: applying control u for time $t_1 + t_2$ from state q is equivalent applying u for time t_1 and then for time t_2.

3. *For every $q \in \overline{\mathcal{Q}}$, $u \in \mathcal{U}$, $0 \leq t_1 < t_2$*

$$F(q, u, t_1) = \perp \implies F(q, u, t_2) = \perp.$$

   This requires that once the failure state $\perp$ has been reached the system is stuck. In other words, F must be defined so that $\perp$ is a sink.

The next definition, Definition 3.2.3, in this section formalizes a solution of a motion planning problem.

Definition 3.2.3 (Motion Planning Problem Solution). *A solution of a given instance of the motion planning problem*

$$(\mathcal{Q}, \mathcal{U}, F, q_{initial}, G)$$

consists of a sequence of

$$\textit{controls } u_1, ..., u_n \in \mathcal{U} \textit{ and times } t_1, ..., t_n$$

such that $q_0 = q_{initial}$, $q_n \in G$ *and for* $1 \leq i \leq n$,

$$q_i := F(q_{i-1}, u_i, t_i).$$

3.3 Solving the MMP with an Incremental Tree Building Algorithm

As described in the previous chapter incremental tree-based algorithms are a popular approach to solving motion planning problems. The algorithm developed in this thesis is also an incremental tree-based algorithm with some fundamental differences than earlier approaches.

3.3.1 The Overall Scheme

The illustration of searching for a solution path in Figure 3.2 shows a sequence of tree structures in state space constructed by incrementally adding branches. The definition given here is intuitive and used for exposition.

A tree consists of a set of related path segments which, in the language of this treatment, are referred to as path samples. Each path sample can be thought of as a triple (q, u, T) where q is a state, u is a control and $T \geq 0$ is a time. The path sample represents the constant control feasible path π given by $F(q, u, t)$ for $0 \leq t < T$.

Described simply, a tree is a set of samples which is incrementally constructed. The initial set of samples is $\mathbf{S}_0 = \{(q_{\text{initial}}, \cdot, 0)\}$. The state $q_{\text{initial}} \in \mathcal{Q}$ is the initial state of the

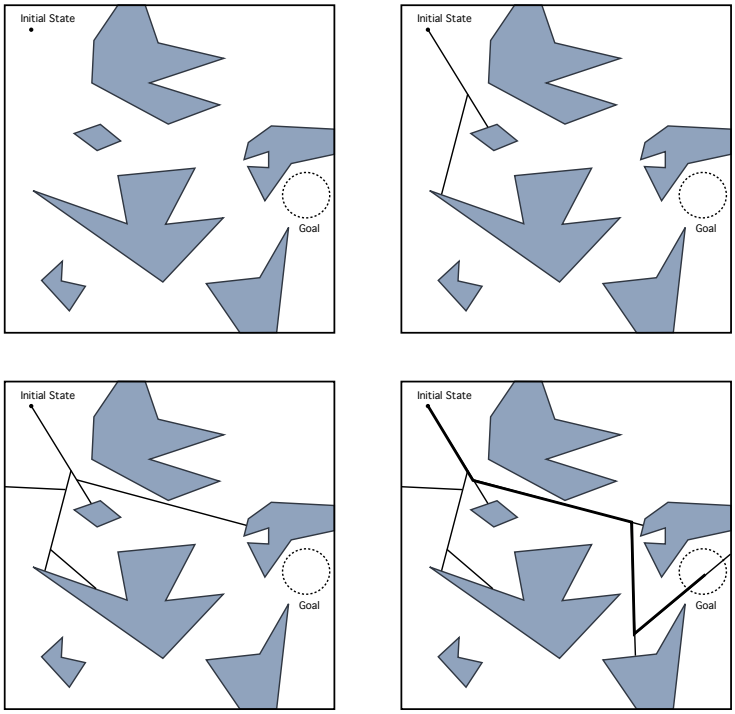

Figure 3.2 : Incremental tree construction for the 2-D point robot example

motion planning problem and the path sample is the zero duration path segment beginning at that state. The control for the sample is irrelevant and is denoted "·". Incrementally, the abstract scheme for adding a path sample operates as follows:

1. Select a path sample (q, u, T) from sample set $\mathbf{S}_i$.
2. Choose $t \in [0, T)$ and compute $q' = F(q, u, t)$.
3. Choose a control u'.
4. Find the smallest $T' > 0$ such that $F(q', u', T') = \perp$ or choose $T' = T_{\text{max}}$, some constant.
5. Add path sample (q', u', T') to obtain $\mathbf{S}_{i+1} = \mathbf{S}_i \cup (q', u', T')$.

The general termination condition for this process is to continue until some maximum number of iterations have been executed or some $\pi \in \mathbf{S}_i$ is generated such that $\pi \cap G$ is non empty. In the latter case, the discussion in the previous subsection describes a solution which can be easily extracted from the tree structure on the samples.

3.3.2 Design Decisions

The process in the previous subsection fits with the illustration in Figure 3.2. However, there are many decisions to make in designing a practical implementation of the described schema.

- How is the branch state on the tree chosen, i.e. which path sample and time t?
- How is the control u chosen?

This treatment proposes that these design decisions should be made to further the following objectives:

(i) The algorithm should be probabilistically complete *.

(ii) The algorithm should admit an efficient implementation.

(iii) The algorithm should be capable of exploiting the structure in motion planning problems.

(iv) The algorithm should be applicable to a large class of physical systems.

Objective (i) can be easily achieved for a very large class of physical systems by opting for a totally random selection strategy or for a systematically exhaustive strategy. Although

*A formal discussion of this concept appears in Chapter 7.

these solutions are very easy to implement, these approaches fail to exploit any of the structure present in physical systems and are unusably inefficient for anything other than trivial motion planning problems. Put differently, the challenge is to design a selection scheme for building trees that is probabilistically complete for the same class of systems as the random approach but has an experimentally effective mechanism for exploiting structure without being overly inefficient as an implementation.

3.3.3 Specifics and Definitions

The final part of this section develops a set of formal definitions for the notion of a path sample and a tree that was introduced in the above subsection. These definitions will be used in the description of the proposed planner and in the associated proof of probabilistic completeness.

Path Sample

This definition is with respect to a given `MPP`, $(\mathcal{Q}, \mathcal{U}, F, q_{\text{initial}}, G)$. A path sample is a tuple (q, u, D) where $q \in \mathcal{Q}$ is a state, $u \in \mathcal{U}$ is a control, and D is a non-empty finite union of real positive intervals, meaning for some $n > 0$,

$$D = \bigcup_{i=1}^{n} I_i \text{ where } I_i = (a, b), [a, b), (a, b], \text{ or } [a, b] \text{ for } a \leq b \in \mathbb{R}^{\geq 0},$$

such that for $t \in D$, $F(q, u, t) \neq \perp$.

Note that in contract with previous work, the proposed algorithm will use paths as samples and not individual states.

Primitive Path

If for some path sample $\pi = (q, u, D)$, $D = [0, T)$ for $T \geq 0$ [†], then π is primitive path sample.

The `integrate` Operator

The operator `integrate` is used to construct new primitive path samples. This operator takes a state and a control as arguments and returns a primitive path sample

$$\texttt{integrate}(q, u) = (q, u, [0, T))$$

where either T is minimum among all $T > 0$ such that $F(q, u, T) = \perp$ or $T = T_{\text{max}}$, some fixed positive constant, in the case $F(q, u, t) \neq \perp$ for $t > 0$.

Intersection of a Path Sample and a Subset of the State Space

Let $A \subset \mathcal{Q}$ and let be (q, u, D) be a path sample. The intersection operator is defined by

$$(q, u, D) \cap A = (q, u, D')$$

where

$$D' = \{t \in D : F(q, u, t) \in A\}.$$

The result of the intersection is well defined if (q, u, D') is also a path sample. If $D' = \emptyset$ then, by convention, $(q, u, D) \cap A = \emptyset$ can express this.

In the proposed algorithm a decomposition of the state space will be used. The above operation will be critical for the design of the algorithm.

[†] an interval of the form $[a, b)$ is interpreted as $[a]$ when $a = b$

Union of Two Path Samples

Let $\pi_1 = (q, u, D_1)$ and $\pi_2 = (q, u, D_2)$ such that $D_1 \cap D_2 = \emptyset$. The union operator for path samples is well defined in this case and yields a new path sample in the following way:

$$\pi_1 \cup \pi_2 = (q, u, D_1 \cup D_2).$$

Primitive Trees

A set of samples $\mathbf{S}$ is a primitive tree if the following conditions are met:

1. Every path sample $\pi \in \mathbf{S}$ is a primitive path sample.

2. The sample set $\mathbf{S}$ contains a distinguished sample $\pi \in \mathbf{S}$ such that $\pi = (q, \cdot, [0])$. This sample is called the root and is denoted $\texttt{root}(\mathbf{S})$.

3. Every path sample $\pi = (q, u, [0, T)) \in \mathbf{S}$, except for the root, has a parent $\pi' = (q', u', [0, T')) \in \mathbf{S}$ such that there exists $t \in [0, T')$ with $\pi'(t) = \pi(0)$. The operator `parent` expresses the parent relationship in that $\texttt{parent}(\mathbf{S}, \pi) = \pi'$ where π' is the parent of π. Additionally, the graph induced by the `parent` relation on the elements of $\mathbf{S}$ must be cycle-free and therefore a tree in the graph-theoretic sense.

Contractions

A sample set $\mathbf{S}$ is a simple contraction of sample set $\mathbf{S}'$ if for every $\pi \in \mathbf{S}$ either

1. $\pi \in \mathbf{S}'$, or

2. there are $\pi_1, \pi_2 \in \mathbf{S}'$ such that the operation $\pi_1 \cup \pi_2$ is well defined, $\pi = \pi_1 \cup \pi_2$ and $\pi_1, \pi_2 \notin \mathbf{S}$.

Trees

A set of samples of $\mathbf{S}$ is a tree if there is a sequence $\mathbf{S}_1, ..., \mathbf{S}_n$ such that $\mathbf{S}_1 = \mathbf{S}$, $\mathbf{S}_{i+1}$ is a simple contraction of $\mathbf{S}_i$, and $\mathbf{S}_n$ is a primitive tree. The operators `root` and `parent` can be extended to be compatible across the contractions and by convention $\texttt{parent}((q, u, D)) = (q', u', D')$ implies that there is $t \in D'$ such that $F(q', u', t') = q$.

All definitions above will be used in Chapter 4 for the definition of the algorithm and Chapter 7 for the proof of its completeness.

3.4 Black Box Simulation

Up to now the discussion has focused on defining a formal mathematical framework for representing models of physical systems. To this end, the definitions in Section 3.2 were given. A 2D example was used to illustrate ideas, however this work is , in essence, directed towards practical results grounded in actual computation. For this reason, this section will develop the computational semantics of the model of Definition 3.2.1. The section begins with a discussion of the usual computational properties of physical system simulators and proposes the computational abstraction employed by this work.

3.4.1 The Black Box Computational Model so Far

Early practical motion planning algorithms leveraged specific structure present in problem categories to obtain results. For example, 2-D point robot path planning can be done efficiently using visibility-based decomposition [OY82, dBvKO97]. Decomposition and other explicit methods quickly gave way to early potential field [BL91] and sampling-based methods [KvLO96]. Interestingly, these newer algorithms could be applied to a larger class of planning problems. The application focus shifted towards exploiting computation-

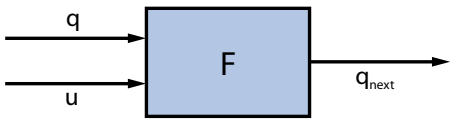

Figure 3.3 : Black box simulator

ally cheap primitives and the theoretical underpinnings of showing completeness results for algorithms composed from such primitives. Specifically, the last decade of progress in motion planning has been enabled by two cheap primitives:

1. collision detection for collections of 2-D and 3-D polyhedra and
2. proximity queries in general metric spaces.

3.4.2 The Black Box Computational Model in the Future

This thesis maintains that the next revolution in motion planning will be enabled from the use of physical simulators as black boxes. These simulators are not as cheap today as one would like to, but their capabilities and speed are increasing day by day. Physical simulators are driven by an independent and powerful industry, mainly games and entertainment. They

are bound to reach new levels of abstraction and performance. Motion planners that use physical simulators as black box primitives can exploit these advances.

When planning for a complex robotic system, a state transition equation may not be available. In the future and as we plan for more and more complex systems, state transition equations will in general not be available. It may still be possible to compute future states given the current state and a set of controls. As an extreme, take the example of a car which is modeled as a set of rigid parts that are connected together as in a real car. One can imagine different levels of abstraction starting from a car having four wheels, three axes and a frame, to a car having all of its parts each modeled with precision that may include the material properties of each part. Car manufacturers build increasingly accurate simulators to model the dynamics and the behavior of cars and test certain of aspects of their behavior in simulation before these cars are manufactured. Such simulators can be used as a black box in a planner. The input to the simulator is the current state and a set of controls as depicted in Figure 3.3. The output is a new state. By using a simulator the planning algorithm does not need to deal with the details and the complexity of the system representation. The equations, explicit or implicit, that govern the system's behavior are all encapsulated in the simulator, which has to decide what methods to use to integrate and/or solve the underlying equations. Issues related to discrete time integration, numerical precision, forward and backward integration, parallel kinematics and dynamics, friction models etc can all be hidden inside the simulator. Chapter 2 provides a glimpse to what modern simulators can do.

For the purposes of this thesis, a motion planning algorithm is a computable process that takes a `MPP` as input, and outputs either a solution $u_1, ..., u_n$ and $t_1, ..., t_n$ or a failure condition. If a `MPP` has a solution, it is called solvable.

The algorithm presented in the next chapter keeps a clean interface between the algorithm and the interface to the simulator and makes decisions that facilitate the use of a simulator.

Chapter 4

The PDST-EXPLORE Algorithm

4.1 Overview of Algorithm Operation

The algorithm presented in this chapter is called Path Directed Subdivision Tree exploration algorithm. We will refer to it as the PDST-EXPLORE algorithm. Figure 4.1 provides an illustration of the operation of the PDST-EXPLORE algorithm. The motion planning problem in this example is the same as in Figure 3.1. The small tree that is shown in Figure 4.1 is the same as the one constructed in Figure 3.2. It was constructed by hand but is strictly according to the rules of PDST-EXPLORE. The additional notation on the diagram shows some of the internal workings of PDST-EXPLORE.

The algorithm proceeds by iteratively adding branches to the tree; in the same way as the abstract schema presented in Section 3.3. The dotted lines denote a space partition of the state space into cells. The cell partition is refined incrementally, with a new split added at each iteration. The numbers beside individual path samples provide a labeling of the path segments.

It is important to note that as the cell partition is refined, the path samples are split accordingly to satisfy the invariant rule that every path sample is contained in exactly one cell. Formally, this representation for path sample and tree are supported by and motivate the notation introduced at the end of Section 3.3.

The decisions for selection, propagation and cell refinement are governed by a heuristic optimization scheme designed to satisfy the complete properties while providing a powerful

mechanism for exploiting problem structure.

The remainder of this chapter deals with the specifics of the PDST−EXPLORE. Experiments are given in Chapters 5 and 6. An analysis of the probabilistic completeness of this algorithm is given in Chapter 7 and a discussion is included in Chapter 8. This chapter is meant to precisely expose the details of the PDST−EXPLORE algorithm. The focus is for this to be a workable implementation guide as well as to provide notation that will be used in the theoretical analysis in Chapter 7. The algorithm is presented in entirety as Algorithm 1 and described in the body of the next section. The notation used and the definitions of certain key operators are given as they are introduced in the algorithm.

4.2 Algorithm Details

The object of this section is to provide a detailed description of PDST−EXPLORE which is given in its entirety as Algorithm 1. The methodology will be discuss the algorithm in a sequence of logical blocks, giving the relevant pseudo-code at the beginning of each segment and exposing the meaning. Rather than having a centralized discussion of the various data structures used in PDST−EXPLORE, they will be defined as they are introduced and used.

4.2.1 Explore Procedure (lines 1–27) and Initialization (lines 2–5)

```
 1: procedure EXPLORE(q_initial, N, G)      ▷ Find a path from q_initial to G in N iterations
 2:     SampleSet ← CreateSet()
 3:     InsertSet(SampleSet, q_initial)     ▷ Initial state is added to SampleSet
 4:     priority(q_initial) = 1             ▷ Priority of the initial state is 1
 5:     CellBSP ← CreateBSP(Q)              ▷ Cell BSP begins with the entire state space
 6:     ⋮
27: end procedure
```

The arguments to Explore are the initial state q_{initial}, a maximum number of iterations, N, and a goal region, G. There is an unspoken additional argument consisting of the "universe", i.e., the motion planning problem consisting of the state space $\mathcal{Q}$, the control

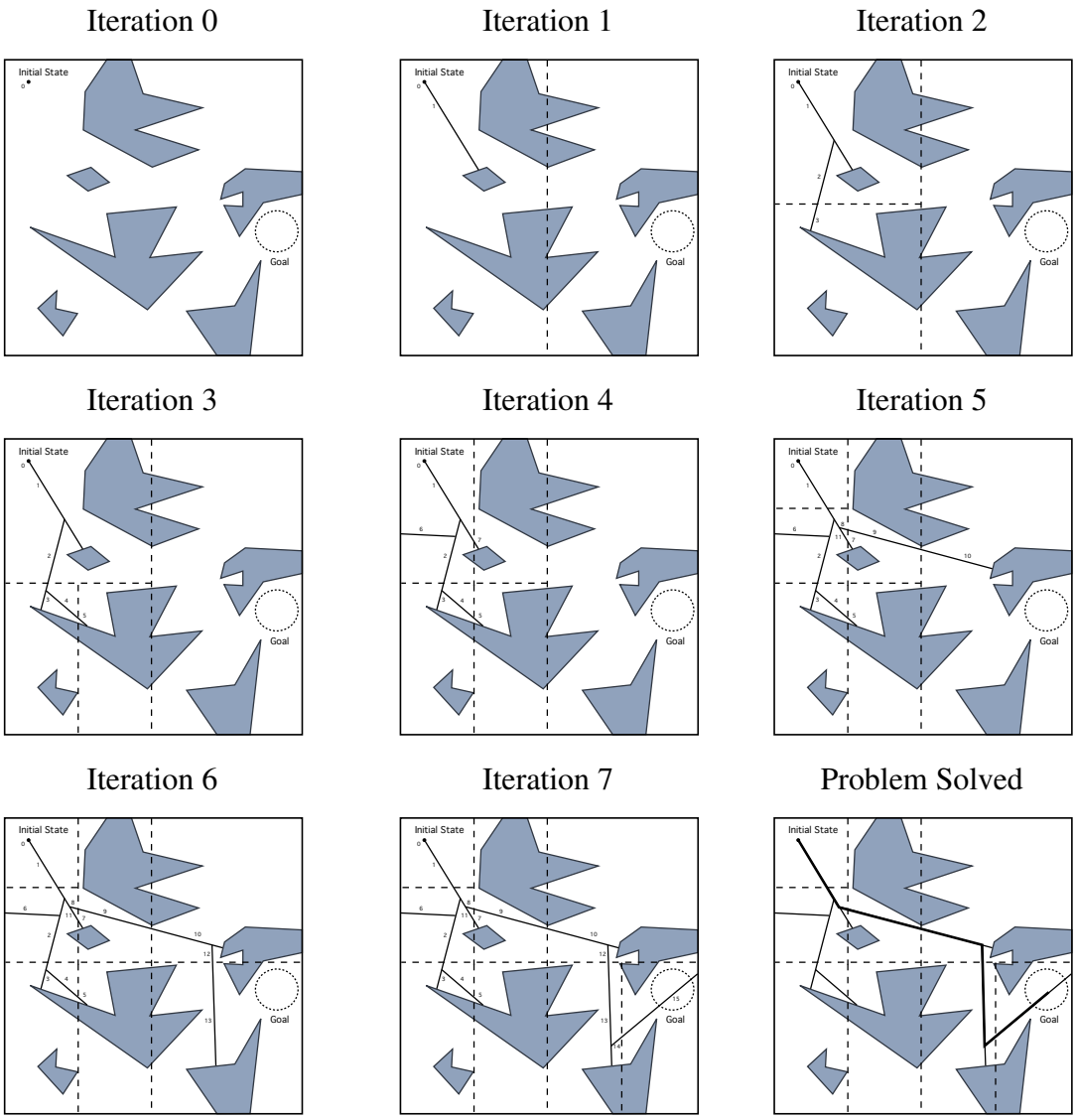

Figure 4.1 : An example of PDST−EXPLORE execution

space $\mathcal{U}$ and the transit function F. In my reference implementation, this information is attached as an object which deals with representing and manipulating states and controls, as well as providing the black box integrator for the system. Additionally, as part of the universe, are any integration parameters, e.g. time step and maximum path time, measures for $\mathcal{Q}$ and $\mathcal{U}$, and the subdivision scheme. These concepts are defined in the write-ups for the code blocks that use them.

The set **SampleSet** is a set of path samples and is initially created as an empty set on line 2. In Section 3.3, this set was referred to as $\mathbf{S}$ and is a set of path samples. It will always be a tree in the sense of the definitions of Section 3.2. As stylistic policy, verbose variable and operator names are used in the algorithm description to reduce the opacity of the notation. The **SampleSet** initially has the singleton path sample $(q_{\text{initial}}, \cdot, [0])$ inserted into it. The description on line 3 is slight abuse of notation and follows the rule that a state q interpreted as a path sample is the path sample $(q, \cdot, [0])$.

Line 4 introduces the priority operator. The operator `priority` maps path samples in **SampleSet** to positive integers. The priority of a path sample is used to construct an ordering over the path samples and to determine which segment is selected for branching. A lower priority value means it will be selected earlier.

Finally, on line 5, the cell subdivision introduced and Figure 4.1 is created. The set **CellBSP** consists of a set of cells which define a partition of the state space, $\mathcal{Q}$. The set **CellBSP** is initialized to a single cell, the state space $\mathcal{Q}$.

There is a relationship between **SampleSet** and **CellBSP**. An invariant is constructively maintained throughout the execution of `PDST-EXPLORE`. In plain language, the invariants states that every path sample in the sample set lies in exactly one cell from the subdivision.

