RICE UNIVERSITY

Kinodynamic Motion Planning for High-dimensional Physical Systems

by

Ioan Alexandru Şucan

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

Master of Science

Approved, Thesis Committee:

Dr. Lydia E. Kavraki (Chair)
Professor,
Computer Science, Rice University

Dr. Joe Warren
Professor,
Computer Science, Rice University

Dr. Marcia K. O'Malley
Assistant Professor,
Mechanical Engineering, Rice University

Houston, Texas

2008

Abstract

Kinodynamic Motion Planning for High-dimensional Physical Systems

by Ioan Alexandru Şucan

This thesis presents a kinodynamic motion planner, Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (`KPIECE`), specifically designed for systems with complex dynamics, where physics-based simulation is necessary. A multiple-level grid-based discretization is used to estimate the coverage of the state space. The coverage estimates help the planner detect the less explored areas of the state space. The planner also keeps track of the boundary of the explored region of the state space and focuses exploration on the less covered parts of this boundary. Extensive experiments show `KPIECE` provides computational gain over state-of-the-art methods and allows solving some harder, previously unsolvable problems. A shared memory parallel implementation is presented as well. This implementation provides better speedup than an embarrassingly parallel implementation by taking advantage of the evolving multi-core technology.

Acknowledgements

First of all, I would like to thank my advisor, Dr. Lydia Kavraki, for her guidance and help with this work, and the Physical and Biological Computing Group for their valuable comments and suggestions.

I would like to thank Mark Yim and Jonathan Kruse for their help in defining the CKBot `ODE` model. Many thanks go to Marius Şucan for designing most of the images this document includes, and to Derek Ruths for naming the algorithm (`KPIECE`).

This work was supported in part by NSF IIS 0713623 and Rice University funds. The experiments were run on equipment obtained by NSF CNS 0454333 and NSF CNS 0421109 in partnership with Rice University, AMD and Cray.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Given a robotic system that can be controlled in a specific way, motion planning is the problem of taking that system from a given starting state to a goal state while respecting a set of constraints [1, 2, 3]. Over the last two decades, this field has grown from one that considered basic geometric problems, such as the piano movers' problem [4], to a field that addresses planning for complex robots (see Figure 1.1) with kinematic and dynamic constraints [5, 6]. Although the motion planning problem stemmed from the artificial intelligence and robotics fields, its applications have expanded to other domains such as graphics and computational biology [7, 8, 9]. Exploration robots, tour guides, surgical robots, digital actors, folding proteins are only a few examples of cases where motion planning is used.

A simplified case of the motion planning problem is finding a continuous collision-free path for a robot consisting of polyhedral parts among polyhedral obstacles. This version of the problem is usually referred to as motion planning under geometric

constraints, path planning or kinematic motion planning and has been shown to be PSPACE-complete [10, 11]. When considering some of the physical limitations the robot exhibits, such as bounded forces, the problem is known as planning under differential constraints, or kinodynamic motion planning. In this latter case, it is not known if the problem is even decidable except in some particular cases [12, 13, 14].

Complete algorithms for the kinematic motion planning problem and some cases of the kinodynamic motion planning problem exist [1, 2, 3], however they are impractical for realistic problems. Significant improvements were achieved with the development of sampling-based algorithms [15, 16, 17]. More details on this class of methods and a literature survey follow in Section 2.1.

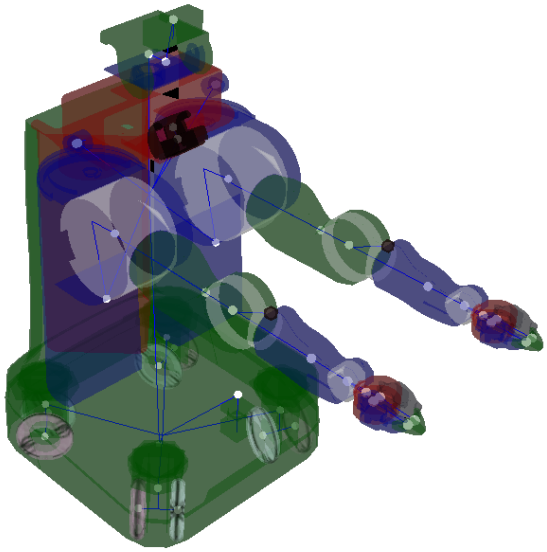

Figure 1.1: A model of a complex robot: `PR2` from Willow Garage. This robot has 36 degrees of freedom.

1.1 Problem Definition

Before defining the motion planning problem as it is used in this work, we introduce a few notations:

- Q is a differentiable manifold of dimension m, representing the state space of the robotic system.

  A single element $q \in Q,\ q = (q_1, ..., q_m)$ completely describes the state of a robotic system. This means that each component q_i defines a parameter of the system; such parameters can be joint angles, positions in space, velocities, accelerations, etc.

- $g(q, \dot{q}) \geq 0$ is a set of constraints the robotic system must satisfy in its motion (*e.g.*, collision avoidance, maximum speeds, maximum forces).

  - $Q_{obs} \subseteq Q$ is the set of invalid states.

    A state is invalid if it would cause the robotic system to break one of the constraints under which it operates.

  - $Q_{free} = Q \setminus Q_{obs}$ is the set of valid states.

    This is the set of states in which the robot is allowed to be.

- U is a manifold representing the control space for the robotic system.

  A single element $u \in U$ represents the set of inputs given to a robotic system at a point in time.

An instance of the motion planning problem addressed here can be formally defined by the tuple $S = (Q, U, I, G, f)$ where Q is the state space, U is the control space, $I \subset Q_{free}$ is the set of initial states, and $G \subset Q$, $G \cap Q_{free} \neq \emptyset$ is the set of goal states. The dynamics are described by a forward propagation routine $f : Q \times U \rightarrow TgQ$, where TgQ is the tangent bundle of Q (f does not need to be explicit). A solution to a motion planning problem instance consists of a sequence of controls $u_1, \ldots, u_n \in U$ and times $t_1, \ldots, t_n \in \mathbb{R}^{\geq 0}$ such that $q_0 \in I$, $q_n \in G$ and $q_k \in Q_{free}, k = 1, \ldots, n$, can be obtained sequentially by integration of f.

For the purposes of this work, the function f is computed using the Open Dynamics Engine (`ODE`) [18] physics simulator. Instead of equations of motion to be integrated, a model of a robot and its environment needs to be specified. More details about physics simulation follow in Section 2.2.