Invariant 4.2.1. *At any step in the execution of* `PDST-EXPLORE` *the following property*

holds:

For every $\pi \in$**SampleSet**, *there exists* $C \in$ **CellBSP** *such that* $\pi \subset C$.

4.2.2 Outer Loop (line 6–25) and Failure Condition (line 26)

6: **for iteration** $\leftarrow 1$ to N **do**
7: $\quad\vdots$
25: **end for**
26: **return** No Path Found

The input variable N is the maximum number of iterations that PDST−EXPLORE tries before concluding that no path exists. After completing N iterations, the algorithm returns "No Path Found" on line 26.

4.2.3 Select (line 7)

7: $\pi_{\text{select}} = \pi \in$ **SampleSet** such that $\texttt{score}(\pi)$ is minimal ▷ Select

Line 7 shows the criterion that determines which path sample is selected for branching. The sample that is selected is the one that minimizes the scoring function. The scoring function is defined by

$$\texttt{score}(\pi) = \frac{\texttt{priority}(\pi)}{\mu_C(\texttt{cell}(\mathbf{C}, \pi))},$$

and the operator cell is given by

$$\texttt{cell}(\pi) = C \text{ such that } C \in \textbf{CellBSP} \text{ and } \pi \subset C.$$

In other words, the score of a path sample is the priority of that segment divided by the volume of the cell that contains it. The measure μ_C is a probability measure for the state space, i.e. $\mu_C(\mathcal{Q}) = 1$, and is an implicit parameter of the algorithm. The sample with the minimal score is chosen as the selected sample for that iteration, π_{selected}.

Figure 4.2 shows **SampleSet** on the 5th and 6th iterations of the PDST−EXPLORE run shown in Figure 4.1. The numbering in Figure 4.2 is consistent with the numbering

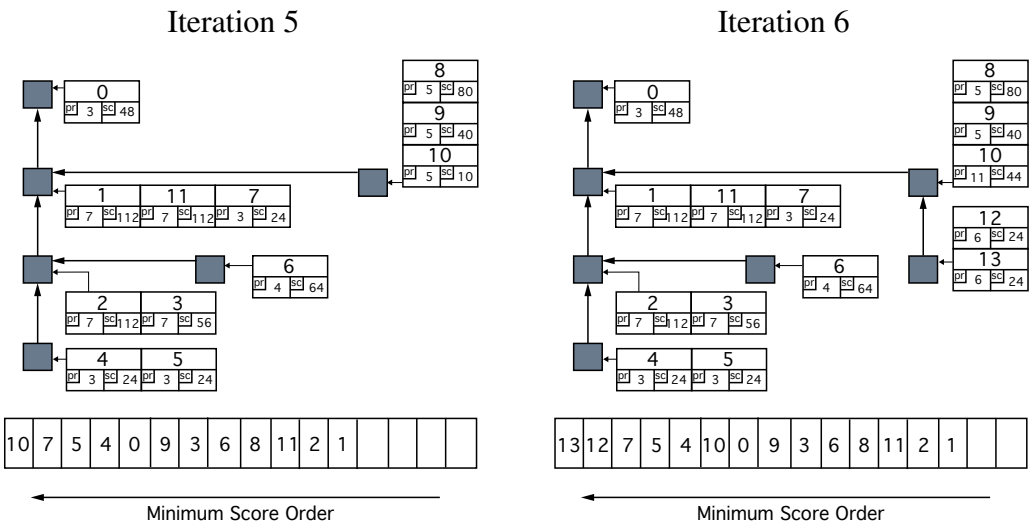

Figure 4.2 : Sample set

Figure 4.1. Each dark square represents a primitive path sample that appears in the tree and the arrows represent the branching relation. The entries associated with square are path samples that have been split to satisfy Invariant 4.2.1. Each entry contains its sample identifier, its priority and its score. At the bottom of Figure 4.2, the set of all path samples is ordered from left to right from least to largest score. At iteration 5, the path sample with the identifier 10 will be selected.

4.2.4 Propagate (line 8)

8: $\pi_{\text{new}} \leftarrow \texttt{integrate}(\pi_{\text{select}}(\texttt{uniform}(\texttt{domain}(\pi_{\text{select}}))), \texttt{randomControl}())$ ▷ Propagate

Line 8 describes how a new branch is added to the tree; this step is also called propagate. Suppose $\pi_{\text{select}} = (q, u, D)$. The operator $\texttt{domain}(\pi_{\text{select}})$ returns D, the domain of the path segment. The operator $\texttt{uniform}(D)$ returns a random $t \in D$, uniformly distributed over D using the usual distribution on the real line. Then π_{select} is evaluated at time t to produce a state $q' = \pi_{\text{select}}(t)$. Finally, a random control $u = \texttt{randomControl}()$ is

generated and a new primitive path sample is produced by integrating control u from state q', $\pi_{\text{new}} = \texttt{integrate}(q', u)$. The integrate operator is defined in the same way as in Section 3.2. The distribution used by `randomControl` and the value $T_{\max}$ in the integrator are implicit parameters to algorithm.

4.2.5 Check for Solution (line 9–11)

9: **if** $\pi_{\text{new}} \cap G \neq \emptyset$ **then** ▷ If the goal is reached...
10: **return** the $\texttt{pathTo}(\pi_{\text{new}}, G)$ ▷ then return the solution path
11: **end if**

Next PDST−EXPLORE checks to see if the newly generated path sample is a solution to planning problem. On line 9, it is checked if π_{new} intersects the goal region G. If not, then no solution was found and the algorithm proceeds, otherwise a solution was found and the solution path is returned.

The `pathTo` operator is defined as follows. $\texttt{pathTo}(\pi, G)$ returns the path given by first finding the longest sequence $\pi_1, ..., \pi_n$ with $\pi_i \in$**SampleSet**, $\pi_n = \pi$ and $\texttt{parent}(\pi_{i+1}) = \pi$. Noting that $\pi_1 = (q_{\text{initial}}, \cdot, [0])$ and then determining $t_1, ..., t_n$ such that $\pi_i(t_i) = \pi_{i+1}(0)$ and $\pi_n(t_n) \in G$, the solution path is given by, for $\pi_i = (q_i, u_i, D_i)$,

$$\texttt{pathTo}(\pi, G) = \texttt{path}(q_{\text{initial}}, u_1, t_1, ..., u_n, t_n).$$

4.2.6 Adjust Priorities (line 12–14)

12: $\texttt{priority}(\pi_{\text{select}}) \leftarrow 2 \cdot \texttt{priority}(\pi_{\text{select}}) + 1$ ▷ Penalize
13: $\texttt{priority}(\pi_{\text{new}}) \leftarrow$ **iteration** ▷ Initialize priority for new sample
14: $\texttt{InsertSample}(\pi_{\text{new}})$ ▷ Insert the new sample

In the case where no solution is found, the new sample is inserted into the tree. Before this occurs, the priority for the selected sample, π_{select}, must be adjusted and the priority of the new sample must be set. This is governed by line 12 and 13 and the rules are:

1. priority is doubled after selection

2. new priorities are set to the current iteration count

The basic intuition is that selection of a given sample should occur exponentially infrequently. Secondly, there is a linear ordering on new samples, i.e. excluding bias introduced by the cell volumes, a sample created on iteration 7 should be selected before a sample created on iteration 12. A more formal explanation of why these are good rules is given in Chapter 7.

After setting the priorities, the new sample is inserted on line 14 by calling the insertion subroutine.

4.2.7 Insertion Subroutine (line 28–39)

28: **procedure** INSERTSAMPLE(π) ▷ Insert path sample π into the **SampleSet** splitting if necessary
29: $C \leftarrow$ `StabBSP`(**CellBSP**,π) ▷ Find the cell that π begins in
30: **if** $\pi \subset C$ **then** ▷ If π is entirely contained in cell C
31: `InsertSet`(**SampleSet**, π) ▷ The sample was contained in one cell
32: **else**
33: $\pi_1 \leftarrow \pi \cap C$ ▷ Intersect π with C
34: $\pi_2 \leftarrow \pi - \pi_1$ ▷ π_2 is the path sample not in C
35: `priority`(π_1) $\leftarrow$ `priority`(π_2) $\leftarrow$ `priority`(π) ▷ Priority is inherited
36: `InsertSet`(**SampleSet**, π_1) ▷ Add π_1 to the sample set
37: `InsertSample`(π_2) ▷ Recursively insert π_2
38: **end if**
39: **end procedure**

The insertion subroutine deals with inserting given path sample, π, which may be recursively split in order to maintain Invariant 4.2.1. Line 29 determines the first cell containing the path sample π. Formally, the operator is defined by, for $\pi = (q, u, D)$,

`StabBSP`(**CellBSP**, π) $= C \in$ **CellBSP** such that there is $t \in D$ for all $t' \in D, t' \leq t$, $\pi(t') \in C$.

On line 30, the procedure checks if π is contained entirely in C. If so, the direct insertion into **SampleSet** is sufficient on line 31 and nothing else needs to be done. Otherwise, on

line 33, π_1 is set to the part of π that intersects C, i.e.

$$\pi_1 = (q, u, D_1) \text{ where } D_1 \subset D \text{ maximal s.t. } t \in D_1, F(q, u, t) \in C.$$

Similarly, line 34 sets π_2 as the remainder, $\pi_2 = (q, u, D - D_1)$. The priorities of the split samples π_1 and π_2 are inherited from the priority of π. Since π_1 is entirely contained C, it is directed inserted into **SampleSet** and a call to `InsertSample` occurs recursively on π_2 on line 37.

4.2.8 Subdivision (line 15–24)

15: $C_{\text{select}} \leftarrow \texttt{cell}(\pi_{\text{select}})$ ▷ This cell will be subdivided
16: **ReinsertionSet** $\leftarrow$ `CreateSet()`
17: **for** $\pi \in$ **SampleSet** such that $\texttt{cell}(\pi) = C_{\text{select}}$ **do**
18: `InsertSet`(**ReinsertionSet**, π) ▷ Accumulate all samples contained in the cell
19: `RemoveSet`(**SampleSet**, π) ▷ Remove them from the sample set
20: **end for**
21: `SubdivideCellBSP`(**CellBSP**,C_{select}) ▷ Subdivide the cell
22: **for** $\pi \in$ **ReinsertionSet do**
23: `InsertSample`(π) ▷ Reinsert all samples from the subdivided cell
24: **end for**

Before the end of an iteration of `PDST-EXPLORE`, the cell that contained π_{select} is subdivided. On line 15, C_{select} is set to the cell that contains π_{select}, $\texttt{cell}(\pi_{\text{select}})$. On the subsequent line, a temporary set called **ReinsertionSet** is created. The **ReinsertionSet** will contain all $\pi \in$ **SampleSet** such that $\texttt{cell}(\pi) = C_{\text{select}}$. The **for** loop from lines 17–20, removes all of the path samples contained in the cell C_{select} from the sample set, **SampleSet**, and places them in the **ReinsertionSet**. On line 21, the cell C_{select} is subdivided. The effect is that it is partitioned into two new pieces which are added to **CellBSP** and C_{select} is removed from the **CellBSP**. The scheme for subdividing cells is an implicit parameter of the algorithm. The **for** loop on lines 22–24 reinserts all of samples in the **ReinsertionSet** using the `InsertSample` procedure where they may be split if necessary.

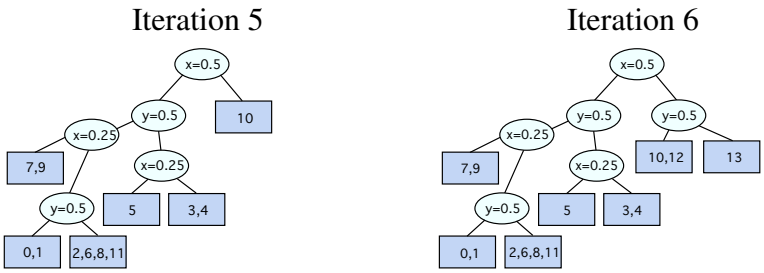

Figure 4.3 : Cell subdivision

The illustration in Figure 4.3 shows cell binary space partitions for iterations 5 and 6 of the execution of `PDST-EXPLORE` in Figure 4.1.

4.3 The Full Algorithm

Algorithm 1 contains all the full algorithm whose individual parts have already been explained in this chapter.

Algorithm 1 `PDST-EXPLORE` Pseudo-Code

procedure EXPLORE(q_{initial}, N, G) ▷ Find a path from q_{initial} to G in N iterations
  SampleSet ← `CreateSet()`
  `InsertSet`(**SampleSet**, q_{initial}) ▷ Initial state is added to **SampleSet**
  `priority`(q_{initial}) $= 1$ ▷ Priority of the initial state is 1
  CellBSP ← `CreateBSP`($\mathcal{Q}$) ▷ Cell BSP begins with the entire state space
  for iteration ← 1 to N **do**
    $\pi_{\text{select}} = \pi \in$ **SampleSet** such that `score`(π) is minimal ▷ Select
    $\pi_{\text{new}} \leftarrow$ `integrate`(π_{select}(`uniform`(`domain`(π_{select}))), `randomControl`()) ▷ Propagate
    if $\pi_{\text{new}} \cap G \neq \emptyset$ **then** ▷ If the goal is reached...
      return the `pathTo`(π_{new}, G) ▷ then return the solution path
    end if
    `priority`(π_{select}) ← $2 \cdot$ `priority`(π_{select}) $+ 1$ ▷ Penalize
    `priority`(π_{new}) ← **iteration** ▷ Initialize priority for new sample
    `InsertSample`(π_{new}) ▷ Insert the new sample
    $C_{\text{select}} \leftarrow$ `cell`(π_{select}) ▷ This cell will be subdivided
    ReinsertionSet ← `CreateSet()`
    for $\pi \in$ **SampleSet** such that `cell`(π) $= C_{\text{select}}$ **do**
      `InsertSet`(**ReinsertionSet**, π) ▷ Accumulate all samples contained in the cell
      `RemoveSet`(**SampleSet**, π) ▷ Remove them from the sample set
    end for
    `SubdivideCellBSP`(**CellBSP**,C_{select}) ▷ Subdivide the cell
    for $\pi \in$ **ReinsertionSet do**
      `InsertSample`(π) ▷ Reinsert all samples from the subdivided cell
    end for
  end for
  return No Path Found
end procedure
procedure INSERTSAMPLE(π) ▷ Insert path sample π into the **SampleSet** splitting if necessary
  $C \leftarrow$`StabBSP`(**CellBSP**,π) ▷ Find the cell that π begins in
  if $\pi \subset C$ **then** ▷ If π is entirely contained in cell C
    `InsertSet`(**SampleSet**, π) ▷ The sample was contained in one cell
  else
    $\pi_1 \leftarrow \pi \cap C$ ▷ Intersect π with C
    $\pi_2 \leftarrow \pi - \pi_1$ ▷ π_2 is the path sample not in C
    `priority`(π_1) ← `priority`(π_2) ← `priority`(π) ▷ Priority is inherited
    `InsertSet`(**SampleSet**, π_1) ▷ Add π_1 to the sample set
    `InsertSample`(π_2) ▷ Recursively insert π_2
  end if
end procedure

Chapter 5

Experiments with Simplified Physics

This chapter presents several experimental results using PDST−EXPLORE. The cases considered in this chapter are cases where the physics of the underlying system are fairly simple and in some cases extensive prior work can be used for trajectory generation. First a 2D Kinodynamics robot, a second-order differential drive robot and a blimp robot are considered. Then we present a game, called the Games of Koules which is a second-order dynamical multi-agent system, and which exemplifies many of the characteristics of problems that are difficult for current planners. The material of this chapter has been presented in [LK05a, LK05b].

5.1 2D Kinodynamic, Differential Drive and Blimp Robots

In this section, we begin by describing a novel extension to Maneuver Automata theory [Fra] which uses PRM sampling to generate Maneuver Automata. The generated automata are then used for trajectory generation for the robots considered in this section. We continue by outlining the specific robot systems we have implemented PDST−EXPLORE for. Finally, we describe our experiments and present our experimental results.

5.1.1 Maneuver Automata

We propose using the Maneuver Automata [FDF05] to implement the integrate function used in our exploration planner. This is novel extension to the Maneuver Automata

literature and can applied to any planner that uses propagation for a robot that satisfies the symmetry property we have described in this subsection. The advantage we gain is that we can restrict ourselves to nice family of motions and eliminate numerical integration from the call to `integrate`. These techniques only apply when special structure exists in the motion of the robot, however the class of applicable robot systems is an important one. A sufficient condition occurs when the state space Q is a direct product of a Lie Group G and a shape manifold Z. Furthermore, the Lie Group G operating on Q must preserve path feasibility [Fra]. If this is the case, we say that G is a symmetry group for Q and Maneuver Automata theory can be applied.

For our purposes a Maneuver Automata is a finite directed multi-graph $MA = (V, E)$. The vertex set V is a finite subset of Z. An element of E is (z_1, λ, z_2) is a directed edge between vertices z_1 and z_2 together with a control function $\lambda : [0, T] \rightarrow U$ of duration T. The control function λ must satisfy the property that if $q_1 = (e, z_1)$ then result of integrating the control function λ starting at state q_1 produces a state $q_2 = (g, z_2)$ where g can be arbitrary. In other words, λ gives a control schedule for transitioning from any state with shape z_1 to some state with shape z_2. Such a transition defines a set of paths equivalent under G-symmetries which is called a *maneuver* motion. A second kind of motion can be effected from a state $q = (g, z)$ where $z \in V$ by executing the zero control $0 \in U$ for any amount of time. These motions can be called *trim* motions and can be represented by Lie group exponentiation. Precisely, if $q = (g, z)$ for $z \in V$ and for any time $t \geq 0$ there exists $g_z \in G$ such that integrating the constant control $0 \in U$ starting at q produces a motion $\alpha(t)$ which can be written as $\alpha(t) = (g \exp(g_z, t), z)$. The Maneuver Automata can be used to generate motions by alternating between the fixed duration maneuver motions and the anytime trim motions. If an automata MA satisfies certain properties then the set $Q_{MA} = \{q = (g, z) : g \in G \text{ and } z \in V\}$ is strongly connected by these generated motions.

The graph $MA = (V, E)$ is a kind of roadmap in shape space. Continuing this analogy, we can sample MA using PRM techniques. So, given $z_1, z_2 \in Z$, we need a local planning primitive which can compute a control function which defines a maneuver motion that connects them. The local planner and the sampling distribution over Z can be used to implement PRM. It seems likely that Quasi-Random-Lattice methods might be particularly effective for this task [LB02].

5.1.2 Robot Systems

In this subsection, we describe the robot systems for which we implemented PDST−EXPLORE. In each case, we specify the state space, the dynamics, inequality constraints, the symmetries, primitive trajectories and cell subdivision scheme.

5.1.3 2-D Kinodynamic Robot

The state space for the 2-D kinodynamic robot is denoted by $Q = \mathbb{R}^2 \times \mathbb{R}^2$. A state $q = (x, y, \dot{x}, \dot{y})$, where (x, y) is the robot's position in the plane and $(\dot{x}, \dot{y})$ is the robot's velocity. The symmetry group we use for this robot is $G = \mathbb{R}^2$, the group of 2-D translations. The shape manifold for the robot is $Z = \mathbb{R}^2$ and represents positionless velocities. Every $z = (\dot{x}, \dot{y}) \in Z$ defines a trim primitive $[\alpha_z]$ and corresponds to the straight line at that fixed velocity. Maneuvers between trim primitives consist of the robot acceleration towards the different velocity state at maximum acceleration. We impose a constraint that velocity is bounded, i.e. $||(\dot{x}, \dot{y})|| \leq v_{\max}$. The subdivision scheme builds a kd-tree in the state space by equal splits on the first and second dimensions.