1.2 Contribution

This thesis introduces a new sampling-based motion planning algorithm that is able to handle high dimensional systems with complex dynamics. The algorithm is innovative in the sense that while it is general (uses physics simulation for evaluating robotic systems), it reduces both runtime and memory requirements by making better use of the information collected during the planning process. This planning algorithm will be referred to as Kinodynamic Planning by Interior-Exterior Cell Exploration (`KPIECE`). While there are other planners for systems with complex dynamics, `KPIECE` was designed with additional goals in mind. One such design goal is the ease

of use for systems where only a forward propagation routine is available (that is, the simulation of the system can only be done forward in time). Another goal is that no state sampling and no distance metric are required. These requirements make `KPIECE` particularly well suited for complex systems described by physical models instead of equations of motion, since in such cases only forward propagation is available and sampling of states is in general expensive. Since `KPIECE` does not need to evaluate distance between states, it is also well suited for systems where a distance metric is hard to define or when the goal is not known until it is actually reached. It is typical for sampling-based planners to spend more than 90% of their computation performing forward propagation. Since physics simulation is considerably more expensive than integration of motion models, it is essential to use as few propagation steps as possible. This fact was a major motivation behind this work and as will be shown later, `KPIECE` provides significant computational improvements over previous methods (up to two orders of magnitude), which allows tackling more complex problems that could not be previously addressed. Since motion planning is usually a subproblem of a more complex task, it is generally desirable to have fast methods for the computation of motion plans. To this end, `KPIECE` was also designed with shared memory parallelism in mind and the developed implementation can take advantage of the emerging multi-core technology (multiple cores per chip). The implementation can use a variable number of processors and shows super-linear speedup in some cases. The combination of obtained speedup and physics-based simulation, could make `KPIECE` fast and accurate enough to be applicable in real-time motion planning for complex reactive robotic systems.

1.3 Thesis Overview

The rest of the thesis is organized as follows: Chapter 2 describes background and related work, Chapter 3 contains a description of the proposed algorithm, and Chapter 4 presents experiments using `KPIECE`. Conclusions and future work are in Chapter 5.

Chapter 2

Background and Previous Work

Around two decades ago, the focus in motion planning was on *planning under geometric constraints*. This problem is sometimes referred to as the piano movers' problem, or in 2D, the sofa movers' problem, and it was the subject of extensive research [4, 19, 20]. A number of complete algorithms were developed for various cases of the problem and it was eventually shown to be PSPACE-complete [10, 11]. The developed algorithms are of theoretical interest only, since they are computationally prohibitive and difficult to implement. Techniques such as cell decomposition methods and potential fields [1, 2, 3] were studied as well, but few were successful at solving problems where the state space is highly dimensional [21].

In addition to geometric constraints, planning for real robotic systems requires accounting for dynamic constraints (*e.g.*, friction, gravity, limits in forces). In general it is not known if this version of the problem, *planning under differential constraints*, is even decidable [13]. However, for the simplified case of a point mass robot, a polyno-

mial algorithm exists [12]; the reconfiguration of modular robots under kinodynamic constraints is possible in $\Theta(\sqrt{n})$ time under certain assumptions [14].

The proven difficulty of planning under geometric constraints and the need to consider even more complex versions of the problem, such as planning under differential constraints, pushed the research in motion planning towards approximate techniques. There are multiple directions of research that deal with such approximate techniques, and this work continues in one of these directions, namely sampling-based motion planning, a direction in which promising results have been shown for planning under differential constraints [2, 3, 5, 53].

2.1 Sampling-based Motion Planning

Much of the recent progress in motion planning is attributed to the development of sampling-based algorithms [2, 3]. Sampling-based motion planning algorithms developed up to this point are only probabilistically complete [22, 23], which means if a solution exists, it will be eventually found, but if no solution exists, the algorithm will not terminate. The main idea behind this class of methods is the notion of sampling the state space Q of the robotic system. Since an exact representation of Q_{obs} is not usually available (and would be hard to compute), collision checking can be used in the environment of the robotic system to test whether a particular state $q \in Q$ is valid or not. With the development of collision checking algorithms [24], this test can be performed quickly. Sampling states from Q and testing whether they are in Q_{free} allows sampling-based algorithms to carry out a systematic search of

the continuous space Q. The search consists of finding path segments $[q_a, q_b]$ such that moving the robotic system from q_a to q_b according to its motion capabilities will not cross any state in Q_{obs}. Obtaining multiple such valid path segments yields an approximate representation of the connectivity of Q_{free}. Various strategies exist for producing sequences of valid segments (motion plans) such that two given states q_{start} and q_{goal} can be connected.

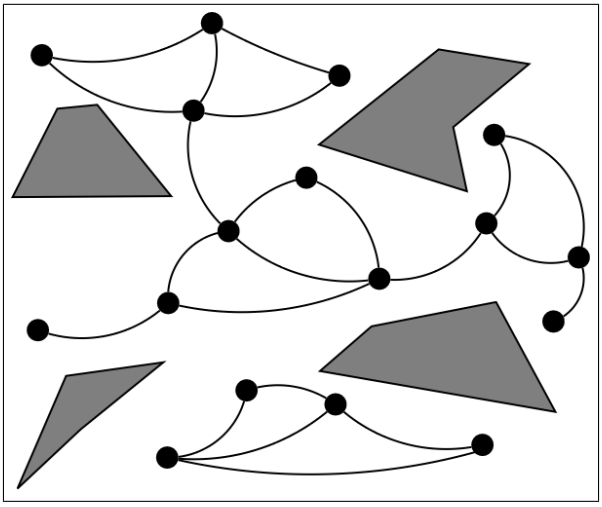

Figure 2.1: An example roadmap for `PRM`. Dark regions represent Q_{obs}.

One of the most influential algorithms in sampling-based motion planning was `PRM` [25, 26, 27]. This method provided a coherent framework for many earlier works that used sampling and opened new directions for research. In a preprocessing step, this algorithm constructs a roadmap – a graph of valid robot states obtained by sampling, connected by valid robot paths (shown in Figure 2.1). In a second step, the desired starting and ending states of a robot are connected to the roadmap and the problem of computing a valid path between them is reduced to a graph search algorithm.

While the underlying principle is simple, the algorithm is efficient and problems that are intractable for complete algorithms can be solved [2]. Because constructing the roadmap requires connection of states by paths, PRM and algorithms based on PRM are better suited for planning under geometric constraints, so finding the input that connects one state to the other (the steering problem) is not necessary [28, 29].

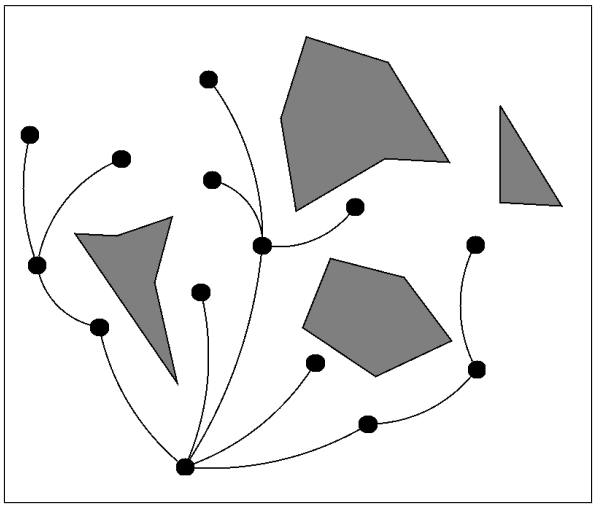

Figure 2.2: An example tree structure. Dark regions represent Q_{obs}.