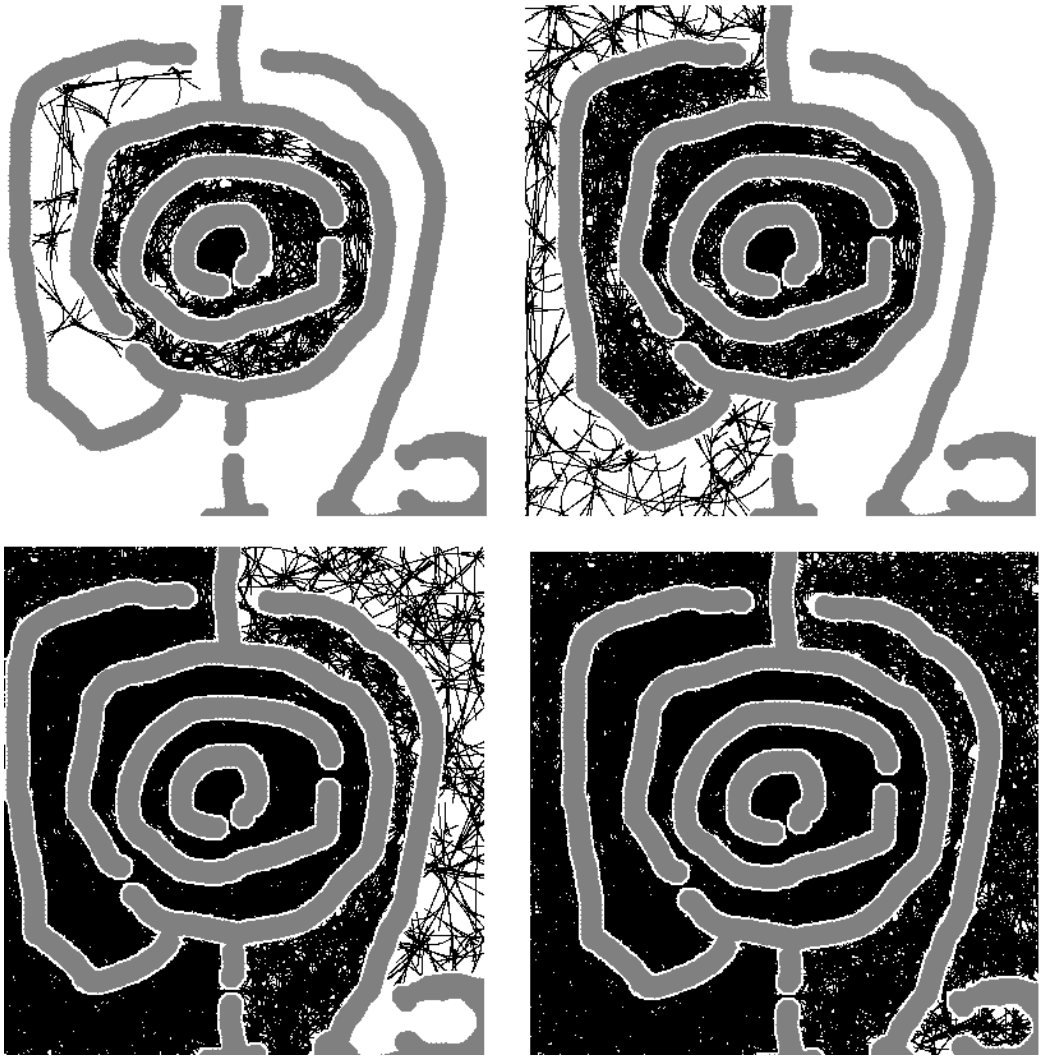

Figure 5.1 : Execution snapshots of PDST−EXPLORE for a differential drive robot

5.1.4 Differential Drive Robot

The state space for this robot is $Q = \mathbb{R}^2 \times S \times \mathbb{R}^2$. A state is given by $q = (x, y, \theta, v_l, v_r)$. The vector (x, y, θ) is the robot's position and orientation and (v_l, v_r) are the robot's wheel velocities. The symmetry group we use for this robot is $G = \mathbb{R}^2 \times S = SE(2)$, the group of 2-D rigid motions. The shape manifold for the robot is $Z = \mathbb{R}^2$ and represents positionless wheel velocities. Every $z = (\dot{x}, \dot{y}) \in Z$ defines a trim primitive $[\alpha_z]$. The canonical path for the trim primitive defined by $z = (v_l, v_r)$ is

$$\alpha_z(t) = \begin{cases} (v_f t, 0, 0, v_l, v_r) & \omega = 0 \\ (\frac{v_f}{\omega}\sin(\omega t), \frac{v_f}{\omega}(1 - \cos(\omega t)), \omega t, v_l, v_r) & \omega \neq 0, \end{cases}$$

where $\omega = \frac{v_r - v_l}{L}$, $v_f = \frac{v_l + v_r}{2}$ and L is the length of the wheel base of the robot. Let $z^1 = (v_l^1, v_r^1)$ and $z^2 = (v_l^2, v_r^2)$. The maneuver primitive that brings z^1 to z^2 is $[\pi_{z^1 z^2}]$. It is determined by $a_{\max}$, the maximum wheel acceleration for the robot. Let

$$T = \max\left\{\left|\frac{v_l^2 - v_l^1}{a_{\max}}\right|, \left|\frac{v_r^2 - v_r^1}{a_{\max}}\right|\right\}$$

be the duration of $[\pi_{z^1 z^2}]$ which has canonical representative

$$\pi_{z^1 z^2}(t) = \left(v_l^1 + (v_l^2 - v_l^1)\frac{t}{T}, v_r^1 + (v_r^2 - v_r^1)\frac{t}{T}\right).$$

We impose a constraint which bounds the maximum wheel velocity, $|v_l|, |v_r| \leq v_{\max}$. The subdivision scheme we use for this space builds a kd-tree in the state space and uses equal splits on the first, second and third dimensions.

5.1.5 Blimp Robot

The the state space for this robot is $Q = \mathbb{R}^3 \times S \times \mathbb{R}^3 \times S$. A state $q = (x, y, z, \theta, \dot{x}, \dot{y}, \dot{z}, \dot{\theta})$ is the robot's position, orientation and velocity. The symmetry group that we use for this robot

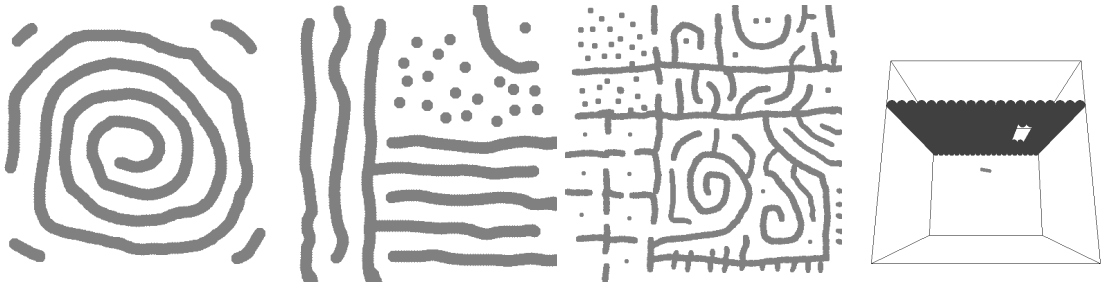

Figure 5.2 : Workspaces (from left to right) spiral-1, varied-1, varied-2 and slot

is $G = \mathbb{R}^3$, the group of translations in 3-D. The shape manifold for this robot is $\mathbb{R}^3 \times S$. Every $z = (\dot{x}, \dot{y}, \dot{z}, \dot{\theta}) \in Z$ represents the velocities of the robot. Each $z = (\dot{x}, \dot{y}, \dot{z}, 0)$ defines a trim primitive $[\alpha_z]$. The canonical path for the trim primitive defined by such a z is $\alpha(t) = (\dot{x}t, \dot{y}t, \dot{z}t, 0, \dot{x}, \dot{y}, \dot{z}, 0)$. The controls for this robot are a_f, a_z and a_θ. The robot is subject to the following constraints: $\ddot{x} = \cos(\theta)a_f$, $\ddot{y} = \sin(\theta) \cdot a_f$, $\ddot{z} = a_z$ and $\ddot{\theta} = a_\theta$. Furthermore, $a_f \in [0, a_f^{\max}]$, $a_z \in [-a_{\max}^z, a_{\max}^z]$ and $a_\theta \in [-a_{\max}^\theta, a_{\max}^\theta]$. In particular, since a_f must be positive the robot's motion is highly constrained. The calculation of the maneuver primitives are accomplished using a controller which tries to minimize the amount of time taken to switch between two shapes. During the switch, the z-dimension is controlled independently and the controller attempts to minimize the change in z by keeping $|\dot{z}| = 0$ for as long as possible. The controller that we use is expensive to compute but is effective at minimizing the time used. The trajectories taken through shape space to connect two shapes are not symmetric for this robot and can differ greatly in the total time used. Finally, the time step used to integrate the motion of the robot is very small in the controller and once the path is computed we resample using a variable sized time step which approximates the motion in the state space. The subdivision scheme we use for this robot builds a kd-tree in the state space and uses equal splits on the first, second, third and fourth dimensions.

5.1.6 PDST-EXPLORE Experiments

In Figure 5.2, we depict some of the workspaces in which the experiments were carried out. In each experiment, a maneuver automata was built offline which took less than two seconds for the kinodynamic robot and differential drive robot and between 50 and 70 seconds for the blimp example. During the experiment, the maneuver automata was loaded off the disk and the PDST-EXPLORE planner was run until the measured dispersion [LB02] in the free space became very small. Dispersion was measured on a high-resolution cell grid. Cells containing an obstacle were not considered in the dispersion measure. The threshold we used ensured that over 0.999 of the space was covered. The number of iterations was then reported. In every example, 384 trials were carried out. The number of iterations required to solve the problem tended to be very similar to the mean number of iterations with the occasional outlier requiring between two and six times more iterations. In Figure 5.1, we show snapshots of the execution of the exploration of the free space for a differential drive robot. The example in question is referred to as chambers-1. The time costs and collision detect calls were very consistent across multiple runs. The raw data is presented in Figure 5.3. Experiments were carried on a standard 2004 technology desktop. Cost in time per iteration is roughly $O(n \log n)$ experimentally, which is expected because of the binary heap.

5.2 The Game of Koules

Our version of the game of Koules takes place in a 2-D workspace, specifically a square. There are two types of robots inside the workspace: a single ship and the koules. The ship is controlled by the user and the koules follow independent trajectories. When a robot touches the boundary of the workspace, it is killed. The user loses the game if the ship is

problem	robot	avg. # iterations	avg. # time	avg. # collision detects
spiral-1	kino	54205	0.76 s	51274
chambers-1	kino	95963	1.88 s	94112
varied-1	kino	76549	1.28 s	77974
varied-2	kino	431736	5.8 s	290904
spiral-1	dd	86000	4.6 s	71808
chambers-1	dd	282708	13.9 s	160350
varied-1	dd	288067	22.9 s	297662
varied-2	dd	1069687	66.7 s	717530
six	blimp	10000	3.4 s	391737
slot	blimp	65000	22.0 s	2515246

Figure 5.3 : Average running times to obtain full coverage

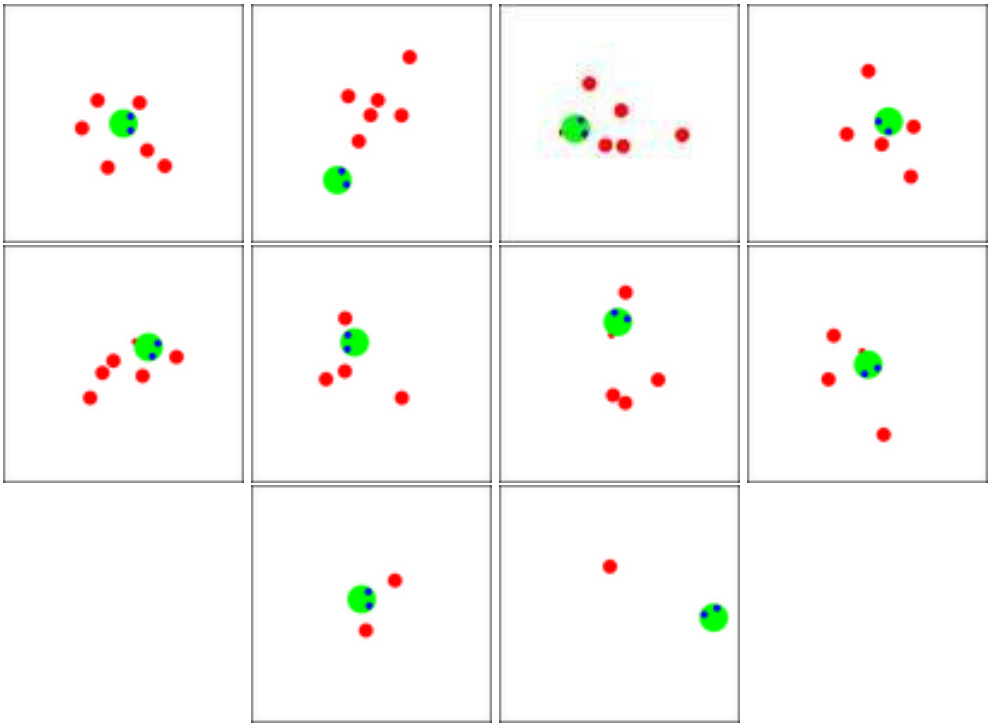

Figure 5.4 : Execution snapshots for a solution to the game of Koules with 6 koules

killed and the user wins the game if all of the koules are killed. When two robots touch, an elastic collision occurs and the robots bounce away from each other. The ship is capable of four different actions that the user can control: to cruise, to turn left or right at a constant speed, or to apply a constant thrust in the direction of the ship's current heading. The koules are attracted towards the center by a damped spring which makes them difficult to push towards the sides. The user can only influence the koules by colliding with them. An illustration is offered in Figure 5.4.

Solving an instance of the game of Koules requires the generation of sequence of timed controls such that the ship survives and all of the koules are killed. In the remainder of this section, we describe the implementation of our version of the game of Koules. In the next section, we describe the planner that we use to solve input instances of the game.

5.2.1 State Space and Controls

We begin by describing the state and control spaces. The state space for the game of Koules with n koules is determined as follows:

$$Q_n = ([0,1]^2 \times S^1 \times \mathbb{R}^2) \times ([0,1]^2 \times \mathbb{R}^2)^n.$$

A state $q = (x_s, \theta, v_s, x_1, v_1, ..., x_n, v_n)$ determines the position, x_s, heading, θ, and velocity v_s of the ship together with the positions, $x_1, ..., x_n$ and velocities $v_1, ..., v_n$ of the koules.

There are four distinct control inputs in the set of controls for the game of Koules, $U = \{u_0, u_L, u_R, u_1\}$, which correspond to cruise, u_0, turn left, u_L, turn right, u_R, and thrust, u_1.

An instance of the game of Koules consists of n, the number of koules and an initial state $q_0 \in Q_n$. A partial solution to that instance is a path π of duration T such that at state $\pi(T)$, a koule touches the boundary and no boundary collisions occur on the path before

time T. A full solution is a sequence of paths $\pi_n, ..., \pi_1$ with durations $T_n, ..., T_1$ such that for all $i < n$, π_i is a partial solution to the instance $(i, \pi_{i+1}(T_{i+1}))$.

5.2.2 The Dynamic System

The game of Koules is a second-order dynamic system. The motion of the ship is determined by its state and the control input using the following equations:

$$\begin{bmatrix} \dot{x_s} \\ \dot{\theta} \\ \dot{v_s} \end{bmatrix} = \begin{bmatrix} v_s \\ v_\theta \\ R(\theta) \cdot [a\ 0]^T \end{bmatrix} \tag{5.1}$$

where v_θ is the turning speed, $R(\theta)$ is the rotation matrix in $SO(2)$ determined by θ and a is the thrust. The turning speed, v_θ, and thrust, a are determined as functions of the current control input $u \in U$,

u	$v_\theta(u)$	$a(u)$
u_0	0	0
u_L	$v_\theta^{\max}$	0
u_R	$-v_\theta^{\max}$	0
u_1	0	$a^{\max}$

.

The motion of each koule is determined by its state and the position of the ship using the following damped spring equation:

$$\begin{bmatrix} \dot{x}_i \\ \dot{v}_i \end{bmatrix} = \begin{bmatrix} v_i \\ (o - x_i) \cdot \lambda_c - v_i \cdot h \end{bmatrix} \tag{5.2}$$

where o is the center of the workspace, λ_c is spring constant attracting towards the center and h is a friction parameter.

In the simulator, control inputs are applied over a fixed timestep Δt and numerically integrated with a fourth-order Runge-Kutta-Nystrom method [AS74].

5.2.3 Rules for Elastic Collisions

During each time-step of the simulator must simulate the system to generate the state that results from applying the current control, $u \in U$, to the initial state. This is a two-step process: first, a numerical integration of the equations of motion and followed by a discrete event simulation to resolve any collisions.

At the beginning of the time-step, the system is in state q^0. The result of integrating the control u for time Δt is a new state, q^f. However, although q^0 is collision-free, it is possible that collisions between robots or between the robots and the boundary of the workspace occur along the path between q^0 and q^f. In order to calculate collisions and the results of the induced velocity changes, a locally linear approximation is used and first-order motions are simulated with a discrete event simulator. To begin with, a new initial state,

$$q^+ = (x_s^+, \theta^+, v_s^+, x_1^+, v_1^+, ..., x_n^+, v_n^+),$$

is constructed from q^0 and q^f as follows: $x_s^+ = x_s^0$, $\theta^+ = \theta^0$, $x_i^+ = x_i^0$, $v_s^+ = (x_s^f - x_s^0)/\Delta t$ and $v_i^+ = (x_i^f - x_i^s)/\Delta t$.

All robots are then assumed to begin at q^+ and to move along the lines determined by their velocities during the discrete event simulation. If there are no collisions, after time Δt has elapsed, the system will reach a state with the same positions as state q^f and with the velocities of state q^+. The velocities are constant along the time step and are approximately correct with error linearly proportion to Δt.

The events in the discrete event simulation occur when a pair of robots collide or when a robot touches the boundary. The ship has radius r_s and mass m_s. Each koule has radius

r_k and mass m_k.

Pairwise collisions occur when the distance between two robots is equal to the sum of their radii. This is predicted by the solution of the appropriate quadratic equation. It is best to use iterative root polishing to avoid simulation errors caused by near singular states. Collisions with the boundary are determined by solving linear equations. Inter-robot collisions are resolved by applying the 1-D elastic collision formula and boundary collisions end the simulation.

The minimum amount of information required to store a path is the initial state q^0 and a sequence of timed control inputs: $0 = t_0, u_1, t_1, ..., t_{m-1}, u_m, t_m$ where the input u_i is applied from time t_{i-1} to time t_i and $u_i \neq u_{i+1}$. In order to reconstruct the state, q^t, at time t the integrator and discrete event simulator must be run. Our implementation stores key frames at times were collisions occurred and with a certain minimum density to reduce the amount of integration that needs to be done during interpolation while maintaining a compact representation for path data.

5.2.4 Trajectory Generation

Let γ be a path segment of duration T. The operation `propagate`(γ) creates a path segment π branching from γ. There are many possible choices for the `propagate` operation and the performance of the `PDST-EXPLORE` planner depends on this choice. We have observed that the following design principles are good choices: an iterated sequence of calls to `propagate` should be able to approximate any given path with some non-zero probability and a short sequence of iterated calls should extend into the local space around the initial segment. These principles were taken into the design and testing of the trajectory generation scheme which was used in the planner described in this paper. We now present `propagate` in Algorithm 2.

Algorithm 2 `propagate`(π)

Generate uniformly at random $t \in [0, |\pi|]$.

Let $q^0 := \pi(t)$.

Let x_s^0 be the ship's position at q^0.

Generate $x \in [0, 1]^2$ uniformly and at random.

Generate $v_s^{\text{mag}} \in [v_s^{\min}, v_s^{\max}]$.

Set $v_s^{\text{targ}} := v_s^{\text{mag}} \frac{x - x_s^0}{||x - x_s^0||}$.

for i ranges from 0 to $N_{\max}$ **do**

    Let v_s be the ship velocity of state q^i.

    Let θ be the ship direction of state q^i.

    Let $v := v_s^{\text{targ}} - v_s$.

    Let θ^{targ} be the direction of vector v.

    Let $\Delta\theta := \theta^{\text{targ}} - \theta$.

    if $|v| < \delta$ **then** $u = u_0$.

    else if $|\Delta\theta| < \epsilon$ **then** $u = u_1$.

    else if $\Delta\theta > 0$ **then** $u = u_L$.

    else $u = u_R$.

    end if

    Let $q^{i+1} :=$ `simulate`(q^i, u).

    if q^i is a terminal state **then return** the path $\{q^0, ..., q^i\}$.

end for

return $\emptyset$.

Algorithm 2 incrementally constructs a path by running a controller with the simulator. The operation `simulate`(q^i, u) is the result of running the simulator to compute the state that results from applying control u for time Δt from state q^i. The controller is designed to change the ship's velocity into a given target velocity. The target velocity has a random magnitude. Its direction is towards a randomly and uniformly chosen point in the workspace (unit square) from the ship's position at initial state of the new path. The initial state is chosen randomly from the states along the path being branched, π. The controller runs until the ship or a koule collides with the boundary or until $N_{\max}$ iterations have occurred.

In lines $4, 5$ and 6 of Algorithm 2, the target velocity is computed. Notice the biased sampling that occurs as a function of the ship's current position. When the ship is close to the boundary of the workspace, the target velocity will tend to move away from the boundary. The target velocity is sampled this way to reduce the probability that the ship will collide with the boundary at the beginning of the path.

Algorithm 2 has several external parameters: the maximum number of iterations, $N_{\max}$, the minimum and maximum velocity magnitudes, $v^{\min}$ and $v^{\max}$ respectively, and the switching bounds for the controller, δ and ϵ. Choosing $\delta = \frac{a^{\max} \cdot \Delta t}{2}$ and $\epsilon = \frac{v_\theta^{\max} \cdot \Delta t}{2}$ guarantees stability.

5.2.5 Coverage Estimation

The subdivision scheme used in our implementation was relatively unsophisticated. Initial tests determined that subdividing the velocity dimensions led to poor performance. Consequently, the scheme we employed only worked on the position dimensions: $x_s, \theta, x_1, ..., x_n$. The variables were subdivided in that order and we employed uniform splits. In an example with n koules, the coverage space is $3 + 2n$-dimensional and the state space is $5 + 4n$-

dimensional. The measure μ is uniform probability measure on $\mathbb{R}^2 \times S^1 \times \mathbb{R}^{2n}$.

5.2.6 Full Solution Algorithm

The PDST−EXPLORE planner creates partial solutions. In order to construct a full solution, a sequence of partial solutions must be generated. It is possible that the endpoint of a partial solution may leave the system in a state from which no further solution exists. Therefore the full solution planner needs a backtracking mechanism. The method presented as Algorithm 3 is very simple but was quite effective for the purposes of the game of Koules. The method proceeds recursively: PDST−EXPLORE is invoked to find a solution and if one is found then Algorithm 3 runs on the end state of the solution path. If repeated invocations of PDST−EXPLORE fail to find a solution or if the recursive calls fail, then the recursion stack pops one level and another attempt is made.

Algorithm 3 SOLVE$(n, q^n, N_{\text{iter}}, N_{\text{attempts}})$

1: **for** i ranges from 1 to N_{attempts} **do**
2: Let $\pi^n :=$ PDST−EXPLORE(q^n, N_{iter}).
3: **if** $\pi^n = \emptyset$ **then continue**.
4: **if** $n = 1$ **then return** π^1.
5: Let q^{n-1} be the endpoint of π^n.
6: Let $\pi^{n-1} :=$ SOLVE$(n-1, q^{n-1}, N_{\text{iter}}, N_{\text{attempts}})$.
7: **if** $\pi^{n-1} \neq \emptyset$ **then return** $\pi^n \circ \pi^{n-1}$.
8: **end for**
9: **return** $\emptyset$.

5.2.7 Koules Experimental Methodology

Two different kinds of experiments were run to establish evidence for our claims: partial solutions and full solutions. The partial solution experiments were run for various numbers of koules. They use `PDST-EXPLORE` to search for paths that eliminate a koule. The planner is allowed to continue after finding a solution and may generate many solutions. The full solution experiments were also run for various numbers of koules and uses Algorithm 3 to construct a sequence of partial solutions each, in turn, generated with `PDST-EXPLORE`.

The experiments were conducted on a cluster of 16 dual AMD 1900MPs with 1 GB of RAM running Debian unstable with the 2.4.18 Linux kernel. The code is written in `C/C++/fluid` and uses the `FLTK`, `GLUT`, `OpenGl` and `S-Lang` packages. Throughout the experiments, the following parameters were used: $v_\theta^{\max} = \pi$, $a^{\max} = 1$, $\lambda_c = 4$, $h = 0.05$, $m_s = 0.75$, $m_k = 0.5$, $r_s = 0.03$, $r_k = 0.015$ and $\Delta t = 0.005$. These parameters were set to create a challenging motion planning task and were tuned by using an interactive interface to the game. With these parameters, we found that human players in our research group were not able to solve examples with more than a few koules.

5.2.8 Partial Solutions

In this set of experiments, we measure the cost per iteration of `PDST-EXPLORE` during partial solutions. Each run was for 60000 iterations and worked on a randomly generated problem instance. The data was merged and averaged from 80 runs, but for these results there was very little variations. In Figure 5.5, we see the total time in seconds versus the iteration counter. Although running N iterations of Algorithm 1 is guaranteed to take at least time proportional to $N \log N$, the timing plots are very close to linear. This is explained by observing that most of the runtime is spent in the simulator. The additional cost of the `PDST-EXPLORE` algorithm is a slight super-linear cost due to the binary space

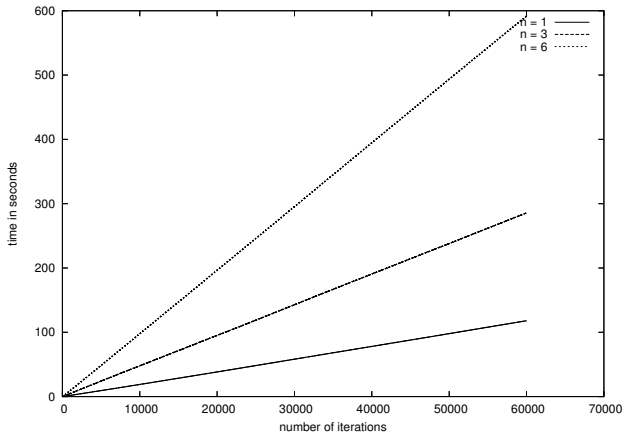

Figure 5.5 : Average time spent versus number of iterations for $1, 3$ and 6 koules

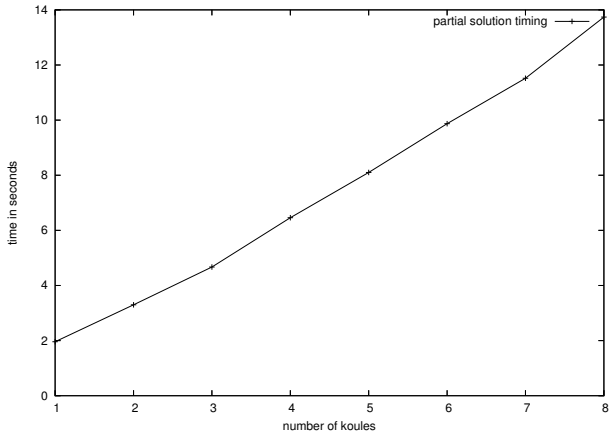

Figure 5.6 : Average time spent per 1000 iterations versus number of koules

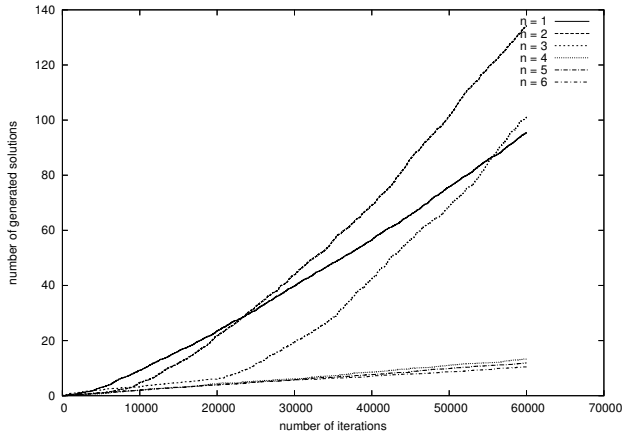

Figure 5.7 : Average number of solutions generated versus number of iterations

partition stab operations and the binary heap make nearly no impact on the scale of a few hundred thousand iterations. The growth in the cost of iterations is shown in Figure 5.6. The super-linear trend is due to the increased number of inter-robot collisions.

An important question that must be asked about Algorithm 1 is: how well does our algorithm PDST−EXPLORE perform as coverage estimates become coarser due to the dimensionality increase? One way to examine this is to look at the number of solutions a run of PDST−EXPLORE generates as a function of the number of iterations. When the space becomes well covered then the rate solutions are generated frequently. Before good coverage is achieved, the solution rate will be much less. In Figure 5.7, we show the average solution count for partial solutions with $n = 1, ..., 6$ koules. The sharp drop-off that occurs when moving from $n = 3$ to $n = 4$ suggests the coverage estimator begins to fail when moving from 9 to 11-dimensional space.

5.2.9 Full Solutions

Algorithm 3 is used for generating full solutions for instances of the game of Koules by repeatedly invoking PDST−EXPLORE. For each trial, we generated a random problem in-

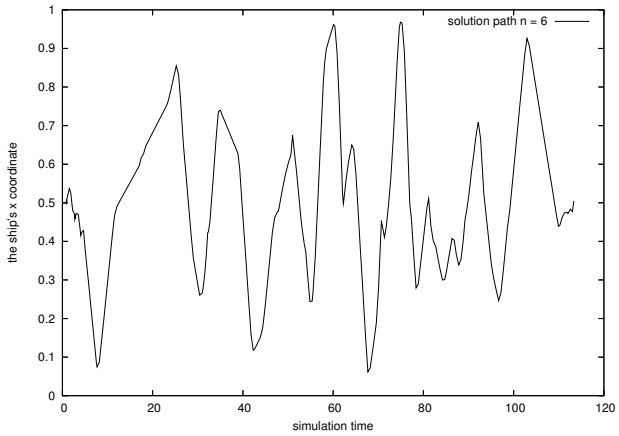

Figure 5.8 : The trajectory taken the ship's x-coordinate during a full solution of a problem with 6 koules

stance and then ran Algorithm 3. In our tests, $N_{\text{attempts}} = 1$ and $N_{\text{iter}} = 40000$ were used.

The computed paths were quite complicated, with durations of several hundred thousand simulator steps and thousands of maneuvers. In Figure 5.8 we see an example of a computed solution for an instance with 6 Koules. The figure shows the path by the ship's x-coordinate. Qualitatively, the paths tended to look quite good. The random trajectory generation did tend to produce occasional path sections where the ship coasted away from the koules, however the usual mode was that the ship would separate a koule from the pack and systematically bounce it into the wall using three or four hits, while avoiding the walls and the other koules.

In Figure 5.9, we present the time used by the planner to solve instances of various complexity. The number of backtracks in Algorithm 3 grew at slightly higher rate than linear with the number of koules. This is due to `PDST-EXPLORE` failing to find solutions more frequently as n increases. The amount of time used grows fairly quickly with the number of koules. This is expected to be worse than quadratic since the number of invocations of Algorithm 1 grows linearly and the cost per iteration is super-linear in the number

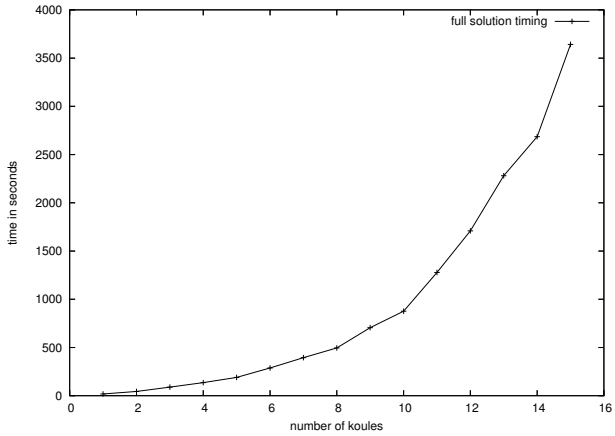

Figure 5.9 : Timing results for full solutions averaged over 90 trials

of koules. Experiments with up to 20 koules were conducted and solutions were produced in less than 3 hours. The runtime began to grow very quickly around $n = 20$ because of memory usage. When $n = 15$, the state space is 65 dimensional and when $n = 20$, the state space is 87 dimensional.

5.2.10 Additional Experiments

At the end of Subsection 5.2.4 we discussed the motivation behind the design of Algorithm 2. The direction of the target velocity vector is set using the procedure on lines 4 and 6. We replaced this procedure with choosing the direction of the target velocity uniformly and randomly. We then ran full solutions trial with 3, 6 and 9 koules and observed a severe performance degradation. Sample bias in trajectory generation and the kinds of paths being generated are extremely important to determine the performance of `PDST-EXPLORE`. Biased trajectory generation helps the planner reduce the time spent searching.

The difficulty of the game of Koules can be varied by modifying the physical parameters. The most important parameters for varying difficulty are relative masses of the koules and the ship, the ship's thrust, $a^{\max}$ and the spring constant for the koules λ_c. To our sur-

prise, reducing the value $v_\theta^{\max}$ by a factor less than 4 did not seem to affect the solution times which is interesting as human players seem to be extremely sensitive to this parameter.

Chapter 6

Experiments with Simulated Physics

This chapter presents additional experimental results using PDST−EXPLORE. The focus is on problems with more complicated dynamics, where a physical simulator is required to properly model the system. First, an abstract interface between the planner and the simulator will be provided. The two specific examples described in this chapter are planning for a weight-lifting robot and a differential drive car. These problems include friction, gravity and take into account rigid body dynamics.

6.1 Open Dynamics Engine

Problems with sophisticated dynamics require a physically-realistic simulator to model actuation. The Open Dynamics Engine (ODE) [Smi06] is an open source, high performance library for simulating rigid body dynamics that has been used in this work. It includes modeling of advanced joint types and integrated collision detection with friction. Section 2.5.1 provides more details on the specific techniques implemented by ODE.

ODE is used for simulating articulated rigid body structures. An articulated structure is created when rigid bodies of various shapes are connected together with joints of various kinds. For example, in a ground vehicle the wheels are connected to the chassis. ODE is designed to be used in interactive or real-time simulation and is appropriate for modeling moving objects in changeable environments. Beyond a stable integration method [ST96] and a proper modeling of hard contacts for non-penetrating rigid bodies [Bar92], ODE also

contains a built-in collision detection system.

6.1.1 Interface with ODE

There are two main usages of ODE in the planning phase: collision detection and propagation of the system's dynamics. To use this functionality, the following initialization steps are performed:

1. Create world with rigid bodies.
2. Attach joints to the bodies.
3. Define a space for collisions.

During the execution of the PDST planner, instead of internally propagating the system's state given control input, the ODE can be used to model the evolution of the system. After sampling a set of candidate controls the following simulation loop is executed:

1. Apply controls through forces and torques to the bodies as necessary.
2. Adjust joint parameters.
3. Call collision detection.
4. Produce a contact joint for every collision point to treat collisions.
5. Propagate dynamics.

The most important concepts in the above simulation loop are presented in the following paragraphs.

Rigid Bodies

The world object is a container for rigid bodies and joints. Objects in different worlds can not interact. In the examples of this thesis, a single world was used. A rigid body has various properties as part of its state. The first four may potentially change as the simulation progresses:

- Position vector of the body's point of reference, which corresponds to the body's center of mass.
- Linear velocity of the point of reference, a vector (v_x, v_y, v_z).
- Orientation of a body, represented by a quaternion (q_s, q_x, q_y, q_z).
- Angular velocity vector $(\omega_x, \omega_y, \omega_z)$ which describes how the orientation changes over time.
- Mass of the body m, which remains constant throughout the simulation. The center of mass coincides with the point of reference.
- Inertia matrix, that describes how the body's mass is distributed around the center of mass. In the current implementation, all bodies are assumed to be homogeneous.

Conceptually each body has an (x, y, z) coordinate frame embedded in it, that moves and rotates with the body, as shown in Figure 6.1. The origin of this coordinate frame is the body's point of reference. Note that the shape of a rigid body is not a dynamical property and is not part of its state. It is only collision detection that cares about the detailed shape of the body.

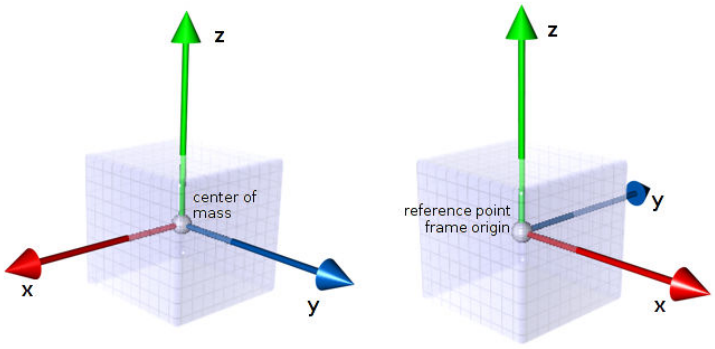

Figure 6.1 : An example of coordinate frame defined for a cube shaped rigid body

Joints and Constraints

Bodies are connected to each other with joints. Joints define the relationship enforced between two bodies so that they can only have certain positions and orientations relative to each other. An "island" of bodies is a group that can not be pulled apart - in other words each body is connected somehow to every other body in the island. Each island in the world is treated separately when the simulation step progresses. For example, the wheels and the chassis of a car define an island. The joint relationship is also called a constraint since a joint typically has limits and represents conditions that cannot be violated.

Joints can have different types in ODE but for the purpose of this thesis all joints are of the "hinge" type. A hinge joint constraints the two parts of the hinge to be in the same location and to line up along the hinge axle as Figure 6.2 shows. Each joint has a number of parameters controlling its geometry. The parameters of a hinge joint are the anchor point where the two bodies connect and the axis around which they rotate as in Figure 6.2. The functions to set joint parameters all take global coordinates, not body-relative coordinates. A consequence of this is that the rigid bodies that a joint connects must be positioned correctly before the joint is attached. The specific frame and joint coordinates used in the

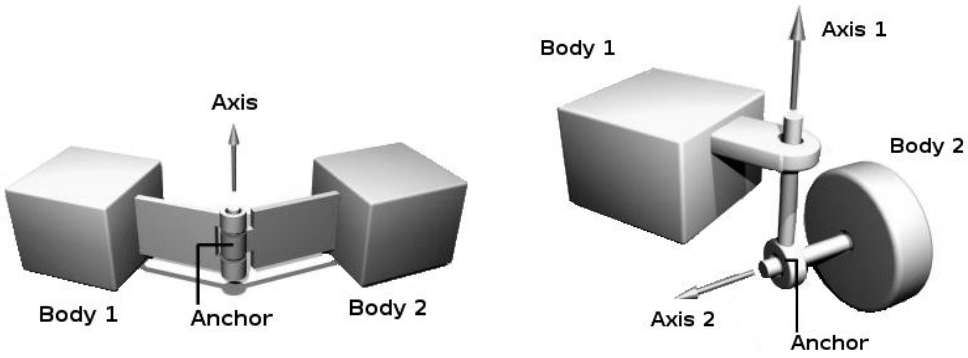

Figure 6.2 : The “hinge” and “hinge-2” joint type used in the experiments

weight lifting and the simulated car will be explicitly provided.

Note that two hinges connected in series, with different hinge axes define an extended type of hinge joint, called “hinge-2” joint shown in Figure 6.2. This type of joint is used in the car model, since a “hinge-2” joint can easily model the steering wheel of a car, where one axis allows the wheel to be steered and the other axis allows the wheel to rotate. The “hinge-2” joint has an anchor point and two hinge axes. Typically, axis 1 is specified relative to body 1, such as the chassis in the case of the car, and has joint limits and a motor. Axis 2 is specified relative to body 2, the wheel in the case of the car, and can only have a motor. Axis 1 can function as a suspension axis, i.e. the constraint can be compressible along that axis.

Integration and Force Accumulation

The process of simulating the rigid body system through time is called integration. Each integration step advances the current time by a given step size, adjusting the state of all the rigid bodies for the new time value.

Each time the integrator takes a step all the joints are allowed to apply constraint forces to the bodies they affect. These forces are calculated such that the bodies move in such a

way to preserve all the joint relationships.

The forces from joints together with external forces are added to "force accumulators" in the rigid body object. When the next integration step happens, the sum of all the applied forces will be used to push the body around. The forces accumulators are set to zero after each integration step.

Collision Handling

Before each simulation steps, collision detection functions from `ODE` are called to determine which bodies touch one another. These functions return a list of contact points. Each contact point specifies a position in space, a surface normal vector and a penetration depth. A special contact joint is created for each contact point. The contact joint is given extra information about the contact, for example the friction present at the contact surface or how bouncy or soft it is. Only after the inclusion of the additional contact joints, a simulation step is propagated.

Geometry objects are the fundamental objects in the collision system. A geometry can represent a single rigid shape (such as a sphere or box), or it can represent a group of other geometries. Any geometry can be collided against any other geometry to yield zero or more contact points. Geometries can be placeable or non-placeable. A placeable geometry has a position vector and a 3 by 3 rotation matrix, just like a rigid body, that is changed during the simulation. Non-placeable geometries, such as static environmental features, do not have this capability. To use the collision engine in a rigid body simulation, placeable geometries are associated with rigid body objects. This allows the collision engine to get the position and orientation of the geometries from the bodies. For example, a box-shaped geometry is positioned at the frame origin of the corresponding rigid body, rotated according to the frame and its shape is defined by the lengths (L^x, L^y, L^z).

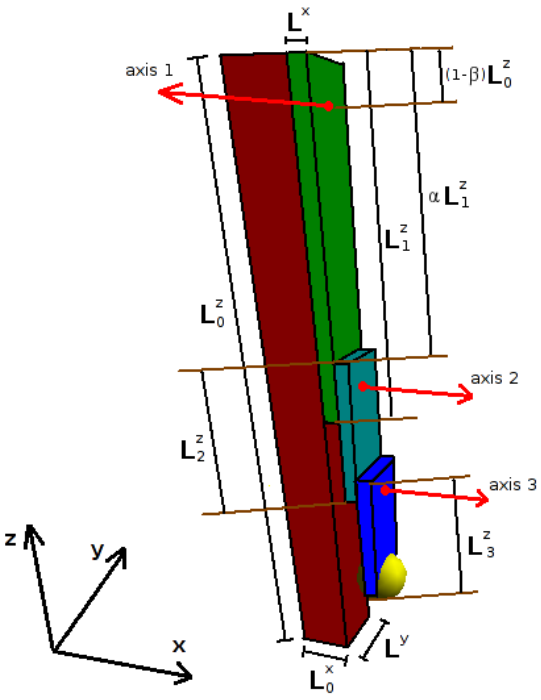

Figure 6.3 : The weightlifting robot

6.2 Weightlifting 3R Planar Chain

The first example corresponds to a manipulator, which is composed of three box-shaped bodies linked through hinge-type joints to form a planar chain as Figure 6.3 shows. The planar chain is mounted on a static pole of height L_0^z and thickness L_0^x. The three bodies have the same thickness L^x, width L^y but different lengths L_0^z, L_1^z and L_2^z. There are motors at the joints that allow the end effector to move on a planar surface parallel to the $(y - z)$ plane. The end effector achieves the minimum z coordinate without any forces being applied due to the modeling of gravity.

A heavy mass is attached to the end effector. The goal for the manipulator is to lift this mass at the maximum z value. Because the mass is heavy this cannot be achieved easily (there is no kinematics paths) and the manipulator must rotate in order to gain momentum

before being able to lift the mass. This goal defines a planning problem in the control space of the manipulator. The torques at the motors of the three joints τ_1, τ_2, τ_3 and the angular velocities of the bodies must be selected so as to compute a path that will eventually result in a solution. Figures 6.8, 6.9 and 6.10 present various solutions to the same problem.

Description of Physical Parameters

The three rigid-body frames attached to the geometries are shown in Table 6.1. The table gives the position of each frame in world coordinates.

frame	x	y	z
1	$0.5L_0^x + 0.5L^x$	0	$L_0^z - 0.5L_1^z$
2	$0.5L_0^x + 1.5L^x$	0	$L_0^z - 0.5L_2^z - \alpha L_1^z$
3	$0.5L_0^x + 2.5L^x$	0	$L_0^z - 0.5L_3^z - \alpha(L_1^z + L_2^z)$

Table 6.1 : Rigid body frames for weightlifting 3R planar chain

The weightlifting robot has four separate parts which are described in Table 6.2 consisting of three rectangular prisms which form the body of the robot and a spherical weight anchored at the end of the third link. Positions and rotations are given in the coordinate frame relative to the rigid body that the part is attached to and masses are uniformly distributed.

The weightlifting robot is a 3R planar chain. The positions in world coordinates and axes of rotation are given in Table 6.3.

The model for the weightlifting robot is parametric to variation in the difficulty of the problem. The constants used in the experiments presented in this thesis are given in Table 6.4.

id	type	frame	position	rotation	mass
1	box (L^x, L^y, L^z_1)	1	$(0,0,0)$	$\vec{0}$	m_1
2	box (L^x, L^y, L^z_2)	2	$(0,0,0)$	$\vec{0}$	m_2
3	box (L^x, L^y, L^z_3)	3	$(0,0,0)$	$\vec{0}$	m_3
4	sphere with radius r	3	$(0,0,-0.5\beta L^z_3)$	$\vec{0}$	m_w

Table 6.2 : Geometry for weightlifting 3R planar chain

id	type	frames	axis	anchor
1	R	$(0,1)$	$(-1,0,0)$	$(0.5L^x_0, 0, \beta L^z_0)$
2	R	$(1,2)$	$(1,0,0)$	$(0.5L^x_0 + L^x, 0, L^z_0 - \beta L^z_1)$
3	R	$(2,3)$	$(1,0,0)$	$(0.5L^x_0 + L^x, 0, L^z_0 - \alpha L^z_1 - \beta L^z_2)$

Table 6.3 : Joints for weightlifting 3R planar chain

Controller

The control space for weightlifting robot is

$$U = \prod_{i=1}^{3} [-\omega_i^{\max}, \omega_i^{\max}].$$

Random controls for the weightlifting robot are chosen uniformly distributed. A given control $(\omega_1, \omega_2, \omega_3) \in U$ is interpreted as a target velocity. The component ω_i is therefore the velocity for the i^{th} link. The velocity of the links are controlled in the simulation by three independent linear feedback controllers applying bounded torques to the joints. The absolute values torques are bounded by the constants $\tau_i^{\max}$ from $i = 1, 2, 3$. The constants used in this implementation are given in Table 6.5.