PRM set the foundation for many other algorithms [30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]. Among these, a notable class is that of tree-based planners. As the name suggests, tree-based planners grow a tree in the state space of the robotic system (as shown in Figure 2.2). The initial tree consists of the robot's starting state. Newly sampled states are connected to some already existing state in the tree. This category of motion planners is more appropriate for planning under differential constraints as they only perform forward integration, thus avoiding the need to solve the steering problem [28, 29]. There are various

ways to guide the tree expansion. Rapidly-exploring Random Trees (RRT) expand towards randomly produced states [32, 33], Expansive Space Trees (EST) attempt to detect less explored regions and expand towards them [34, 35]. A more recent development is the idea of a Path-Directed Subdivision Tree (PDST) [44, 50, 51]. PDST uses an adaptive subdivision of the state space and a deterministic priority scheme to guarantee coverage, avoiding the use of a metric. Other recent work considers issues like safety and exploration of unknown environments: the Greedy, Incremental, Path-directed (GRIP) planner [48]. The Discrete Search Leading continuous eXploration (DSLX) planner uses discrete paths in a discretization of the workspace to lead the continuous tree exploration [49].

KPIECE tries to push the current limits of planning under differential constraints by combining new ideas with ideas from previous work. More details about this follow in Chapter 3.

2.2 Physics Simulation

Sampling-based motion planning algorithms typically need to evaluate the behaviour of the robotic system when certain controls are given as input. This information can be obtained through the numerical integration of motion models of the robotic system. However, as robotic systems become more complex, accounting for additional constraints such as gravity and friction becomes a necessity. In this case, physics simulation can replace integration of motion models.

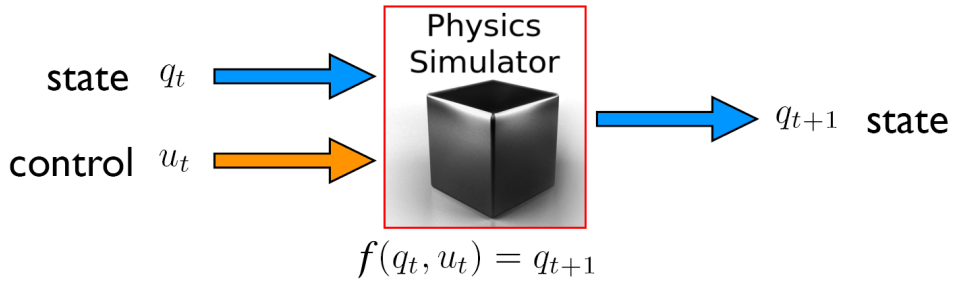

Figure 2.3: The operation of a physics simulator: one simulation step.

Figure 2.3 shows the operation of a typical physics simulator. As in the case of numerical integration, time is discretized and the function describing the state of the robotic system is evaluated at these discrete points in time only. In the case of physics simulation, this discretization interval typically does not vary and is referred to as the "simulation step". When the system is in a particular state q_t, applying the control u_t takes the system to state q_{t+1}, within one simulation step.

One essential difference between integration of motion models and physics simulation is that simulation backward in time is not always possible. For example, if an object is dropped from a certain height, forward simulation can correctly account for gravity and compute that the object will fall and eventually hit the ground. Attempting to simulate backward in time once the object is at rest is not well defined: an infinite number of possible previous states exist.

An additional difference is that at every step of the simulation, contacts between objects need to be determined. This implies that collision checking needs to be performed. When integrating motion models, performing collision checking is not always a necessity (*e.g.*, only the final state of a motion is needed), which makes physical

simulation significantly more computationally expensive. Although simulation incurs more computational costs than simple integration, the benefits outweigh the costs: increased accuracy is available since physics simulators take into account more dynamic properties of the robot and constructing models of systems is easier and less error prone than deriving equations of motion. Nevertheless, limited numerical precision will remain a problem for both approaches.

This work uses physics simulation as a black box only and does not attempt to improve on simulation algorithms. Currently, many options exist when it comes to physics simulation libraries, both commercial and free. Among the better known ones are `PhysX`, `ODE`, `Vortex`, `bullet`, etc. More details on these libraries can be found in [52]. For the purposes of this work, the Open Dynamics Engine (`ODE`) [18] library was used. This library was chosen because it is open source and it has previously been used in the context of sampling-based motion planning [5, 50, 53].

Chapter 3

Algorithm

A high-level description of the algorithm is provided before the details are presented. `KPIECE` iteratively constructs a tree of motions in the state space Q of the robot. Each motion $\mu = (s, u, t)$ is identified by a state $s \in Q$, a control $u \in U$ and a duration $t \in \mathbb{R}^{\geq 0}$. The control u is applied for duration t from s to produce a motion. It is possible to split a motion $\mu = (s, u, t)$ into $\mu_1 = (s, u, t_a)$ followed by $\mu_2 = (\int_{t_0}^{t_0+t_a} f(s(\tau), u)d\tau, u, t_b)$, where $s(\tau)$ identifies the state at time τ and $t_a+t_b = t$. In this exploration process, it is important to cover as much of the state space as possible, as quickly as possible. For this to be achieved, estimates of the coverage of Q are needed. To this end, the discretization described in Section 3.1 is employed. When a less covered area of the state space is discovered, the tree of motions is extended in that area. This process is iteratively executed until a stopping condition such as time limit exceeded or goal reached is satisfied.

3.1 Discretization

During the course of its run, the motion planner must decide which areas of the state space merit further exploration. As the size of the tree of motions increases, making this decision becomes more complex. There are various strategies to tackle this problem (*e.g.*, [33, 35, 37, 49, 51]). The approach taken in this work is to construct a discretization that allows the evaluation of the coverage of the state space. This discretization consists of k levels $\mathcal{L}_1, ..., \mathcal{L}_k$, as shown in Figure 3.1 and Figure 3.2. Each of these levels is a grid where cells are polytopes of fixed size. The number of levels and cell sizes are predefined, however, cells are instantiated only when they are needed. The purpose of these grids is to cover the area of the space that corresponds to the area spanned by the tree of motions. Each of the levels provides a different resolution for evaluating the coverage. Coarser resolution (higher levels) can be used initially to find out roughly which area is less explored. Within this area, finer resolutions (lower levels) can then be employed to more accurately detect less explored areas. The following is a formal definition of a k-level discretization:

- for $i \in \{1, ..., k\}$: $\mathcal{L}_i = \{p_i | p_i$ is a cell in the grid at level $i\}$
- for $i \in \{2, ..., k\}$: $\forall p \in \mathcal{L}_i, \mathcal{D}_p = \{q \in \mathcal{L}_{i-1} | q \subset p\}$, such that
  - $\forall p \in \mathcal{L}_i, \mathcal{D}_p \neq \emptyset$
  - $\bigcup_{p \in \mathcal{L}_i} \mathcal{D}_p = \mathcal{L}_{i-1}$
  - $\forall p, q \in \mathcal{L}_i, p \neq q \rightarrow \mathcal{D}_p \cap \mathcal{D}_q = \emptyset$

The tree of motions exists in the state space Q, but since the dimension of this space may be too large, the discretization is typically imposed on a projection of the state space, $\mathcal{E}(Q)$. It is assumed that if the projected space $\mathcal{E}(S)$ (with $S \subseteq Q$) is well covered, S is well covered. The use of such a projection $\mathcal{E}(Q)$ was also discussed in [49, 50]. Finding projections $\mathcal{E}$ that are appropriate for approximating coverage is a research problem in itself. In this work it is assumed that such projections $\mathcal{E}$ are available, when needed. Section 4.1 presents simple cases of $\mathcal{E}$ used in this work. An important result shown in this thesis is that such simple projections work for complex problems.