In the implementation of the controller makes use of the internal `ODE`linear feedback

L^x	L^y	L_0^z	L_1^z	L_2^z	L_3^z	α	β	r	m_1	m_2	m_3	m_w
0.05	0.2	2.0	1.0	0.5	0.5	0.8	$0.5 + 0.5\alpha$	0.1	0.1	0.05	0.05	8.0

Table 6.4 : Constants for weightlifting 3R planar chain

$\tau_1^{\max}$	$\tau_2^{\max}$	$\tau_3^{\max}$	$\omega_1^{\max}$	$\omega_2^{\max}$	$\omega_3^{\max}$
40.0	35.0	30.0	8.0	6.0	4.0

Table 6.5 : Constants for weightlifting 3R planar chain controller

controller in the following fashion: at the beginning of each timestep, the target velocity for each joint is set to the currently operating control and the torque bounds are set. During the integration of the system for the timestep, `ODE` approximates a continuous linear feedback controller.

6.3 Simulated Car

The second example corresponds to a simulated car, which is composed of a body chassis and four wheels. The back wheels of the cars are linked to the chassis through hinge joints, while the front steering wheels use hinge-2 joints. Figure 6.4 provides an illustration of the simulated car. In terms of dynamics, the body is modeled as a box with dimensions L^x, L^y, L^z. For the initial placement of the car, the origin is considered to be at the center of the body's box. Then the wheels are placed at the four corners of the body with distance $0.5\alpha_x L^x$ and $0.5\alpha_y L^y$ from the origin along the x and y axis correspondingly. The wheels have a radius r and the body frame is positioned higher than the wheel's axis of rotation. There are motors at the joints that allow the wheels to rotate around the x axis, while the front wheels are also able to rotate around the z axis.

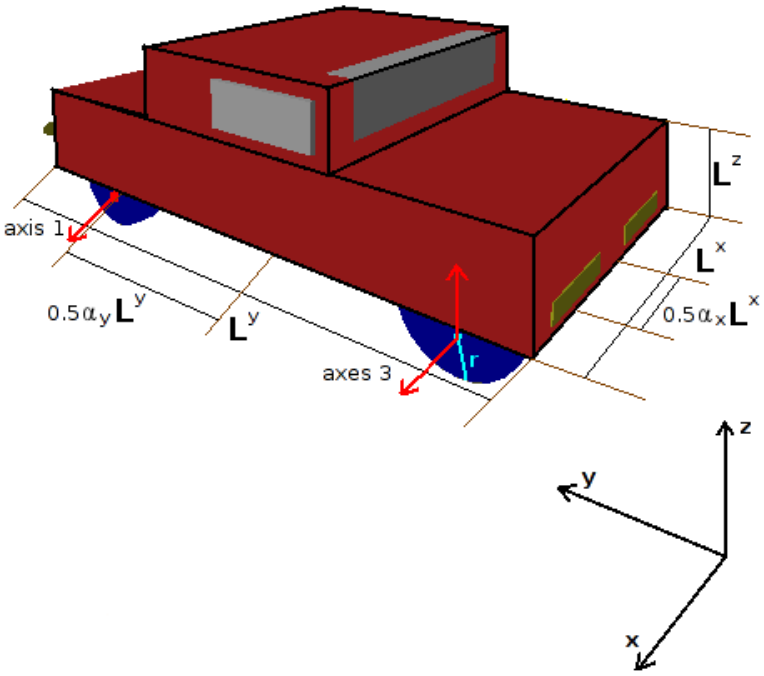

Figure 6.4 : Simulated car

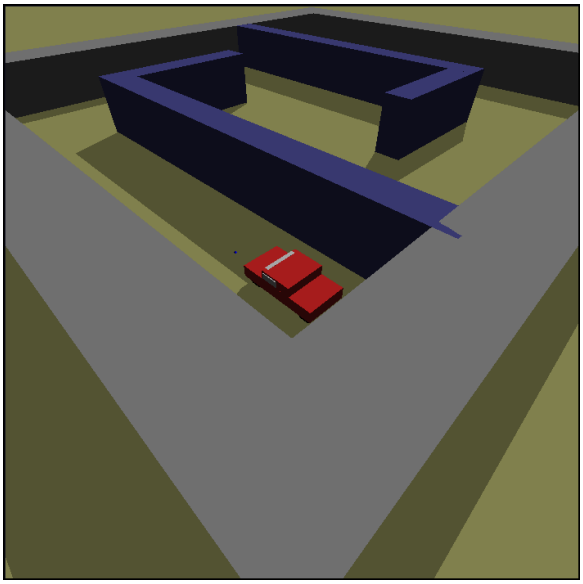

Figure 6.5 : The easy maze environment

Planning problems with this simulated car correspond to a selection of appropriate torque and angular velocity values for the motors at the joints connecting the four wheels. Figure 6.5 provides an example of the type of environments tested with this model. The car must start at the bottom right corner of a maze like environment and reach the top left corner. Figures 6.11, 6.12 and 6.13 present sequences of images from experiments in similar environments. The first figure has the additional difficulty that the car has to go up a ramp and take an abrupt drop before reaching the target. The last two figures require the robot to push through a set of movable obstacles.

Description of Physical Parameters

The five rigid-body frames attached to the body and the four wheels are shown in Table 6.6. The table gives the position of each frame in world coordinates. Note that the wheels are appropriately rotated.

frame	x	y	z	rotation
1	0	0	$0.5L^z + r$	$\vec{0}$
2	$-0.5\alpha_x L^x$	$-0.5\alpha_y L^y$	r	$(0, 1, 0, \pi/2)$
3	$0.5\alpha_x L^x$	$-0.5\alpha_y L^y$	r	$(0, 1, 0, \pi/2)$
4	$-0.5\alpha_x L^x$	$0.5\alpha_y L^y$	r	$(0, 1, 0, \pi/2)$
5	$0.5\alpha_x L^x$	$0.5\alpha_y L^y$	r	$(0, 1, 0, \pi/2)$

Table 6.6 : Rigid body frames for simulated car

As mentioned earlier, the main body of the car is modeled as a box and has a mass m_b. The wheels are cylinders and they have the same mass m_w. Table 6.7 describes the geometries for the parts of the simulated car.

id	type	frame	position	rotation	mass
1	box (L^x, L^y, L^z)	1	$(0,0,0)$	$\vec{0}$	m_b
2	cylinder with radius r and height h	2	$(0,0,0)$	$\vec{0}$	m_w
3	cylinder with radius r and height h	3	$(0,0,0)$	$\vec{0}$	m_w
4	cylinder with radius r and height h	4	$(0,0,0)$	$\vec{0}$	m_w
5	cylinder with radius r and height h	5	$(0,0,0)$	$\vec{0}$	m_w

Table 6.7 : Geometry for simulated car

There are four joints in the simulated car, one for each wheel. Table 6.8 describes how these joints link the various rigid bodies, the anchor point and the axes of rotation. The first two joints that correspond to the two back wheels have only one axis, since they are simple hinge joints. As described in Section 6.1.1, the front steering wheels can make use of hinge-2 joints and they have two axis of rotation.

id	type	frames	axis 1	axis 2	anchor
1	R	$(1,2)$	$(1,0,0)$	n/a	$(0,0,0)$
2	R	$(1,3)$	$(1,0,0)$	n/a	$(0,0,0)$
3	2R	$(1,4)$	$(1,0,0)$	$(0,0,1)$	$(-0.5\alpha_x L^x, 0.5\alpha_y L^y, r)$
4	2R	$(1,5)$	$(1,0,0)$	$(0,0,1)$	$(0.5\alpha_x L^x, 0.5\alpha_y L^y, r)$

Table 6.8 : Joints for simulated car

The constants used in the experiments presented in this thesis for the simulated car are given in Table 6.9. The additional parameters k_{erp} and k_{cfm} are related to ODE's internal parameters for handling errors in modeling of "soft" constraints.

L^x	L^y	L^z	r	h	α_x	α_y	m_b	m_w	k_{erp}	k_{cfm}
0.5	1.0	0.15	0.1	0.04	0.75	0.75	5.0	0.4	1.0	0.0

Table 6.9 : Constants for simulated car

Most constraints are by nature "hard". This means that the constraints represent conditions that are never violated. In practice constraints can be violated by unintentional introduction of errors into the system, but an error reduction process can be used for limiting the effect of these errors. For example, during each simulation step each joint applies a special force to bring its bodies back into correct alignment. Some "soft" constraints, however, are designed to be violated. For example, the contact constraint that prevents colliding objects from penetrating is hard by default, so it acts as though the colliding surfaces are made of steel. But it can be made into a soft constraint to simulate softer materials, thereby allowing some natural penetration of the two objects when they are forced together.

Parameters k_{erp} and k_{cfm} control the distinction between hard and soft constraints. The first is the error reduction parameter that specifies what proportion of the joint error will be fixed during the next simulation step. If $k_{\text{erp}} = 0$ then no correcting force is applied and the bodies will eventually drift apart as the simulation proceeds. If $k_{\text{erp}} = 1$ then the simulation will attempt to fix all joint errors during the next time step. The second is the constraint force mixing value k_{cfm}. If $k_{\text{cfm}} = 0$, the constraint will be hard. If k_{cfm} is set to a positive value, it will be possible to violate the constraints. In this thesis the constraints are modeled as hard and the error reduction parameter is used to its full extent.

6.4 Cumulative Probability of Solution Charts

Let T be the random variable that represents the time taken by a planner to produce a solution. For a given time $t \geq 0$, the function

$$\phi(t) = Prob(T \leq t)$$

can be defined to give the probability that the planner solves the query in less than time t. This function has an inverse, which will be denoted T_p, for $0 \leq p \leq 1$,

$$T_p = t \text{ such that } \phi(t) = p.$$

Observe that $E(T) = T_{0.5}$ by definition. Consider the simple process

1. Run the planner until a solution is produced or until time t_{input}.
2. If a solution was produced then return it, otherwise repeat.

The notation C_t will denote the random variable expressing the running time of planner with the maximum running time clamped at t and C_t^* will denote the random variable expressing the running time of the process described above with $t = t_{\text{input}}$.

Some simple calculations show

$$E(C_t) = \phi(t)T_{\frac{\phi(t)}{2}} + (1 - \phi(t))t$$

and

$$E(C_t^*) = \phi(t)E(C_t) + (1 - \phi(t))(t + E(C_t^*)) = E(C_t) + \frac{(1 - \phi(t))}{\phi(t)}t.$$

Figure 6.6 provides the cumulative probability $E(C_t)$ for the weightlifter robot and Figure 6.7 corresponds to the simulated car.

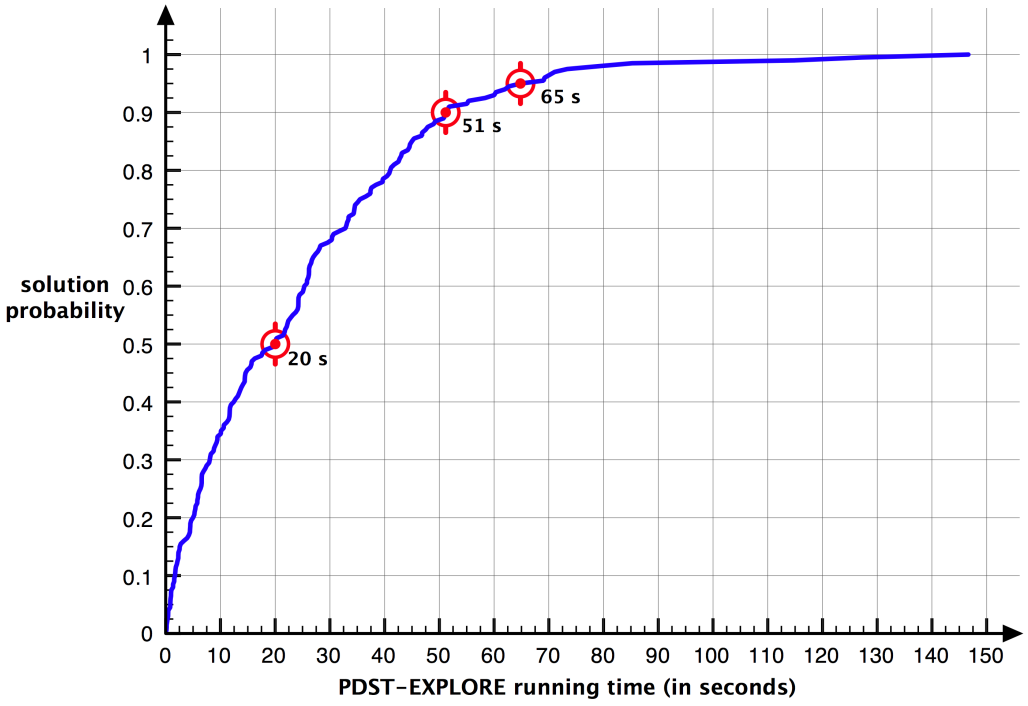

Figure 6.6 : Weightlifter cumulative probability of solution (200 trials)

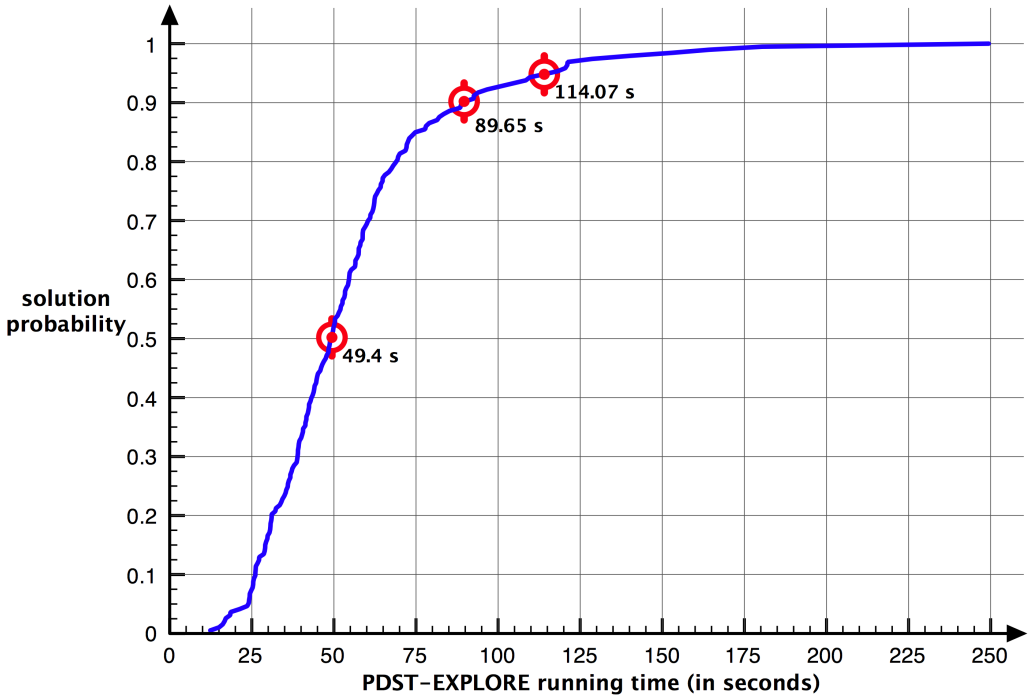

Figure 6.7 : Simulated car in the easy maze environment probability of solution (200 trials)