For any motion μ, each level of discretization contains a cell that μ is part of. A motion μ is considered to be part of a grid cell p if there exists a state s along μ such that the projection $\mathcal{E}(s)$ is inside the bounding box of cell p. If a motion spans more than one cell at the same level of discretization, it is split into smaller motions such that no motions cross cell boundaries. This invariant is maintained to make sure each motion is accounted for only once. For every motion μ, there will be exactly one cell at every level of discretization that μ is part of. This set of cells forms a tuple $\mathbf{c} = (p_1, ..., p_k), p_i \subset p_{i+1}, p_i \in \mathcal{L}_i$ and will be referred to as the "cell chain" for μ. Since cells in $\mathcal{L}_1$ will determine whether a motion is split, we augment the definition of the discretization:

- $\forall p \in \mathcal{L}_1, \mathcal{M}_p = \{m_i | m_i \text{ is a motion contained in } p\}$

For all $p \in \mathcal{L}_1$ we say p contains $\mathcal{M}_p$ and for all $p \in \mathcal{L}_i, i > 1$ we say p contains $\mathcal{D}_p$. While the discretization spans the potentially very large projection space $\mathcal{E}(Q)$, cells

are instantiated only when a motion that is part of them is found, hence the grids are not fully instantiated. This allows the motion planner to limit its use of memory to reasonable amounts. The size of the grid cells is discussed in Section 3.4.

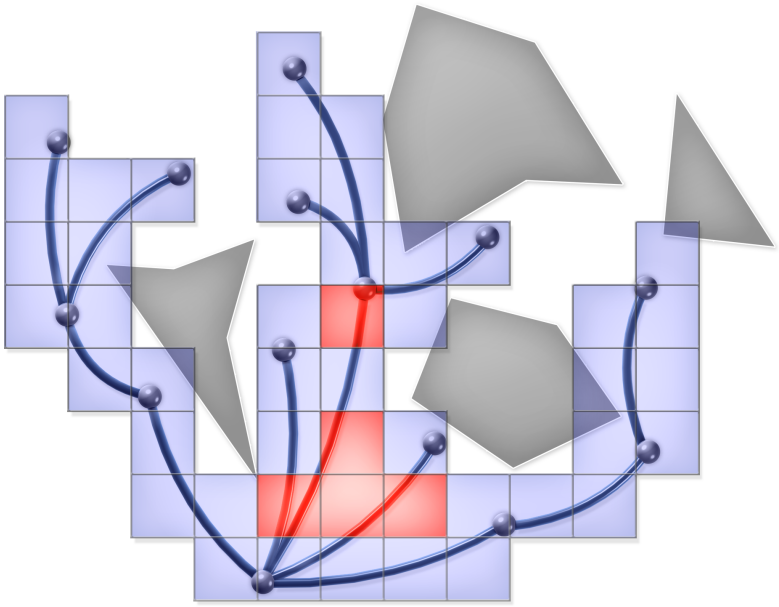

Figure 3.1: One level of discretization. Interior cells are differentiated from exterior cells.

A distinguishing feature of `KPIECE` is the notion of interior and exterior cells. A cell is considered exterior if it has less than $2n$ instantiated neighboring cells (diagonal neighboring cells are ignored) at the same level of discretization, where n is the dimension of $\mathcal{E}(Q)$. Cells with $2n$ neighboring cells are considered interior (there can be no more than $2n$ non-diagonal neighboring cells in an n-dimensional space). As the algorithm progresses and new cells are created, some exterior cells will become interior (see Figure 3.1). When larger parts of the state space are explored, most

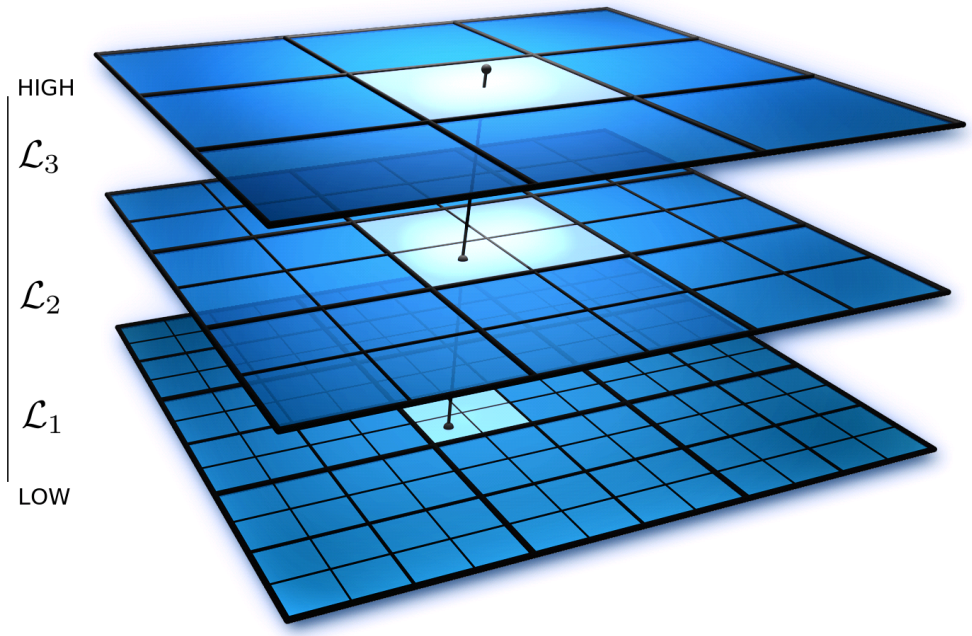

Figure 3.2: An example discretization with three levels. The line intersecting the three levels defines a cell chain. Cell sizes at lower levels of discretization are integer multiples of the cell sizes at the level above.

cells will be interior. However, for very high dimensional spaces, to avoid having only exterior cells, the definition of interior cells can be relaxed and cells can be considered interior before all $2n$ neighboring cells are instantiated. For the purposes of this work, this relaxation was not necessary.

With these notions in place, a measure of coverage of the state space can be defined. For a cell $p \in \mathcal{L}_1$, the coverage is simply the sum of the durations of the motions in $\mathcal{M}_p$. For higher levels of discretization, the coverage of a cell $p \in \mathcal{L}_i, i > 1$ is the number of instantiated cells in $\mathcal{D}_p$.

3.2 Algorithm Execution

A run of the `KPIECE` algorithm proceeds as described in Algorithm 1. The tree of motions is initialized to a motion defined by the initial state q_{start}, a null control and duration 0 [line 1]. Adding this motion to the discretization will create exactly one exterior cell for every level of discretization [lines 2,3].

At every iteration, a cell chain $\mathbf{c} = (p_1, ..., p_k)$ is sampled. This means $p_i \in \mathcal{L}_i$ will have to be selected, from p_k to p_1, as will be shown later. It is important to note here that "samples" in the case of `KPIECE` are chains of cells. This can be regarded as a natural progression (selection of "volumes") from the selection of states ("points") as in the case of `RRT` and `EST`, and selection of motions ("curves") as in the case of `PDST`. Experiments show that selecting chains of cells benefits from the better estimates of coverage that can be maintained for cells at each level, as opposed to estimates for single motions or states. Sampling a cell chain $\mathbf{c} = (p_1, ..., p_k)$ is a k-step process that proceeds as follows: the decision to expand from an interior or exterior cell is made [line 5], with a bias towards exterior cells. An instantiated cell, either interior or exterior, is then deterministically selected from $\mathcal{L}_k$, according to the cell importance (higher importance first). The idea of deterministic selection was inspired by [51], where it has been successfully used. The importance of a cell p, regardless of the level of discretization it is part of, is computed as:

$$Importance(p) = \frac{\log(\mathcal{I}) \cdot \texttt{score}}{\mathcal{S} \cdot \mathcal{N} \cdot \mathcal{C}}$$

where $\mathcal{I}$ stands for the number of the iteration at which p was created, `score` is

initialized to 1 but may later be updated to reflect the exploration progress achieved when expanding from p, $\mathcal{S}$ is the number of times p was selected for expansion (initialized to 1), $\mathcal{N}$ is the number of instantiated neighboring cells at the same level of discretization, and $\mathcal{C}$ is a positive measure of coverage for p, as described at the end of Section 3.1.

Once a cell p is selected, if $p \notin \mathcal{L}_1$, further levels of discretization can be used to better identify the more important areas within p. The selection process continues recursively: an instantiated cell from $\mathcal{D}_p$ is subsequently selected using the method described above until the last level of discretization is reached and the sampling of the cell chain is complete. At the last level, a motion μ from $\mathcal{M}_p$ is picked according to a half-normal distribution [line 6]. The half-normal distribution is used because order is preserved when adding motions to a cell and motions added more recently are preferred for expansion. A state s along μ is then chosen uniformly at random [line 7]. Expanding the tree of motions continues from s [line 9].

The controls applied from s are selected uniformly at random from U [line 8]. The random selection of controls is what is typically done if other means of control selection are not available. This choice is not part of the proposed algorithm, and can be replaced by other methods, if available.

If the tree expansion was successful, the newly obtained motion is added to the tree of motions and the discretization is updated [lines 11,13]. An estimate of the achieved progress is then computed. For every level of discretization j, the coverage of some cells may have increased:

$$\begin{aligned} \Delta\mathcal{C}_j &= \Sigma_{p\in\mathcal{L}_j}\Delta p, \text{ where } \Delta p = \text{increase in coverage of } p \\ P_j &= \alpha + \beta \cdot \text{ (ratio of } \Delta\mathcal{C}_j \text{ to time spent computing simulations).} \end{aligned}$$

P_j is considered the progress at level j [line 16]. The values α and β are implementation specific and should be chosen such that $P_j > 0$, and $P_j \geq 1$ implies good progress. The offset α needs to be strictly positive since the increase in coverage can be 0 (*e.g.*, in case of an immediate collision). The value of P_j is also used as a penalty if not enough progress has been made ($P_j < 1$): the cell at level j in the selected cell chain has its `score` multiplied by P_j [line 17]. If good progress has been made ($P_j \geq 1$), the value of P_j is ignored, since we do not want to over-commit to specific areas of the space.

3.3 Implementation Details

To aid in the implementation of the KPIECE algorithm, an efficient grid data-structure (`Grid`) is defined. `Grid` maintains the list of cells it contains, grouped into interior and exterior, sorted according to their importance. To maintain the lists of interior and exterior cells sorted, binary heaps are used. For every cell p, `Grid` also maintains some additional data: another `Grid` instance (stands for $\mathcal{D}_p$), for all but the lowest level of discretization, and for the lowest level of discretization, an array of motions (stands for $\mathcal{M}_p$). Algorithm 2 shows the steps for adding motions to `Grid`.

Algorithm 1 KPIECE(q_{start}, $N_{iterations}$)

Let μ_0 be the motion of duration 0 containing solely q_{start}
Create an empty `Grid` data-structure G
G.AddMotion(μ_0)
for $i \leftarrow 1...N_{iterations}$ **do**
  Select a cell chain **c** from G, with a bias on exterior cells (70% - 80%)
  Select μ from **c** according to a half normal distribution
  Select s along μ
  Sample random control $u \in U$ and simulation time $t \in \mathbb{R}^+$
  Check if any motion $(s, u, t_\circ)$, $t_\circ \in (0, t]$ is valid (forward propagation)
  if a motion is found **then**
    Construct the valid motion $\mu_\circ = (s, u, t_\circ)$ with $t_\circ$ maximal
    If $\mu_\circ$ reaches the goal region, **return** path to $\mu_\circ$
    G.AddMotion($\mu_\circ$)
  end if
  for every level $\mathcal{L}_j$ **do**
    $P_j = \alpha + \beta \cdot$ (ratio of increase in coverage of $\mathcal{L}_j$ to simulated time)
    Multiply the `score` of cell p_j in **c** by P_j if and only if $P_j < 1$
  end for
end for

Algorithm 2 AddMotion(s, u, t)

Split (s, u, t) into motions $\mu_1, ..., \mu_k$ such that $\mu_i, i \in \{1, ..., k\}$ does not cross the boundary of any cell at the lowest level of discretization
for $\mu_\circ \in \{\mu_1, ..., \mu_k\}$ **do**
  Find the cell chain corresponding to $\mu_\circ$
  Instantiate cells in the chain, if needed
  Add $\mu_\circ$ to the cell at the lowest level in the chain
  Update coverage measures and lists of interior and exterior cells, if needed
end for

`KPIECE` is implemented as a set of components that interact with `Grid`. The main component of `KPIECE`, the tree expansion, was presented in Section 3.2. Goal biasing is implemented as another component and is described in Section 3.5.

3.4 Computing the Discretization

An important issue not discussed so far is the selection of number of levels in the discretization and the grid cell sizes. This section presents a method to compute these cell sizes if the discretization is assumed to consist of only $\mathcal{L}_1$ (a one-level discretization).

While `KPIECE` is running, we can keep track of averages of how many motions per cell there are, how many parts a motion is split into before it is added to the discretization, and the ratio of interior to exterior cells. While we do not know how to compute optimal values for these statistics (if they exist), there are certain ranges that may work better than others. In particular, it was observed that for good runtime performance the following should hold:

- Less than 10% of the motions cover more than 2 cells in one simulation time-step. This value should be in general less than 1% as the event occurs only when the velocity of the robotic system is very high.
- At least 50% of the motions need to be 3 simulation time-steps or longer.
- The average number of parts in which a motion is split should be larger than 1 but not higher than 4.

- As the algorithm progresses, at least some interior cells need to be created.
- The average number of samples per cell should be in the range of tens to hundreds.