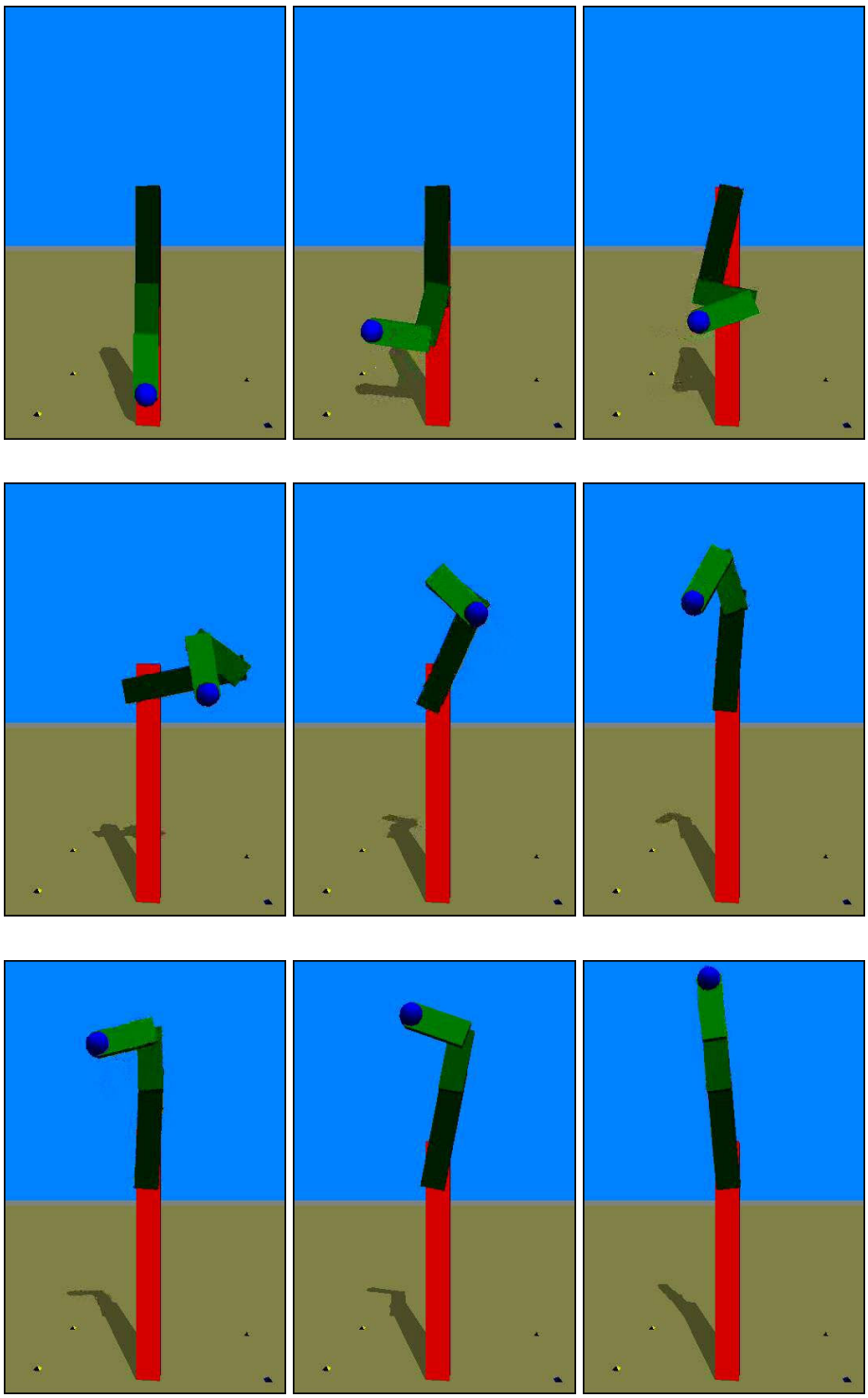

Figure 6.8 : Weightlifter: first example

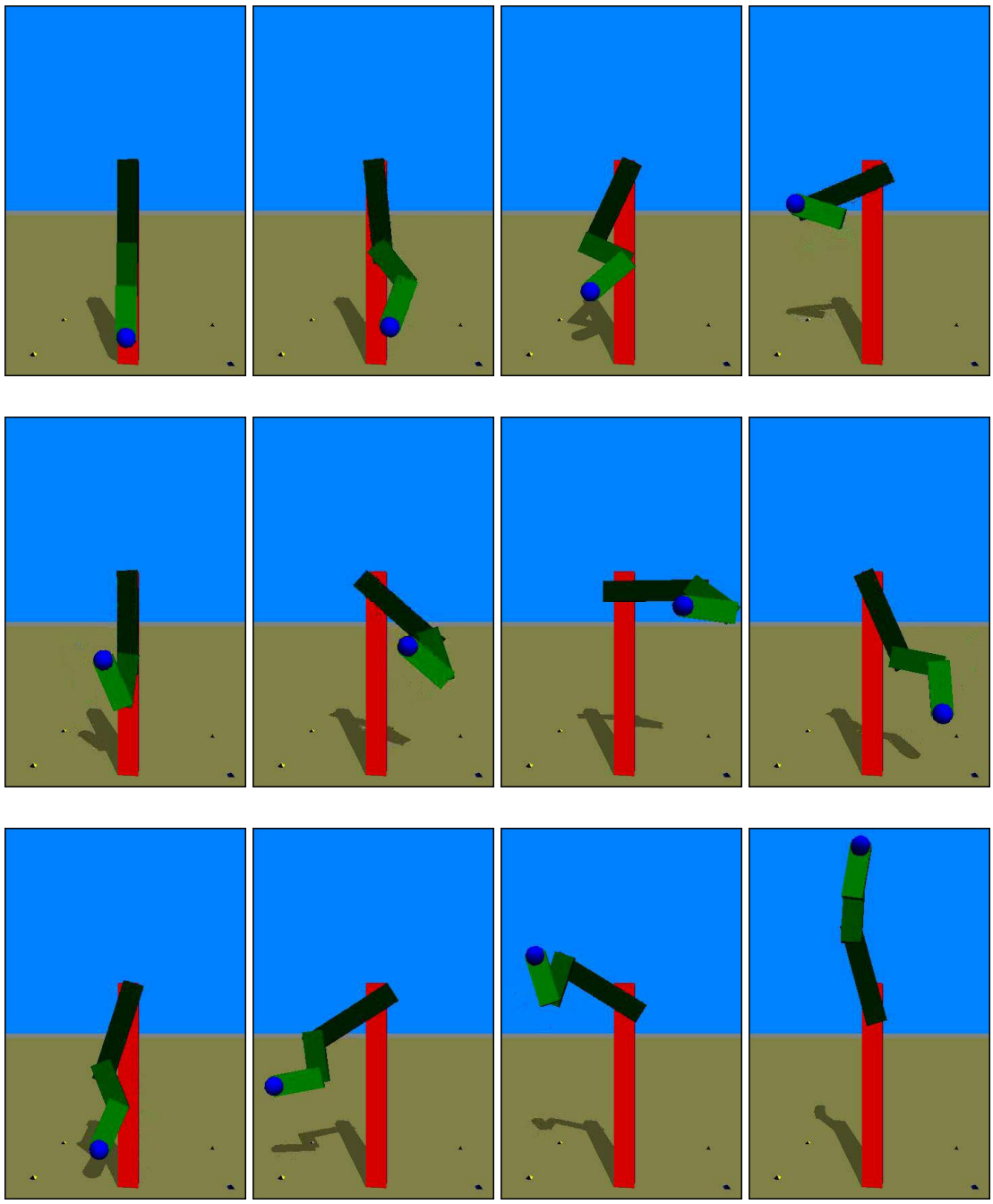

Figure 6.9 : Weightlifter: second example

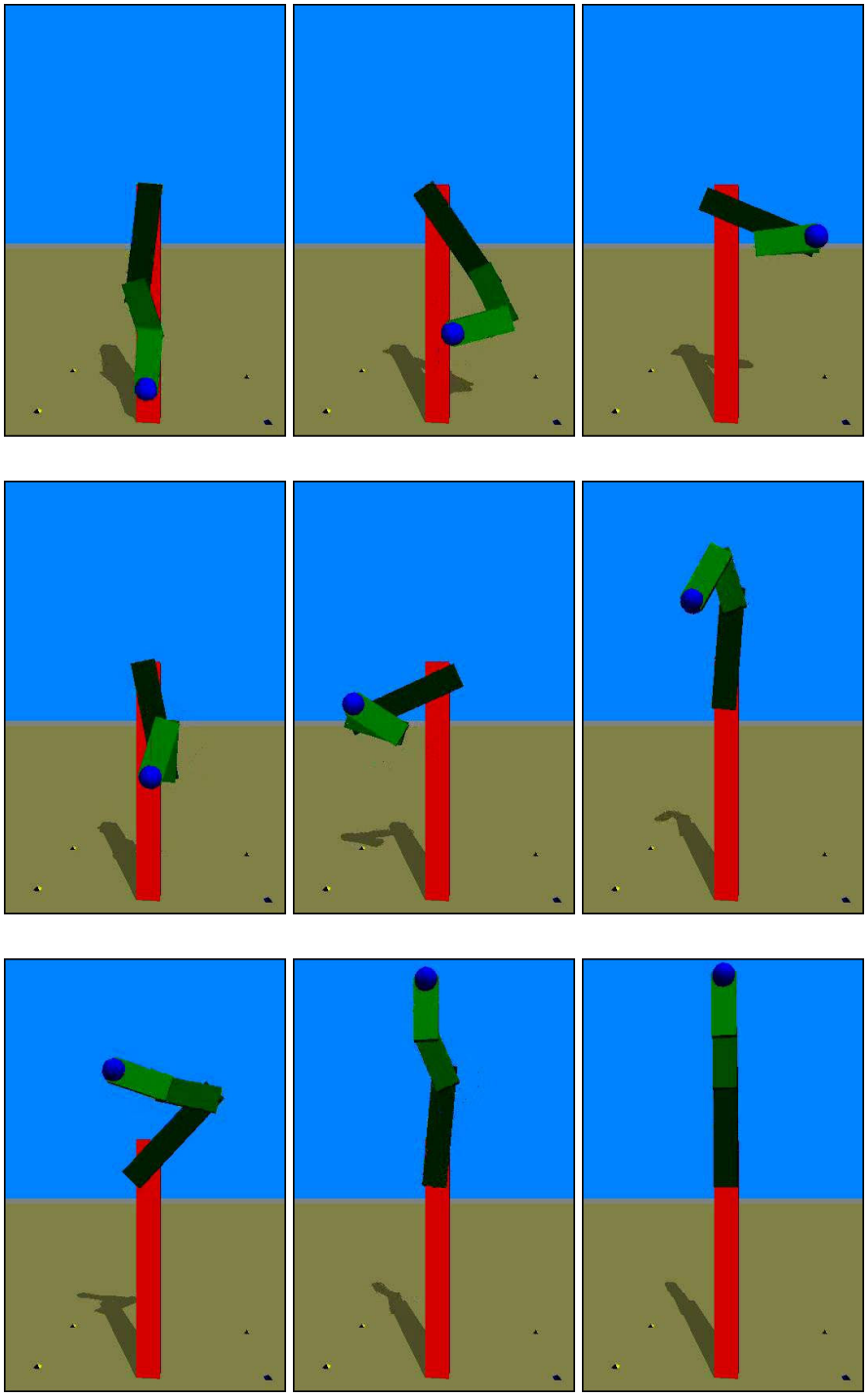

Figure 6.10 : Weightlifter: third example

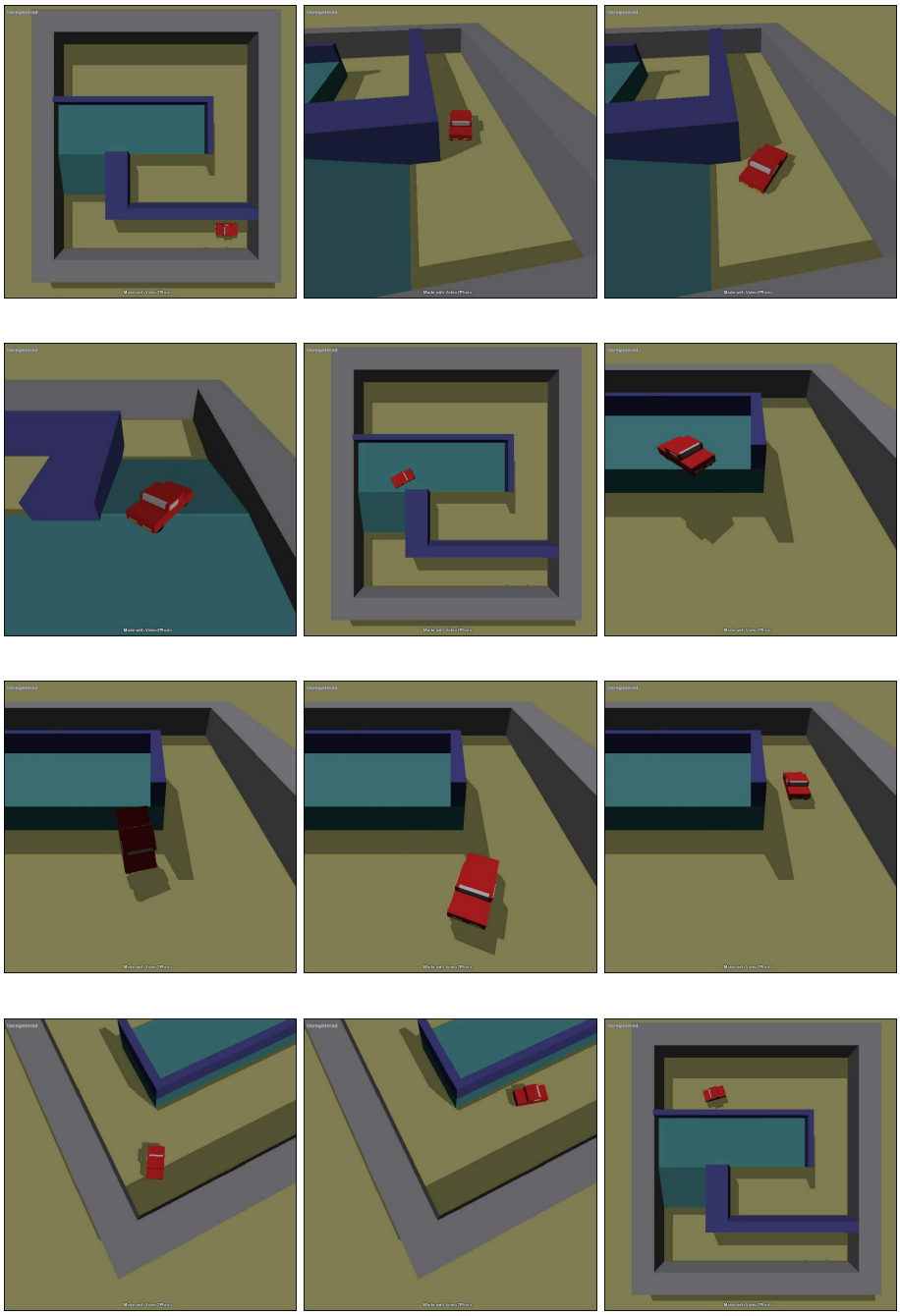

Figure 6.11 : Simulated car in the fancy ramp

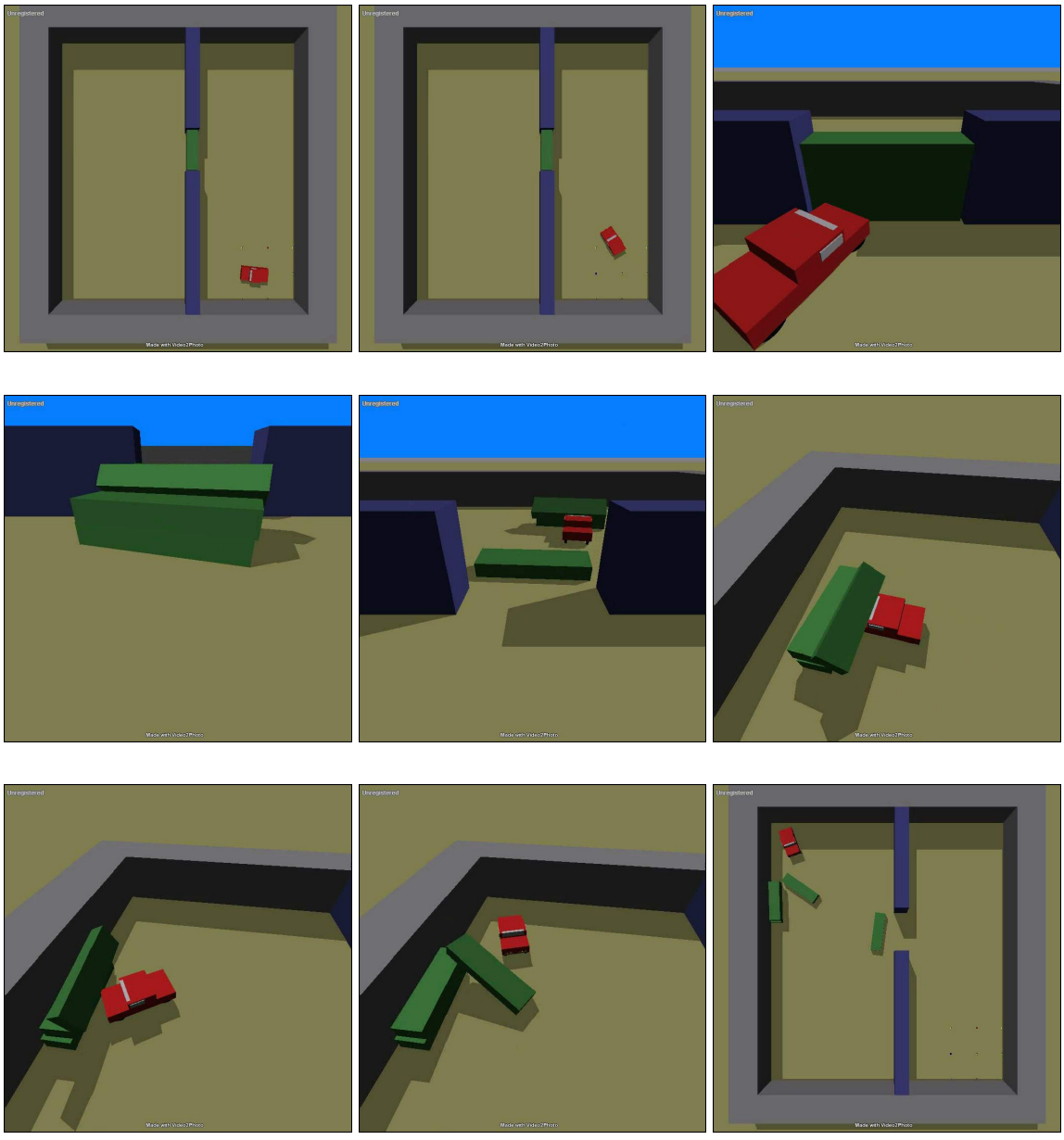

Figure 6.12 : Simulated car with a fancy barrier: first example

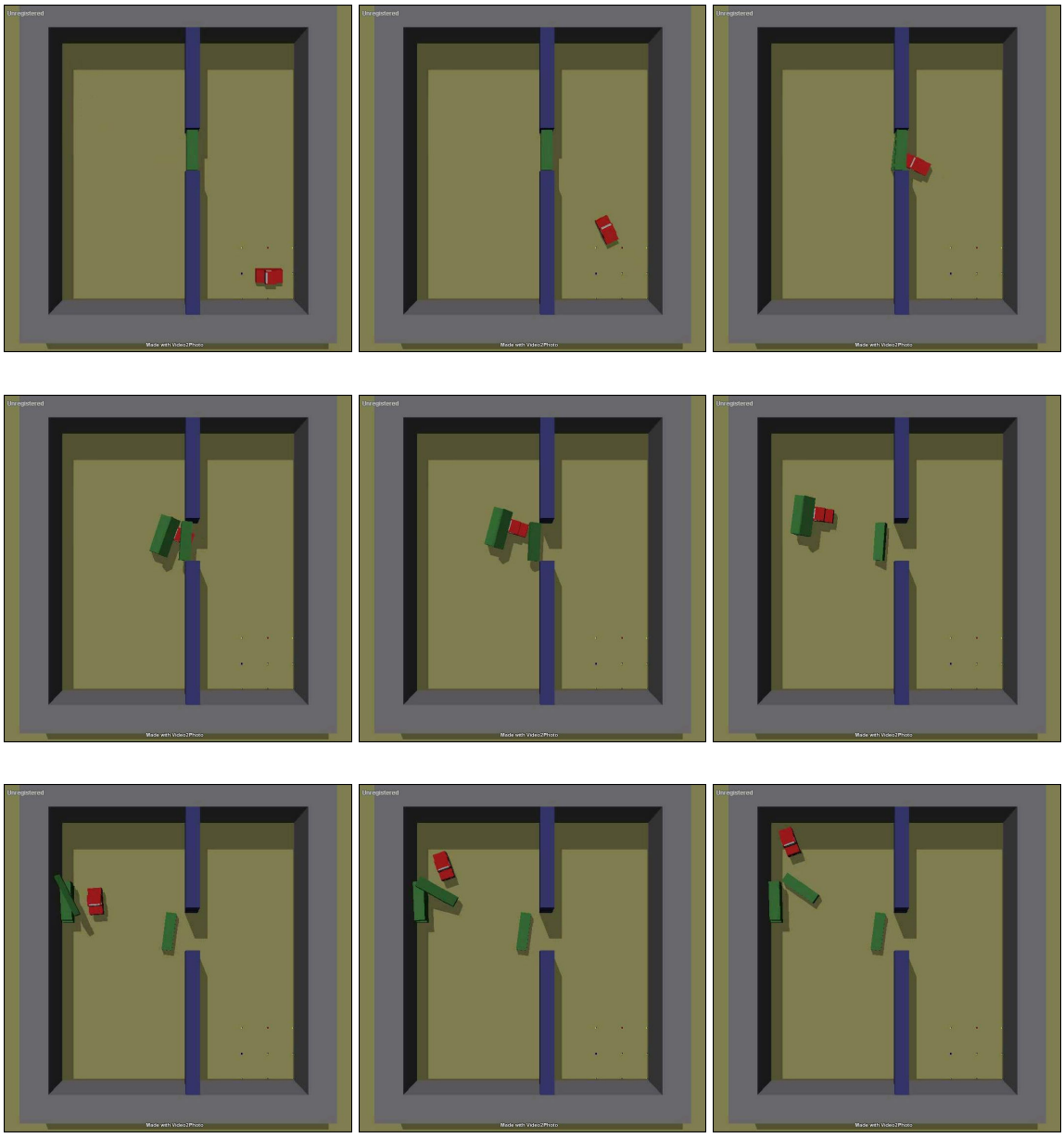

Figure 6.13 : Simulated car with a fancy barrier: second example

Chapter 7

Proof that PDST-EXPLORE is Probabilistically Complete

In this chapter, a proof of probabilistic completeness will be presented for the algorithm presented in this thesis. Throughout, a specific MPP is assumed together with a probability measure for the control space μ_C and a specific subdivision operator subdivide. Several concepts will be given names and notation in order to express the proof. Most of these devices are only necessary for explaining the main result.

7.1 Random Walk Criteria

Let $(\mathcal{Q}, \mathcal{U}, F, q_0, G)$ be a MPP. A *primitive path segment* is defined by three quantities $q \in \mathcal{Q}$, $u \in \mathcal{U}$ and $T \geq 0$. A primitive path is *feasible* if for all $t \in [0, T)$ *,

$$F(q, u, t) \neq \perp.$$

The space of all feasible primitive path segments is $\mathcal{P} \subset \mathcal{Q} \times \mathcal{U} \times \mathbb{R}^{\geq 0}$.

A primitive path $\pi = (q, u, T) \in \mathcal{P}$ can be thought of as defining a function from $[0, T)$ to $\mathcal{Q}$. When $T = 0$, then the domain of the function is $[0]$. This function is given by the rule

$$\pi(t) = F(q, u, t).$$

A primitive path $\pi = (q, u, T) \in \mathcal{P}$ can be also be thought of as defining a subset of $\mathcal{Q}$.

*an interval of the form $[a, b)$ is interpreted as $[a]$ when $a = b$

This subset is written

$$\text{Im}(\pi) = \{q \in \mathcal{Q} : \text{there exists } t \in [0, T) \text{ such that } \pi(t) = q\}.$$

An random operator called $\texttt{propagate} : \mathcal{P} \to \mathcal{P}$ will now be defined. For any two real numbers a and b, the random operator $\texttt{uniform}(a, b)$ picks a number $x \in [a, b)$ uniformly and at random. Additionally, the random operator $\texttt{randomControl}()$ chooses $u \in \mathcal{U}$ at random according to some distribution. The probability measure $\mu_{\mathcal{U}}$ represents that distribution in that for any measurable subset $U \subset \mathcal{U}$,

$$\mu_{\mathcal{U}}(U) = \text{Prob}(\texttt{randomControl}() \in U).$$

The *propagation operator* is now defined in terms of these two operators and a strictly positive real number T_{max}

$$\texttt{propagate}\left((q, u, T)\right) = (q', u', T'),$$

where

$$q' = F(q, u, \texttt{uniform}(0, T))$$

$$u' = \texttt{randomControl}()$$

$$T' = \begin{cases} T_{\text{max}} & \text{if } F(q', u', T_{\text{max}}) \neq \perp \\ T_{\text{fail}} & \text{where } T_{\text{fail}} = \inf\{t \geq 0 : F(q', u', t) = \perp\} \end{cases}$$

The propagation operator can be written in an iterated form

$$\texttt{propagate}^i(\pi) = \begin{cases} \texttt{propagate}(\pi) & i = 0 \\ \texttt{propagate}(\texttt{propagate}^{i-1}(\pi)) & \text{otherwise} \end{cases}$$

The next definition defines a useful criteria. There is a slight abuse of notation in defining the intersection of a path sample and a subset of the state space

$$(q, u, D) \cap G = (q, u, D')$$

where

$$D' = \{t \in D : F(q, u, t) \in G\}.$$

Property 7.1.1. *A* MPP, $(\mathcal{Q}, \mathcal{U}, F, q_0, G)$ *and propagation operator* $\mu_{\mathcal{U}}, T_{max}, \epsilon$ *satisfies the* **Random Walk Criteria**(RWC) *if there exists* $N > 0$ *such that*

$$\textit{Prob}\left(\texttt{propagate}^N((q_0, \cdot, 0)) \cap G \neq \emptyset\right) > 0$$

or the MPP *is not solvable*[†].

7.2 The Propagation Operator

In order to give a formal treatment of the propagate operator and its random sampling properties, it is convenient to examine a space which will be called the *propagation space* and is defined as

$$\mathcal{J} = \bigcup_{n=1}^{\infty} \left(\prod_{i=1}^{n} ([0, 1) \times \mathcal{U}) \right).$$

The propagation space, $\mathcal{J}$, has the natural measure structure inferred by $[0, 1)$ and $\mathcal{U}$. It is equipped with probability measure which is determined by its behavior on the rectangles, for any $n \geq 1$

$$\mu_{\mathcal{J}}\left(([a_1, b_1) \times U_1) \times \cdots \times ([a_n, b_n) \times U_n)\right) = \frac{1}{2^n} \left(\prod_{i=1}^{n} (b_i - a_i) \mu_{\mathcal{U}}(U_i) \right),$$

where for all $1 \leq i \leq n$, $0 \leq a_i \leq b_i \leq 1$ and $U_i \subset \mathcal{U}$.