Based on collected statistics and these observations, it can be automatically decided whether the cell sizes used for $\mathcal{L}_1$ are good, too large or too small. This information is reported for each dimension of the space. If the used cell size is too small or too large in some dimension, the size in that dimension is increased or decreased, respectively, by a factor larger than 1 and the algorithm is restarted. This process usually converges in 2 or 3 iterations.

These statistics do not offer any information about higher levels of discretization, nor do they provide information about how many levels of discretization should be used. The presented constants are implementation specific, but they seem not to vary across the examined robotics systems.

3.5 Goal Biasing

Goal biasing is a standard technique used by motion planners to reduce their runtime. The basic idea is that if the planner knows where the goal is, it can greedily attempt to reach it [32]. This section presents how goal biasing can be used by `KPIECE`.

If a heuristic $h : Q \to \mathbb{R}^+$ for estimating the distance to the goal is available, the information provided by that heuristic can be used in computing the cell importance: cells closer to the goal get higher `score` and are thus selected for expansion more

often. In particular, when a cell is instantiated, its `score` is set to $h(s)$ where s is the last state of the motion m that required the creation of the cell (we say $h^*(m) = h(s)$). While this technique usually contributes to reducing the runtime of the algorithm, it may become a problem and slow down the algorithm when obstacles block the way and higher `score` is given to cells that cannot be crossed. In such cases, learning becomes an essential component since the higher score that was assigned by the heuristic is decreased as progress stagnates.

In addition to influencing the `score` of cells, the biasing component also maintains a set of so-called good motions, $\mathcal{B}$. For every motion m, $h^*(m)$ is computed; $\mathcal{B}$ is a limited-size set (at most 30) of motions with the largest values of $h^*(m)$. With a low probability (usually 5%), tree expansion is continued from a motion in $\mathcal{B}$, rather than from a motion selected based on the coverage estimates. An additional constraint imposed on the motions in $\mathcal{B}$ is that they have to be in different cells at the lowest level discretization. This constraint prevents having all motions in $\mathcal{B}$ be almost the same.

Chapter 4

Experiments

The presented algorithm was benchmarked against well-known efficient algorithms (RRT, EST, PDST) with three different robotic systems, in different environments. For modeling the robots, the ODE [18] physics-based simulator was used. For the implementations of RRT [33] and EST [34], the OOPSMP framework was used [54]. A plugin for linking OOPSMP with the ODE simulator was developed. Every effort was made to tune the parameters of both RRT and EST. For RRT, a number of different metrics were tested for each robot and experiments are presented with the metric that performed best. In addition, random controls were selected instead of attempting to find controls that take the robotic system toward a desired state, as this strategy seemed to provide better results. For EST, the nodes to expand from were selected both based on their degree [35] and based on a grid subdivision of the state space [37]. Experiments are shown for the selection strategy that performed best. PDST and KPIECE were implemented by the author. A projection was defined for each robot and the

same projection was used for both PDST and KPIECE. In addition to the projection, KPIECE needs a discretization to be defined for each robot. When comparing with other algorithms, only discretizations computed as shown in Section 3.4 were used. Separate experiments are shown when using empirically chosen discretizations with multiple levels. Explanations on how these multiple levels were chosen are given later in this section. No goal biasing was used for any of the algorithms, unless otherwise specified. Experiments for RRT with biasing are shown as $\texttt{RRT}_b$ and experiments for KPIECE with biasing are marked as $\texttt{KPIECE}_b$. All implementations are in C++ and were tested on the Rice Cray XD1 Cluster, where each machine runs at 2.2 Ghz and has 8 GB RAM. For each system and each of its environments, each algorithm was executed 50 times. The best two and worst two results in terms of runtime were discarded and the results of the remaining 46 runs were averaged. The time limit was set to one hour and the memory limit was set to 2 GB. If an execution exceeded the time or memory limit, its execution time was considered equal to the time limit, for the purpose of computing average runtimes.

4.1 Robot Models

Three different robots were used in benchmarking the planner, to show its generality: a modular robot, a car, and a blimp. These robots have been chosen to be different in terms of the difficulties they pose to a motion planner. Details on what these difficulties are follow in the next paragraphs. ODE version 0.9 was used to model the robots. The used simulation step size was 0.05s.

4.1.1 Modular Robot

The model for this robot was created in collaboration with Mark Yim[1], and characterizes the CKBot modules [55] used by the GRASP Laboratory of Robotics Research and Education[2]. Using these modules, different robots can be constructed. Each CKBot module contains one motor. An ODE model for serially linked CKBot modules has been created [5]. The task is to compute the controls for lifting the robot from a vertical down position to a vertical up position for varying number of modules, as shown in Figure 4.1. Each module adds one degree of freedom. The controls represent torques that are applied by the motors inside the modules. The difficulty of the problem lies in the high dimensionality of the control and state spaces as the number of modules increases, and in the fact that at maximum torque, the motors in the modules are only able to statically lift approximately 5 modules. Consequently, the planner has to find swinging motions to solve the problem. The employed projection $\mathcal{E}$ was a 3-dimensional one, the first two dimensions being the (x, z) coordinates of the last module (x, z is the plane observed in Figure 4.1) and the third dimension, the square root of the sum of squares of the rotational velocities of all the modules. The environments the system was tested in are shown in Figure 4.1.

[1]Mark Yim is with the Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania yim@grasp.upenn.edu

[2]http://www.grasp.upenn.edu/

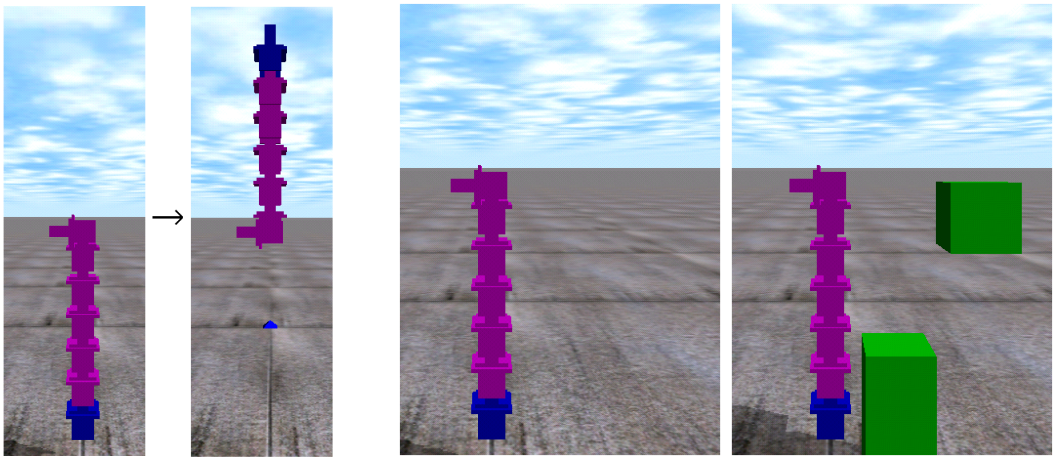

Figure 4.1: Left: start and goal configurations. Right: environments used for the chain robot (7 modules). Experiments were conducted for 2 to 10 modules. In the case without obstacles, the environments are named chain1-x where x stands for the number of modules used in the chain. In the case with obstacles, the environments are named chain2-x.

4.1.2 Car Robot

A model of a car [2] was created. The model is fairly simple and consists of five parts: the car body and four wheels. Since `ODE` does not allow for direct control of accelerations, desired velocities are given as controls for the forward velocity and steering velocity (as recommended by the developers of the library). These desired velocities go together with a maximum allowed force. The end result is that the car will not be able to achieve the desired velocities instantly, due to the limited force. In effect, this makes the system a second order one. The employed projection $\mathcal{E}$ was the (x, y) coordinates of the center of the car body. The environments the system was tested in are shown in Figure 4.2.

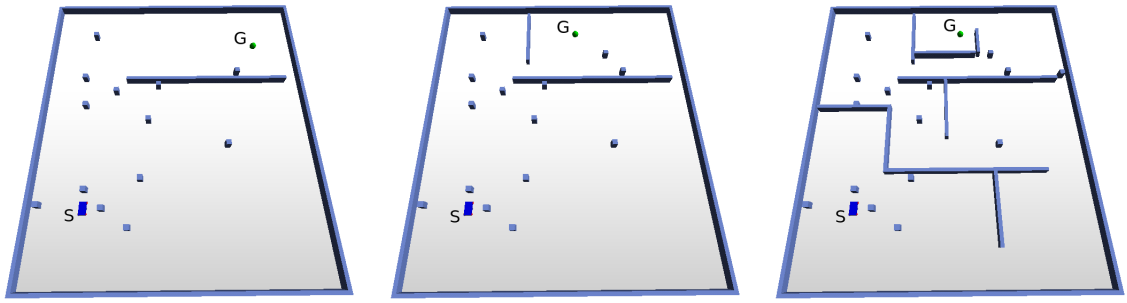

Figure 4.2: Environments used for the car robot (car-1, car-2, car-3). Start and goal configurations are marked by "S" and "G".

4.1.3 Blimp Robot

The third robot that was tested was a blimp robot [51]. The motion in this case is executed in a 3D environment. This robot is particularly constrained in its motion: the blimp must always apply a positive force to move forward (slowing down is caused by friction), it must always apply an upward force to lift itself vertically (descending is caused by gravity) and it can turn left or right along the direction of forward motion. Since `ODE` does not include air friction, a Stokes model of drag was implemented for the blimp. The employed projection $\mathcal{E}$ was the (x, y, z) coordinates of the center of the blimp. The environments the system was tested in are shown in Figure 4.3.

4.2 Experimental Results

In terms of runtime, when compared to other algorithms such as `RRT`, `EST`, and `PDST`, Table 4.1 shows significant computational gains for `KPIECE` (Figure 4.4 and Figure 4.5 show the data in Table 4.1 as well, in a different format). In particular, as

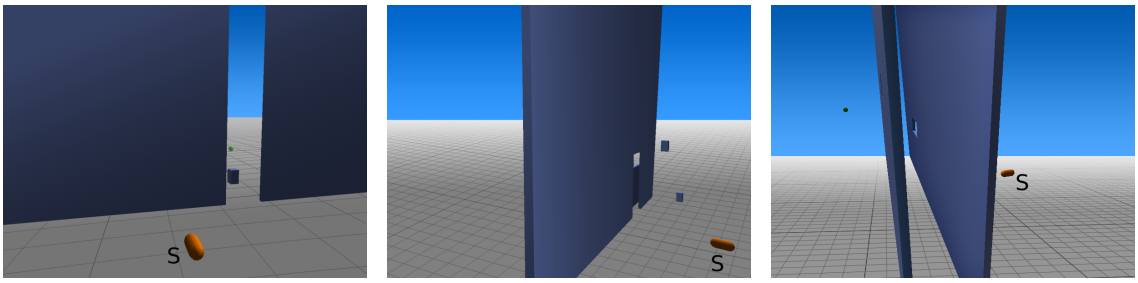

Figure 4.3: Environments used for the blimp robot (blimp-1, blimp-2, blimp-3). Start configurations are marked by "S". The blimp has to pass between the walls and through the hole(s), respectively.

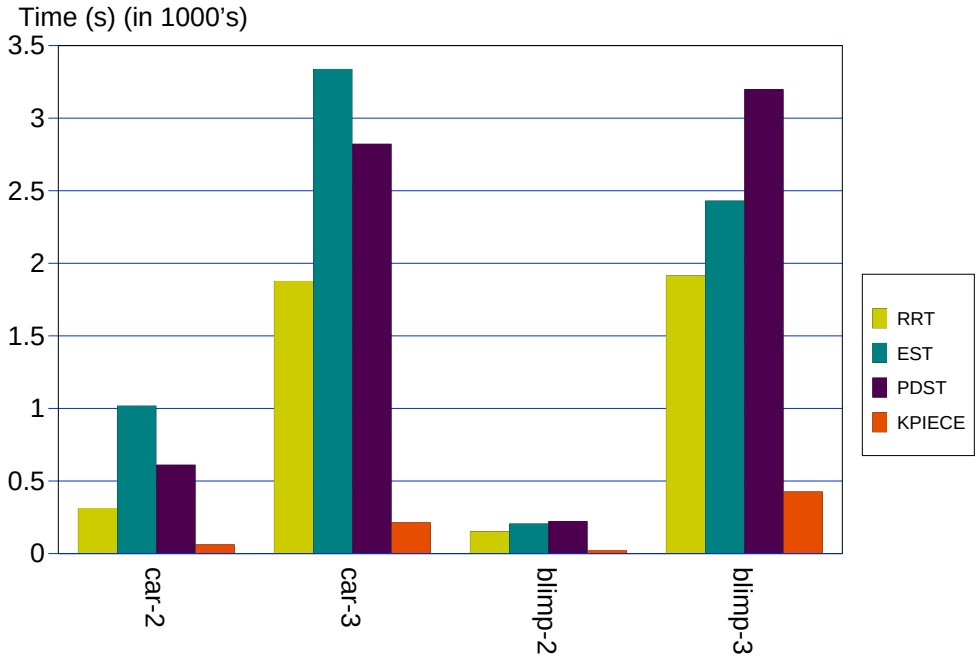

Figure 4.4: Runtimes of different algorithms on the car and blimp models. The achieved speedup is shown in Table 4.1.