The importance of the propagation space is that it provides an avenue to analyze sampling behavior of the propagate operator. This connection is made via the *propagation*

[†]the symbol $\cdot$ is used in place of a value when the variable it represents is "unbound" either because all values are equivalent or to denote functional currying

projection function. The definition of the propagation projection function, $h : \mathcal{P} \times \mathcal{J} \to \mathcal{P}$, is given recursively by

$$h((q,u,T),j_n) = \begin{cases} (q',u_n,T') & n = 1 \\ h(q',u_n,T'),j_{n-1}) & \text{otherwise} \end{cases}$$

where

$$q' = F(q,u,t_n \cdot T),$$

$$j_n = (t_1,u_1,...,t_n,u_n) \in \mathcal{J},$$

for $n > 1$,

$$j_{n-1} = (t_1,u_1,...,t_{n-1},u_{n-1}) \in \mathcal{J},$$

and, as in the definition of propagate,

$$T' = \begin{cases} T_{\text{max}} & \text{if } F(q',u_n,T_{\text{max}}) \neq \perp \\ T_{\text{fail}} & \text{where } T_{\text{fail}} = \inf\{t \geq 0 : F(q',u_n,t) = \perp\} \end{cases}$$

The ultimate object of interest in the above development is the projection back to the state space which will be denoted $\phi : \mathcal{Q} \times [0,1) \times \mathcal{P} \to \mathcal{Q}$. This projection ϕ is defined by

$$\phi(q,t,j) = F(q',u',T' \cdot t) \text{ where } (q',u',T') = h((q,\cdot,0),j)\ .$$

This projection is used as a stepping stone to write a version of its pre-image

$$\Phi(q,A) = \{j \in \mathcal{P} : \text{there exists } t \in [0,1) \text{ such that } \phi(q,t,j) \in A\}\ .$$

The pre-image function maps measurable sets in $\mathcal{Q}$ to measurable sets $\mathcal{P}$ as a consequence of the measurability of the transit function F.

Using the pre-image function Φ, a probability measure for the state space which captures the behavior of `propagate` can be constructed simply

$$\mu_{hq_0}(A) = \mu_{\mathcal{J}}\left(\Phi(q_0,A)\right).$$

Lemma 7.2.1. *If the* MPP *satisfies the* RWC *then*

$$\mu_{hq_0}(G) > 0.$$

Proof. If the MPP satisfies the RWC then by Property 7.1.1, there exists $N > 0$ such that

$$\text{Prob}\left(\texttt{propagate}^N((q_0, \cdot, 0)) \cap G \neq \emptyset\right) > 0.$$

A call to Nth iterated propagate operator uses N calls to the uniform operator and the randomControl operator. The results of the calls to uniform will be written $t_1, ..., t_N$ and the calls to randomControl will be written $u_1, ..., u_N$. Without loss of generality, the values of t_i can be considered to be in $[0, 1)$ since calls to $\texttt{uniform}(0, T)$ can be rewritten $\texttt{uniform}(0, 1) \cdot T$.

Note that $j = (t_1, u_1, ..., t_N, u_N)$ is a point in $\mathcal{J}$. If the N calls to uniform are $t_1, ..., t_N$ and the N calls to randomControl return $u_1, ..., u_N$ then by the definition of h and propagate,

$$\texttt{propagate}^N((q_0, \cdot, 0)) = h((q_0, \cdot, 0), j).$$

Let the set J be the set of solutions of length N, i.e.

$$J = \left\{ j \in \prod_{i=1}^{N} ([0, 1) \times \mathcal{U}) : h((q_0, \cdot, 0), j) \cap G \neq \emptyset \right\}.$$

The following equivalence is evident:

$$\text{Prob}(\texttt{propagate}^N((q_0, \cdot, 0)) \cap G \neq \emptyset) = 2^N \mu_{\mathcal{J}}(J).$$

Since the RWC holds,

$$\mu_{\mathcal{J}}(J) = \frac{\text{Prob}(\texttt{propagate}^N((q_0, \cdot, 0)) \cap G \neq \emptyset)}{2^N} > 0.$$

Furthermore, it is clear that $J \subset \Phi(q_0, G)$. Therefore it can be concluded that

$$\mu_{hq_0}(G) = \mu_{\mathcal{J}}(\Phi(q_0, G)) > \mu_{\mathcal{J}}(J) > 0.$$

□

Lemma 7.2.2. *If the* MPP *satisfies the* RWC *then there exists a subset*

$$R = \prod_{i=1}^{k} (a_i, b_i) \times U_i \subset \mathcal{J}$$

such that for all $1 \leq i \leq k$, $0 \leq a_i < b_i \leq 1$ *and* $\mu_{\mathcal{U}}(U_i) > 0$.

Proof. Recall the definition of J from the proof of Lemma 7.2.1. It contained the solutions of length N. It was previously established that $\mu_{\mathcal{J}}(J) > 0$. J can be rewritten as

$$J = \prod_{i=1}^{N} A_i \times U_i,$$

where $A_i \subset [0, 1)$ and $U_i \subset \mathcal{U}$. Since primitive intervals of the form (a, b) generate the Borel algebra for the probability space $[0, 1)$, J can be rewritten as

$$J = \bigcup_{j=1}^{\infty} \prod_{i=1}^{N} (a_{i,j}, b_{i,j}) \times U_i,$$

where $0 \leq a_{i,j} < b_{i,j} \leq 1$ for all i, j. By definition

$$\mu_{\mathcal{J}}(J) = \frac{1}{2^N} \sum_{j=1}^{\infty} \prod_{i=1}^{N} (b_{i,j} - a_{i,j}) \cdot \mu_{\mathcal{U}}(U_i),$$

Since $\mu_{\mathcal{J}}(J) > 0$, it follows that there exists j such that

$$\prod_{i=1}^{N} (b_{i,j} - a_{i,j}) \cdot \mu_{\mathcal{U}}(U_i) > 0.$$

which proves the Lemma (for $k = N$). □

7.3 PDST−EXPLORE Produces a Dense Sample

During the operation of PDST−EXPLORE, the data structures $\mathbf{S}$ and $\mathbf{C}$ are built and updated. To refer to these structures at each iteration of the algorithm, the notation $\mathbf{S}_i$ and $\mathbf{C}_i$ can be used. In this way, $\mathbf{S}_i$ refers to the sample set at the end of the ith iteration. Moreover, written this way, $\mathbf{S}_0 = \{(q_0, \cdot, 0)\}$ and $\mathbf{C}_0 = \{C_0\}$.

By ignoring the termination conditions on lines 7,8 and 9 of PDST−EXPLORE and by replacing line 4 with an infinite loop PDST−EXPLORE can be thought of as producing infinite runs of the form

$$(\mathbf{S}_0, \mathbf{C}_0), ..., (\mathbf{S}_i, \mathbf{C}_i), ...$$

Other important variables for the state of PDST−EXPLORE are π^* and π' for the ith iteration, so for a given infinite run, these are denoted π_i^* and π_i'. If $\pi \in \mathbf{S}$ then $\texttt{priority}_i(\pi)$ is the priority of π at the end of the ith iteration. Similarly, $\texttt{score}_i(\pi)$ is defined the same way.

In this way of writing things, the question of completeness asks if there exists i such that $\pi_i' \cap G \neq \emptyset$.

This discussion will begin by proving some handy Lemmas about score. Here the sequence α_i is defined as the minimum score of the elements of $\mathbf{S}_i$ at the end of the ith iteration of a run of PDST−EXPLORE

$$\alpha_i = \min_{\pi \in \mathbf{S}_i} \texttt{score}_i(\pi).$$

Lemma 7.3.1. *Suppose* PDST−EXPLORE *generates the infinite run*

$$(\mathbf{S}_0, \mathbf{C}_0), ..., (\mathbf{S}_i, \mathbf{C}_i), ...$$

For any iteration N, if $\alpha_N \geq \xi$ and $N > 2/xi$ hold then there exists an iteration $M > N$ such that

$$\alpha_M > 2\xi.$$

Proof. This proof proceeds by counting the number of samples with priorities between ξ and 2ξ. The set V_i contains all of the samples with priorities in the desired range at the end of the ith iteration,

$$V_i = \{\pi \in \mathbf{S}_i : \xi \leq \texttt{score}_i(\pi) \leq 2\xi\}.$$

The cardinality of this set will be written ρ_i,

$$\rho_i = |V_i'|.$$

The next goal in this proof is show that for every $i \geq N, \alpha_i \geq \xi$ and that ρ_i is strictly monotone decreasing until converging to 0, where it remains. The argumentation will be carried out on the $i+1$th iteration as an induction using $i = N$ as the base case.

1. For any $\gamma \in \mathbf{S}_i$, $\gamma' \in \mathbf{S}_{i+1}$ such that γ' is a subsample of γ,

   $$\mu_C(\texttt{cell}_i(\gamma')) \geq \mu_C(\texttt{cell}_{i+1}(\gamma)).$$

   This follows from the definition of the subdivision scheme. Observing that $\texttt{cell}_{i+1}(\gamma') \subset \texttt{cell}_i(\gamma')$ completes the result.

   A stronger claim can be proved. Since $\gamma \in \mathbf{S}_i$, it follows that

   $$\texttt{priority}_{i+1}(\gamma') \geq \texttt{priority}_i(\gamma)$$

   and combining the above observation shows

   $$\texttt{score}_{i+1}(\gamma') \geq \texttt{score}_i(\gamma).$$

2. If $\gamma \in V_{i+1}$ then γ is not a subsample of π'_{i+1}.

   The operation of PDST−EXPLORE implies any subsample $\gamma \in \mathbf{S}_{i+1}$ of π'_{i+1} has priority $i+1$ which by assumption is greater than $2\xi > N$. Therefore,

   $$\texttt{score}_{i+1}(\gamma) = \frac{\texttt{priority}_{i+1}(\gamma)}{\mu_C(\texttt{cell}_{i+1}(\gamma))} \geq \texttt{priority}_{i+1}(\gamma) \geq 2\xi.$$

3. $\alpha_{i+1} \geq \xi$.

   Let $\gamma \in \mathbf{S}_{i+1}$ be such that $\texttt{score}_{i+1}(\gamma) = \alpha_{i+1}$. Either γ is a subsample of $\gamma' \in \mathbf{S}_i$ or γ is a subsample of π'_{i+1}. In the former case, it was previously demonstrated that $\texttt{score}_{i+1}(\gamma) \geq \texttt{score}_i(\gamma') \geq \xi$. In the latter case, $\texttt{score}_{i+1}(\gamma) \geq 2\xi$.

4. If $\gamma \in V_{i+1}$ then γ is not a subsample of π^*_{i+1}.

   By induction $\alpha_i \geq \xi$ and therefore $\texttt{score}_i(\pi^*_{i+1}) > \xi$ since $\texttt{score}_i(pi^*_{i+1}) = \alpha_i$. On line 11 of PDST−EXPLORE, the priority of the selected element is penalized. It follows that

   $$2\xi < 2\texttt{score}_i(\pi^*_{i+1}) + \epsilon \leq \frac{2\texttt{priority}_i(\pi^*_{i+1}) + 1}{\texttt{cell}_i(\pi^*_{i+1})} \leq \frac{\texttt{priority}_i(\gamma)}{\texttt{cell}_{i+1}(\gamma)} \leq \texttt{score}_{i+1}(\gamma),$$

   for sufficiently small $\epsilon > 0$.

5. If $\gamma \in \mathbf{S}_i$ then there is at most one subsample $\gamma' \in \mathbf{S}_{i+1}$ such that $\gamma' \in V_{i+1}$.

   Suppose there is more than one subsample of γ in V_{i+1}. The only way this is possible is if the cell C subdivided by PDST−SUBDIVIDE is $\texttt{cell}(\gamma) = C$ and the children of C, C_L and C_R both intersect γ to create subsamples γ_L and γ_R after subdivision. Since both $\texttt{score}_{i+1}(\gamma_L), \texttt{score}_{i+1}(\gamma_R) \geq \texttt{score}_i(\gamma)$, it follows that if either are in V_{i+1} then $\gamma \in V_i$. Noting that by the subdivision definition

   $$\mu_C(C_L) + \mu_C(C_R) = \mu_C(C),$$

   without loss of generality $\mu_C(C_L) \leq 0.5\mu_C(C)$ and therefore

   $$\texttt{score}_{i+1}(\gamma_L) \geq \frac{\texttt{priority}_i(\gamma)}{\mu_C(C_L)} \geq 2\frac{\texttt{priority}_i(\gamma)}{\mu_C(C)} \geq 2\texttt{score}_i(\gamma) \geq 2\xi.$$

   It follows that both γ_L and γ_R cannot be members of V_{i+1}.

6. If $\rho_i > 0$ then $\rho_{i+1} < \rho_i$.

   First consider what is known about elements in V_{i+1}. Every element $\gamma \in V_{i+1}$ is a subsample of an element $\gamma' \in V_i$ since subsamples of π'_{i+1} cannot appear in V_{i+1}. Furthermore, it cannot be that there are two distinct elements $\gamma_1, \gamma_2 \in V_{i+1}$ such that they are both subsamples of the same $\gamma' \in V_i$. It follows that there is an injective

map $f : V_{i+1} \to V_i$ such that $f(\gamma) = \gamma'$ where γ is a subsample of γ'. Therefore $\rho_{i+1} \leq \rho_i$. Recall that $\pi^*_{i+1} \in V_i$ but there is no $\gamma \in V_{i+1}$ such that γ is subsample of π^*_{i+1}. It follows that $\rho_{i+1} < \rho_i$.

□

Lemma 7.3.2. *Suppose* `PDST-EXPLORE` *generates the infinite run*

$$(\mathbf{S}_0, \mathbf{C}_0), ..., (\mathbf{S}_i, \mathbf{C}_i), ...$$

Then the minimum `score` *tends to infinity as the number of iterations increases:*

$$\lim_{i \to \infty} \alpha_i = \infty.$$

Proof. It is trivially true that $\alpha_i \geq 1$ for all $i > 0$. So for iteration $N = 3$, $\alpha_N \geq 1$. Since $N/2 > 1$, it follows by Lemma 7.3.1 that there exists $M > N$ such that for all $i \geq M$, $\alpha_i > 2$. Suppose now inductively that for $k > 0$, there is iteration N such that for all $i > N$,

$$\alpha_i \geq 2^k.$$

Without loss of generality $N > 2^{k+1}$ and by applying Lemma 7.3.1, there is $M \geq N$ such that for all $i > M$

$$\alpha_i > 2^{k+1}.$$

Therefore by induction, for any constant c, there is N such that for all $i > N$, $\alpha_i > c$. It follows that

$$\lim_{i \to \infty} \alpha_i = \infty.$$

□

Lemma 7.3.3. *Suppose* `PDST-EXPLORE` *generates the infinite run*

$$(\mathbf{S}_0, \mathbf{C}_0), ..., (\mathbf{S}_i, \mathbf{C}_i), ...$$

For any iteration N, if $C \in \mathbf{C}_N$ and there is a sample $\pi \in \mathbf{S}_N$ such that $\texttt{cell}_N(\pi) = C$ then there exists $M > N$ such that $C \notin \mathbf{C}_M$.

Proof. Suppose for some iteration N, there is a cell $C \in \mathbf{C}_N$ such that there is a sample $\pi \in \mathbf{S}_N$ such that $\texttt{cell}_N(\pi) = C$ but for all $i \geq N$, $C \in \mathbf{C}_i$. It follows that for any $i \geq N$ and $\gamma \in \mathbf{S}_i$ such that $\texttt{cell}_i(\gamma) = C$ that $\pi^*_{i+1} \neq \gamma$, i.e. γ is never selected since C is never subdivided. It follows that for all $i > N$, that $\pi \in \mathbf{S}_i$ and $\texttt{priority}_i(\pi) = \texttt{priority}_N(\pi)$. Therefore it follows that for all $i > N$

$$\alpha_i = \min_{\gamma \in \mathbf{S}_i} \texttt{score}_i(\gamma) \leq \texttt{score}_i(\pi).$$

This is in direct contradiction with Lemma 7.3.2 and the desired result follows. □

Lemma 7.3.4. *Suppose* PDST−EXPLORE *generates the infinite run*

$$(\mathbf{S}_0, \mathbf{C}_0), ..., (\mathbf{S}_i, \mathbf{C}_i), ...$$

For any $N \geq 0$ iterations, for any sequence of samples π_i for $i \geq N$ such that $\pi_i \in S_i$ and π_{i+1} is a subsample of π_i, there exists $M \geq N$ such that $\pi^*_{M+1} = \pi_M$.

Proof. This will be a proof by contradiction. Suppose the Lemma's statement does not hold then there is $N \geq 0$ and a sequence of samples π_i for $i \geq N$ but for all $M \geq N$, π^*_{M+1} is not π_M. Such a sequence has the property that for all $i \geq N$, $\texttt{cell}_{i+1}(\pi_{i+1}) \subset \texttt{cell}_i(\pi_i)$. Changes in the sequence π_i in the sense of "$\pi_{i+1} \neq \pi_i$" occur as a result of the subdivision of the cell $\texttt{cell}_i(\pi_i)$. The initial assumption eliminates the possibility that π_i was selected ($\pi_i \neq \pi^*_{i+1}$) so subdivision of the cell $\texttt{cell}_i(\pi_i)$ occurs when some other sample in the cell is selected. Let $\xi = \texttt{priority}_N(\pi_N)$ and note that $\texttt{priority}_i(\pi_i) = \xi$ for all $i \geq N$ since π_i is never selected. Finally, define for $i \geq N$ the sequence

$$B_i = \{\pi \in \mathbf{S}_i : \texttt{cell}_i(\pi) = \texttt{cell}_i(\pi_i) \text{ and } \texttt{priority}_i(\pi) \leq \xi\}.$$

Following a similar line of reasoning as the proof of Lemma 7.3.1, a counting argument can be constructed. Now consider the quantity

$$\rho_i = \sum_{\pi \in B_i} 1 + \xi - \texttt{priority}_i(\pi).$$

By definition, $\rho_i \geq 0$ for $i \geq N$. Without loss of generality, N can be taken to be strictly greater than ξ. Since $N > \xi$, there can be no insertions of samples that have priority less than ξ and it follows that ρ_i is monotone decreasing. Consider what happens on a hypothetical iteration $K \geq N$ when

$$\texttt{cell}_K(\pi^*_{K+1}) = \texttt{cell}_i(\pi_i).$$

The priority of π^*_{K+1} is least of all samples $\pi \in \mathbf{S}_K$ such that $\texttt{cell}_K(\pi) = \texttt{cell}_K(\pi^*_{K+1})$. Since $\pi_K \in B_K$ and B_K contains all samples with priority equal or less than ξ, it follows that $\pi^*_{K+1} \in B_K$. Additionally, following the primary assumption, π^*_{K+1} is a not π_K. Since the priorities of π^*_{K+1} is increased on lines 10-12 of PDST−EXPLORE, it follows that $\rho_{K+1} < \rho_K$.

Suppose there exists $K \geq N$ such that for all $j \geq K$, $\rho_j = \rho_K$ and $\rho_K > 0$. It has been shown on an iteration j when a sample from the cell $\texttt{cell}_i(\pi_i)$ is selected that $\rho_{j+1} < \rho_j$. Therefore, if for all $j \geq K$, $\rho_j = \rho_K$ then no sample from $\texttt{cell}_j(\pi_j)$ is ever selected meaning that $\texttt{cell}_j(\pi_j) = \texttt{cell}_K(\pi_K)$ for all $j \geq K$. This violates the statement of Lemma 7.3.3 and therefore cannot be true. Hence, it follows that for every $i \geq N$ such that $\rho_i > 0$, there exists $j > i$ such that $\rho_j < \rho_i$.

Eventually there exists $K \geq N$ such that $\rho_K = 0$. By definition, for all $j \geq K$, B_K contains only π_K. Since the claim that for all $j \geq K$, $\texttt{cell}_j(\pi^*_{j+1}) \neq \texttt{cell}_j(\pi_j)$ violates Lemma 7.3.3, it must be true that eventually a sample from cell $\texttt{cell}_j(\pi_j)$ is selected. However, since B_j contains only π_j, there can be no sample with in that cell with priority

less than or equal to that of π_j. There is no other possibility than to conclude that π_j is selected which completes the contradiction and proves the Lemma. □

Lemma 7.3.5. *Suppose* `PDST-EXPLORE` *generates the infinite run*

$$(\mathbf{S}_0, \mathbf{C}_0), ..., (\mathbf{S}_i, \mathbf{C}_i), ...$$

and after the Nth iteration, the sample $\pi \in \mathbf{S}_N$ has been generated. Let $0 \leq a < b \leq 1$ and let $U \subset \mathcal{U}$ such that $\mu_{\mathcal{U}}(U) > 0$. Then there exists with probability 1 iteration $M > N$ such that $\pi' \in \mathbf{S}_M$ such that there exists $t \in (a, b)$, $u \in U$ and

$$\pi' = h(\pi, (t, u)).$$

Proof. In order to prove the desired result, it will be sufficient to show the existence of a sequence $\pi_i \in \mathbf{S}_i$ for $i \geq N$ such that π_{i+1} is a subsample of π_i, $\pi_N = \pi$ and if π_i is selected during the $i+1$th iteration then

$$\text{Prob}\left(\texttt{propagate}(\pi_i) = h(\pi, (t, u))\right) > p$$

for any $t \in (a, b)$, $u \in U$ and for some constant $p > 0$ independent of i. If such a sequence existed, then by Lemma 7.3.4, there would be infinitely many i such that π_i was selected and each time would have probability greater than p of successfully producing a sample with the desired property. In this way, it is shown that `PDST-EXPLORE` produces a sample $\pi' = h(\pi, (t, u))$ in finite time with probability 1.

The construction proceeds inductively. Let $\pi_N = \pi$ and fix the constant p as

$$p = \text{Prob}(\texttt{propagate}(\pi) = h(\pi, (t, u))) = (b - a)\mu_{\mathcal{U}}(U) > 0.$$

Inductively, there are two possibilities: either the $\texttt{cell}_i(\pi_i)$ is subdivided or not. If not, then $\pi_{i+1} = \pi_i$ and the probability of propagate producing the desired sample remains the same.

After subdivision, two new cells are created and either one or two subsamples of π_i are created. If there is only one created then $\pi_{i+1} = \pi_i$ and nothing changes. So consider the case where π_i is split into γ_L and γ_R such that

$$\pi_i = (q, u, D)$$

$$\gamma_L = (q, u, D_L)$$

$$\gamma_R = (q, u, D_R).$$

Observe that inductively

$$\text{Prob}(\texttt{propagate}(\pi_i) = h(\pi, (t, u))) \geq p.$$

This means that

$$\frac{\mu((a, b) \cap D)}{\mu(D)} \mu_{\mathcal{U}}(U) \geq p.$$

Now consider the following equations

$$\text{Prob}(\texttt{propagate}(\gamma_L) = h(\pi, (t, u))) = \frac{\mu((a, b) \cap D_L)}{\mu(D_L)} \mu_{\mathcal{U}}(U) = \frac{p_L}{T_L} \mu_{\mathcal{U}}(U),$$

$$\text{Prob}(\texttt{propagate}(\gamma_R) = h(\pi, (t, u))) = \frac{\mu((a, b) \cap D_R)}{\mu(D_R)} \mu_{\mathcal{U}}(U) = \frac{p_R}{T_R} \mu_{\mathcal{U}}(U),$$

for constants p_L, p_R, T_L, T_R. Note that D_L and D_R partition D, so it follows that

$$\frac{p_L + p_R}{T_L + T_R} \mu_{\mathcal{U}}(U) = \text{Prob}(\texttt{propagate}(\pi_i) = h(\pi, (t, u))).$$

Suppose $p_L / T_L \mu_{\mathcal{U}}(U) < p$, then it can be deduced that

$$\frac{p_L}{T_L} \mu_{\mathcal{U}}(U) < p \leq \frac{p_L + p_R}{T_L + T_R} \mu_{\mathcal{U}}(U).$$

$$(T_L + T_R) p_L < T_L (p_L + p_R).$$

$$p_L T_R < p_R T_L.$$

$$\frac{p_L}{T_L} < \frac{p_R}{T_R}.$$

Similarily, if $p_R/T_R\mu_{\mathcal{U}}(U) < p$ it follows that $\frac{p_R}{T_R} < \frac{p_L}{T_L}$. Since both inequalities cannot be true, it follows that either $p_L/T_L\mu_{\mathcal{U}}(U) \geq p$ or $p_R/T_R\mu_{\mathcal{U}}(U) \geq p$. In this way, either γ_L or γ_R can be chosen to be π_{i+1}. Inductively, a sequence π_i with the desired properties can be constructed and the proof is given. □

7.4 Iterations are Finite Time

Property 7.4.1. *This property deals with how* `subdivide` *operator affects to the* `PDST-EXPLORE` *algorithm. For any subdivision* $\mathbf{C}$ *produced by a finite number of calls to* `subdivide`*, the following statements should hold.*

1. *Every* $C \in \mathbf{C}$ *has strictly positive measure:* $\mu_C(C) > 0$.

2. *Every path primitive* $\pi = (q, u, T) \in \mathcal{P}$ *intersects with a finite number of cells in the subdivision* $\mathbf{C}$*, i.e. the function* $\phi : [0, T) \to \mathbf{C}$ *defined by*

$$\phi(t) = C,$$

where $\pi(t) \in C$ *and* $C \in \mathbf{C}$*, is piecewise constant.*

Lemma 7.4.2. *A single iteration of the* `PDST-EXPLORE` *algorithm runs in finite time if Property 7.4.1 holds.*

7.5 The Main Result

Theorem 7.5.1. *If a Motion Planning Problem is solvable and satisfies the Random Walk Criteria and a given distribution over control space* $\mu_{\mathcal{U}}$ *and the subdivision operator satisfies Property 7.4.1, then* `PDST-EXPLORE` *will find a solution to the given* `MPP` *in finite time with probability* 1.

Proof. The RWC implies the result of Lemma 7.2.2 and therefore there exists N and R such that

$$R = \prod_{i=1}^{N} (a_i, b_i) \times U_i \subset \mathcal{J}$$

such that for all $1 \leq i \leq N$, $0 \leq a_i < b_i \leq 1$ and $\mu_{\mathcal{U}}(U_i) > 0$. Recall that q_0 is the initial state input to PDST−EXPLORE and $\pi_0 = (q_0, \cdot, 0)$ is the sample added to initial sample set on line 1 of PDST−EXPLORE. For convenience, the sets R_k for $1 \leq k \leq N$ can be defined as

$$R_k = \prod_{i=1}^{k} (a_i, b_i) \times U_i \subset \mathcal{J}.$$

Note that $R_N = R$. A partial solution of length k for $1 \leq k \leq N$ is a subsample π of any sample $h(\pi_0, r_k)$ such that $r_k \in R_k$ and, additionally for $k < N$, there exists t, $u \in \mathcal{U}$, such that $h(\pi, (t, u)) = h(\pi_0, r_{k+1})$ for some $r_{k+1} \in R_{k+1}$. There is a unique partial solution of length 0 and it is π_0.

It is clear that PDST−EXPLORE eventually generates a partial solution of length 0. Inductively, it will be shown that if π_k is partial solution of length $k < N$, PDST−EXPLORE eventually will generate a partial solution of length $k + 1$ with probability 1.

Let $\pi_k \in S_M$ and note that there is $0 \leq a_{k+1} < b_{k+1} \leq 1$ and U_{k+1} with $\mu_{\mathcal{U}}(U_{k+1}) > 0$. Applying Lemma 7.3.5 implies that, with probability 1, there is iteration $M' \geq M$ such that $\pi_{k+1} \in S_{M'}$ and $\pi_{k+1} = h(\pi_k, (t, u))$ for $t \in (a_{k+1}, b_{k+1})$ and $u \in U_{k+1}$. In other words, $\pi_{k+1} = h(\pi_0, r_{k+1})$ for some $r_{k+1} \in R_{k+1}$ meaning that π_{k+1} is a partial solution of length $k + 1$.

A partial solution of length N, $\pi_N = h(\pi_0, r_N)$ for $r_N \in R_N = R \subset \mathcal{J}$ is therefore eventually constructed with probability 1. Furthermore, $\pi_N \cap G \neq \emptyset$ (a solution to the MPP) by construction. By Lemma 7.4.2, each iteration PDST−EXPLORE takes finite time and by the construction the solution is produced in a finite number of iterations, the main result is given. □

Chapter 8

Discussion

This thesis develops a novel method that can handle motion planning for dynamical systems with high-dimensionality, drift, underactuation, discrete system changes and complex physics. The proposed planner, PDST−EXPLORE is based on sampling-based principles and subdivision methods. Importantly, the planner can use a physical simulator as a black box and hence can deal with systems where state transitions can not be described simply by a system of equations. PDST−EXPLORE is the first planner to present a successful framework where core algorithmic work in motion planning is coupled with a physical simulator. It is argued that the impact that physical simulators will have in planning will be comparable to the huge impact that collision checking primitives have had in the development and popularity of sampling-based planners. Examples with simplified and simulated physics were demonstrated. It is the first time that a planner is shown to handle problems of such physical complexity and in such high dimension.

A major issue that arises in planning for dynamical systems is that memory efficiency is a very important concern. The number of iterations that we were able to perform for a solution is memory bounded. Once the number of states we need to represent the tree exceed the size of the core, performance degraded significantly. Since the cost per iteration of PDST−EXPLORE does not grow quickly, storage usage becomes the bottleneck. Using a path sample representation and making these representations as compact as possible was essential to solve the larger examples; however for handling applications with additional state complexity that may result from increases in physical realism, algorithmic changes

may be necessary to further reduce storage requirements during planning. These issues are almost certainly present regardless of the planning algorithm that is employed.

Besides the issue of memory efficiency, other issues that deserve exploration are the accuracy and utility of coverage estimates, the extension to hybrid systems with possibly a large number of discrete states, and large scale implementations. Better connection with physics simulators will broaden the class of problems for which motion planning can provide realistic solution paths. In simulation, PDST−EXPLORE may end up providing interesting extensions to gaming and entertainment. In the real world, the proposed planner can lead to extensive testing of robot systems before these are built and to designing complex gaits for novel mechanisms and particularly reconfigurable robots. PDST−EXPLORE bridges the model gap between traditional motion planning and control to the benefit of both the robotics and the control communities.

Bibliography

[ABC+05] M. Akinc, K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plaku, and L. E. Kavraki. Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps. In P. Dario and R. Chatila, editors, *Robotic Research: The Eleventh International Symposium*, pages 80–89. Springer, STAR 15, 2005.

[ABD+98] Nancy M. Amato, O. Burchan Bayazit, Lucia K. Dale, Christopher Jones, and Daniel Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proceedings of Workshop on Algorithmic Robotics*, pages 155–168, 1998.

[AGM98] Juan-Manuel Ahuactzin, Kamal Gupta, and Emmanuel Mazer. Manipulation planning for redundant robots: A practical approach. *International Journal of Robotics Research*, 17(7):731–747, July 1998.

[APS99] M. Anitescu, F. A. Potra, and D. E. Stewart. Time-stepping for three dimensional rigid body dynamics. *Computer methods in applied mechanics and engineering*, 177(3-4):183–197, 1999.

[AS74] M. Abramowitz and I.A. Stegun, editors. *Handbook of Mathematical Functions*. National Bureau of Standards, Dover, 1974.

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

[AW96] Nancy M. Amato and Yan Wu. A randomized roadmap method for path and manipulation planning. In *Proceedings of IEEE Conference on Robotics and Automation*, volume 1, pages 113–120, 1996.

[Bar89] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *Computer Graphics (Proc. SIGGRAPH)*, volume 23, pages 223–232. ACM, 1989.

[Bar92] D. Baraff. *Dynamic Simulation of nonpenetrating rigid bodies*. PhD thesis, 1992.

[Bar93] D. Baraff. Issues in computing contact forces on non-penetrating rigid bodies. *Algorithmica*, 10:292–352, 1993.

[Bar94] D. Baraff. Fast contact force computation for non-penetrating rigid bodies. In *Proc. SIGGRAPH*, pages 23–34. ACM, 1994.

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

[BB05] B. Burns and O. Brock. Single-query entropy-guided path planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 2124–2129, Barcelona, Spain, 2005.

[BBC+95] J. Buhmann, W. Burgard, A.B. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The mobile robot Rhino. *AI Magazine*, 16(1), 1995.

[BCL+03] K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plaku, and L. E. Kavraki. Multiple query probabilistic roadmap planning using single query planning primitives.

In *2003 IEEE/RJS International Conference on Intelligent Robots and Systems (IROS)*, pages 656–661, Las Vegas, NV, October 2003.

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

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

[BK00] Robert Bohlin and Lydia E. Kavraki. Path planning using lazy PRM. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 521–528, 2000.

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

[BM02] D. J. Balkcom and M. T. Mason. Time optimal trajectories for differential drive vehicles. *Intl. Journal of Robotics Research*, 21(3):199–217, 2002.

[Boh01] Robert Bohlin. Path planning in practice: Lazy evaluation on a multi-resolution grid. In *Proceedings of IEEE/RSJ Conference on Intelligent Robots and Systems*, 2001.

[BOvdS99] Valérie Boor, Mark H. Overmars, and A. Frank van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planner. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1018–1023, 1999.

[BP83] R. Brooks and T. Lozano Perez. A subdivision algorithm in configuration space for findpath with rotation. In *Proccedings of the International Joint Conference on Artificial Intelligence*, pages 799–803, 1983.

[BSB+01] J. Brown, S. Sorkin, C. Bruyns, J.-C. Latombe, K. Montgomery, and M. Stephanides. Real-time simulation of deformable objects: Tools and application. In *The Fourteenth Conference on Computer Animation*, pages 228–258, Seoul, South Korea, 2001.

[Can88] John Canny. *The Complexity of Robot Motion Planning*. PhD thesis, MIT, Cambridge, MA, 1988.

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

[CFL03] P. Cheng, E. Frazzoli, and S. M. LaValle. Exploiting group symmetries to improve precision in kinodynamic and nonholonomic planning. In *Proc. of the Intl. Conf. on Intelligent Robots and Systems*, volume 1, pages 631–636, 2003.

[CL02] P. Cheng and S. M. LaValle. Resolution complete rapidly-exploring random trees. In *Proc. of the IEEE Intl. Conference on Robotics and Automation*, volume 1, pages 267–272, 2002.

[CL03] Peng Cheng and Steve M. LaValle. Exploiting group symmetries to improve precision in kinodynamic and nonholonomic planning. In *Proceedings of IEEE/RSJ Conference on Intelligent Robots and Systems*, 2003.

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

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

[dBvKO97] M. de Berg, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 1997.

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

[DW91] T. L. Dean and M. P. Wellman. *Planning and Control*. Morgan Kaufmann, 1991.

[DXCR93] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, 1993.

[EL00] S. Ehmann and M. C. Lin. Swift: Accelerated distance computation between convex polyhedra by multi-level marching. In *Proceedings of IEEE/RSJ Conference on Intelligent Robots and Systems*, 2000.

[FDF05] E. Frazzoli, M. A. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Trans. on Robotics*, 21(6):1077–1091, December 2005.

[Fis94] P. A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds*. Prentice-Hall, Inc., 1994.

[Fra] E. Frazzoli. Maneuver-based motion planning for non-linear systems with symmetries. Submitted to Transactions on Robotics and Automation.

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

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

[GHK99] Leonidas J. Guibas, Christopher Holleman, and Lydia E. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Proceedings of IEEE/RSJ Conference on Intelligent Robots and Systems*, 1999.

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

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

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

[Hav] Havoc.com. Havoc engine. http://www.havok.com/content/blogcategory/20/37/.

[HBHL06] K. Hauser, T. Bretl, K. Harada, and J.-C. Latombe. Using motion primitives in probabilistic sample-based planning for humanoid robots. In *Proceedings of Workshop on Algorithmic Robotics*, New York, New York, 2006.

[HK00] Christopher Holleman and Lydia E. Kavraki. A framework for using the workspace medial axis in PRM planners. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1408–1413, 2000.

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

[HKLR00a] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. In *Proceedings of Workshop on Algorithmic Robotics*, April 2000.

[HKLR00b] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Kinodynamic motion planning amidst moving obstacles. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 537–543, 2000.

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

[HLM99] David Hsu, Jean-Claude Latombe, and Rajeev Motwani. Path planning in exapansive configuration spaces. *International Journal of Computational Geometry and Applications*, 9(4/5):495–512, 1999.

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

[Hol04] B. W. Hollocks. Still simulating after all these years - reflections on 40 years in simulation. In *Proc. of the 2004 Operational Research Society Simulation Workshop (SW04)*, pages 209–222, Birmingham, 2004. Operational Research Society.

[Hon] Honda. Asimo. website: http://world.honda.com/ASIMO/.

[HWY86] E. J. Haug, S. C. Wu, and S. M. Yang. Dynamics of mechanical systems with coulomb friction, stiction, impact and constraint addition, deletion i, ii and iii. *Mechanism and Machine Theory*, 21:401–425, 1986.

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

[KD86] Subbarao Kambhampati and Larry S. Davis. Multiresolution path planning for mobile robots. *Journal on Robotics and Automation*, 2(3):135–145, September 1986.

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

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

[KKL96] Lydia E. Kavraki, M. N. Kolountzakis, and Jean-Claude Latombe. Analysis of probabilistic roadmaps for path planning. In *Proceedings of IEEE Conference on Robotics and Automation*, volume 4, pages 3020–3025, 1996.

[KL94] Lydia E. Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proceedings of IEEE Conference on Robotics and Automation*, volume 3, pages 2138–2145, 1994.

[KL98] Lydia E. Kavraki and Jean-Claude Latombe. Probabilistic roadmaps for path planning. In K. Gupta and P. del Pobil, editors, *Proceedings of IEEE Conference on Robotics and Automation*, pages 33–53. John Wiley and Sons LTD., 1998.

[KL00] James J. Kuffner and Steven M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 995–1001, 2000.

[KLMR96] L. E. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan. Randomized query preprocessing in robot motion planning. In *Proceedings of the Symposium on the Theory of Computing*, 1996.

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

[KPLM98] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shell: A higher-order bounding volume for fast proximity queries. In *Proceedings of Workshop on Algorithmic Robotics*, 1998.

[KVdP06] M. Kalisiak and M. Van de Panne. Rrt-blossom: Rrt with a local flood-fill behavior. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1237–1242, May 15-19 2006.

[KvLO96] Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional config-

uration spaces. *Transactions on Robotics and Automation*, 12(4):566–580, August 1996.

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

[LaV06] Steven M. LaValle. *Planning Algorithms*. Cambridge, 2006.

[LB02] S. M. LaValle and M. S. Branicky. On the relationship between classical grid search and probabilistic roadmaps. In *Proceedings of Workshop on Algorithmic Robotics*, 2002.

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

[LC91] M. C. Lin and John F. Canny. Efficient algorithms for incremental distance computation. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 1008–1014, 1991.

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

[LK99] Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 473–479, 1999.

[LK00] Steven M. LaValle and James J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Proceedings of Workshop on Algorithmic Robotics*, 2000.

[LK01a] F. Lamiraux and L. E. Kavraki. Planning paths for elastic objects under manipulation constraints. *International Journal of Robotics Research*, 20(3):188–208, 2001.

[LK01b] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 5:348–400, May 2001.

[LK01c] Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, May 2001.

[LK02] Andrew M. Ladd and Lydia E. Kavraki. Generalizing the analysis of PRM. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 2120–2125, 2002.

[LK05a] A. M. Ladd and L. E. Kavraki. Fast tree-based exploration of state space for robots with dynamics. In M. Erdmann, D. Hsu, M. Overmars, and A. F. van der Stappen, editors, *Algorithmic Foundations of Robotics VI*, pages 297–312. Springer, STAR 17, 2005.

[LK05b] A. M. Ladd and L. E. Kavraki. Motion planning in the presence of drift, underactuation and discrete system changes. In *Robotics: Science and Systems I*, pages 233–241, Boston, MA, June 2005. MIT Press.

[LL96] Florent Lamiraux and Jean-Phillippe Laumond. On the expected complexity of random path planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 3306–3311, 1996.

[LL05] S. R. Lindemann and S. M. LaValle. Smoothly blending vector fields for global robot navigation. In *Proceedings IEEE Conference on Decision and Control*, pages 3353–3559, Seville, Spain, 2005.

[LL06] S. R. Lindemann and S. M. LaValle. Multiresolution approach for motion planning under differential constraints. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 139–144, Orlando, Florida, 2006.

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

[LM97] M. C. Lin and D. Manocha. Efficient contact determination between geometric models. *International Journal of Computational Geometry and Applications*, 7(1):123–151, 1997.

[Lot82] P. Lotstedt. Mechanical systems of rigid bodies subject to unilateral constraints. *SIAM Journal of Applied Mathematics*, 42(3):281–296, 1982.

[Lot84] P. Lotstedt. Numerical simulation of time-dependent contact friction problems in rigid body mechanics. *SIAM Journal of Scientific Statistical Computing*, 5(2):370–393, 1984.

[MBOR86] D. Kozen M. Ben-Or and J. Reif. The complexity of elementary algebra and geometry. *Journal of Computational Sciences*, 32:251–264, 1986.

[MC95] B. Mirtich and J. F. Canny. Impulse-based simulation of rigid bodies. *Proceedings of ACM Interactive 3D Graphics*, 1995.

[Mir97] B. Mirtich. Efficient algorithms for two-phase collision detection. Technical Report TR–97–23, Mitsubishi Electric Research Laboratory, December 1997.

[MK06] M. Moll and L. E. Kavraki. Path planning for deformable linear objects. *IEEE Transactions on Robotics*, 22(4):625–636, 2006.

[MSO94] S. McMillan, P. Sadayappan, and D. E. Orin. Parallel dynamic simulation of multiple manipulator systems: Temporal vs. spatial methods. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(7):982–990, July 1994.

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

[Ov94] Mark H. Overmars and Petr Švestka. A probabilistic learning approach to motion-planning. In *Proceedings of Workshop on Algorithmic Robotics*, pages 19–37, 1994.

[Ov95] Mark H. Overmars and Petr Švestka. A paradigm for probabilistic path planning. Technical report, Utrecht University, March 1995.

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

[OY85] Colm Ó'Dúnlaing and Chee K. Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, March 1985.

[PBC+05] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics*, 21(4):597–608, 2005.

[PC06] M. Pidd and A. Carvalho. Simulation software: not the same yesterday, today or forever. *Journal of Simulation*, pages 1–14, 2006.

[Per83] T. Lozano Perez. Spatial planning: a configuration space approach. *Transactions on Computing*, February 1983.

[PG96] F. Pfeiffer and Ch. Glocker. *Multibody Dynamics with Unilateral Contacts*. Willey Series in Nonlinear Science, New York, 1996.

[PK05] E. Plaku and L. E. Kavraki. Distributed sampling-based roadmap of trees for large-scale motion planning. In *IEEE International Conference on Robotics and Automation*, pages 3879–3884, Barcelona, Spain, April 2005.

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

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *Proceedings of IEEE Conference on Robotics and Automation*, 1994.

[Rei79] John H. Reif. Complexity of the generalized mover's problem. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979.

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

[Rob] Robocup legged league. website: http://www.tzi.de/4legged/.

[RPE+05] L. Ren, A. Patrick, A. A. Efros, J. K. Hodgins, and J. M. Rehg. A data-driven approach to quantifying natural human motion. *ACM Transactions on Graphics*, 24(3):1090–1097, 2005.

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

[SA01] Guang Song and Nancy M. Amato. Randomized motion planning for car-like robots with C-PRM. In *Proceedings of IEEE/RSJ Conference on Intelligent Robots and Systems*, 2001.

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

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

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

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

[SK05] M. Stilman and J. J. Kuffner. Navigation among movable obstacles : Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, 2(4):1–24, 2005.

[SK06] M. Stilman and J. J. Kuffner. Planning among movable obstacles with artificial constraints. In *Proceedings of Workshop on Algorithmic Robotics*, pages 1–20, New York, New York, 2006.

[SL01] G. Sánchez and J.-C. Latombe. A single-query bidirectional motion planner with lazy collision checking. In *Proceedings of International Symposium on Robotics*, 2001.

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

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

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

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

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

[ST96] D. E. Stewart and J. C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *Intl. Journal of Numerical Methods in Engineering*, 39:2673–2691, 1996.

[Ste94] Robot F. Stengel. *Optimal Control and Estimation*. Dover Books, 1994.

[Ste00] D. E. Stewart. Rigid-body dynamics with friction and impact. *SIAM Rev.*, 42(1):3–39, 2000.

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

[SvLO98] S. Sekhavat, P. Švestka, J.-P. Laumond, and M. H. Overmars. Multilevel path planning for nonholonomic robots using semi-holonomic subsystems. *International Journal of Robotics Research*, 17:840–857, 1998.

[TMD+06] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Winning the darpa grand challenge. *Journal of Field Robotics*, 2006. accepted for publication.

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

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

[YJSL05] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle. Dynamic-domain rrts: Efficient exploration by controlling the sampling domain. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 3856–3861, Barcelona, Spain, 2005.