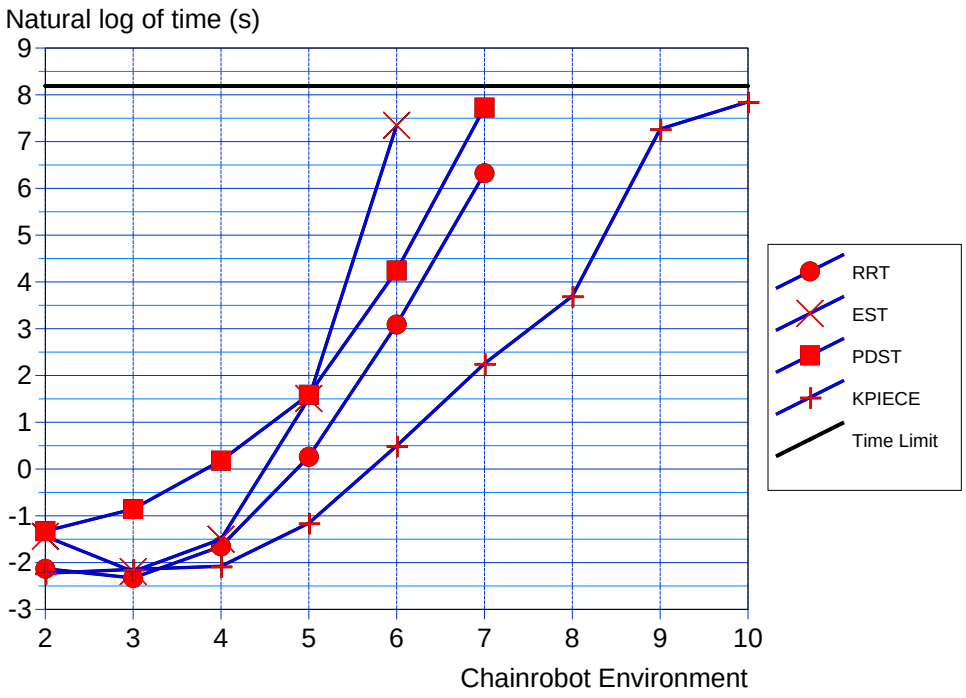

Figure 4.5: Runtimes of different algorithms on the modular robot model with no obstacles, using varying number of modules. The achieved speedup is shown in Table 4.1.

the dimensionality of the problem increases, `KPIECE` does better and speedups up to two orders of magnitude are observed. For simple problems however, other algorithms can be faster (*e.g.*, `RRT` for chain1-3). Using biasing for `RRT` (column marked by $\texttt{RRT}_b$) does not help very much for the presented environments. In fact, biasing slows down `RRT` most of the time since there are obstacles blocking direct paths to the goal. Adding biasing for `KPIECE` does however often improve the runtime, as shown in Table 4.2. The fact that learning is also present limits the negative effects biasing

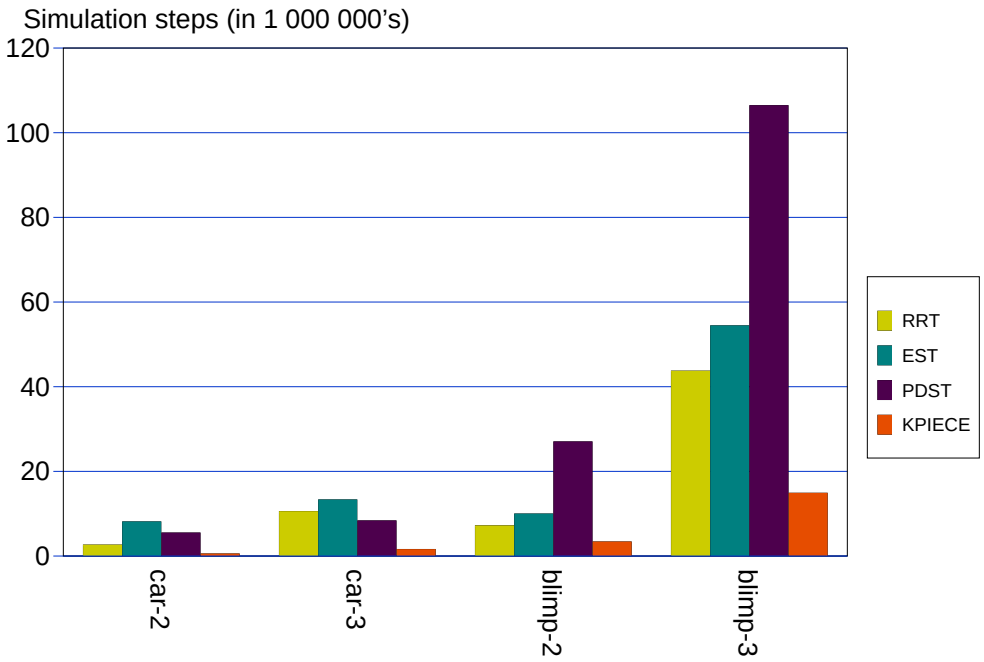

Figure 4.6: Number of simulation steps for different algorithms on the car and blimp models. Notice the similarity to Figure 4.4. This similarity serves to prove that the runtime of sampling-based planning algorithms is dominated by physics simulation, so minimizing the number of simulation steps leads to speedup.

has in certain cases.

The presented speedup values are consistent with the time spent performing simulations, as shown in Figure 4.6, which serves to prove that the computational improvements are obtained by minimizing the usage of the physics-based simulator. Since physics simulation takes up around 90% of the execution time, computational gain will be observed in the case of purely geometric planning as well, where simula-

Table 4.1: Speedup achieved by KPIECE over other algorithms for four different problems. If one of the other algorithms was unable to solve the problem in at least 10% of the cases, "—" is reported. KPIECE was configured with an automatically computed one-level discretization, as described in Section 3.4.

Problem	RRT	RRT_b	EST	PDST
chain1-2	1.1	3.5	2.2	2.5
chain1-3	0.8	2.1	1.0	3.6
chain1-4	1.5	3.9	1.8	9.6
chain1-5	4.1	3.7	14.4	15.6
chain1-6	13.4	9.6	946.8	42.5
chain1-7	58.5	196.3	—	238.1
chain1-8	—	—	—	—

Problem	RRT	RRT_b	EST	PDST
chain2-5	18.3	23.4	—	13.0
chain2-6	35.0	255.7	—	23.0
chain2-7	45.7	124.7	—	81.3
chain2-8	—	—	—	5.9
chain2-9	—	—	—	—

Problem	RRT	RRT_b	EST	PDST
car-1	3.2	3.1	27.7	7.9
car-2	5.0	3.5	16.1	9.7
car-3	8.7	14.8	15.5	13.1
blimp-1	1.6	2.2	3.1	3.3
blimp-2	6.4	7.2	8.7	9.4
blimp-3	4.5	7.3	5.7	7.5

tion is replaced by collision detection. With less time spent performing simulations, there are fewer samples to store in the built tree. This leads to savings in memory consumption, as indicated in Figure 4.7.

Table 4.2: Speedup achieved by KPIECE_b relative to KPIECE, both using the automatically computed one-level discretization.

chain1-2:	1.1	chain1-7:	0.9	chain2-5:	0.9	car-1:	1.3	blimp-1:	1.5
chain1-3:	1.0	chain1-8:	1.0	chain2-6:	0.9	car-2:	1.1	blimp-2:	1.2
chain1-4:	0.8	chain1-9:	3.9	chain2-7:	0.9	car-3:	1.3	blimp-3:	0.8
chain1-5:	1.1			chain2-8:	1.5				
chain1-6:	0.9			chain2-9:	1.0				

While the results shown in Table 4.1 are computed with a one-level discretiza-

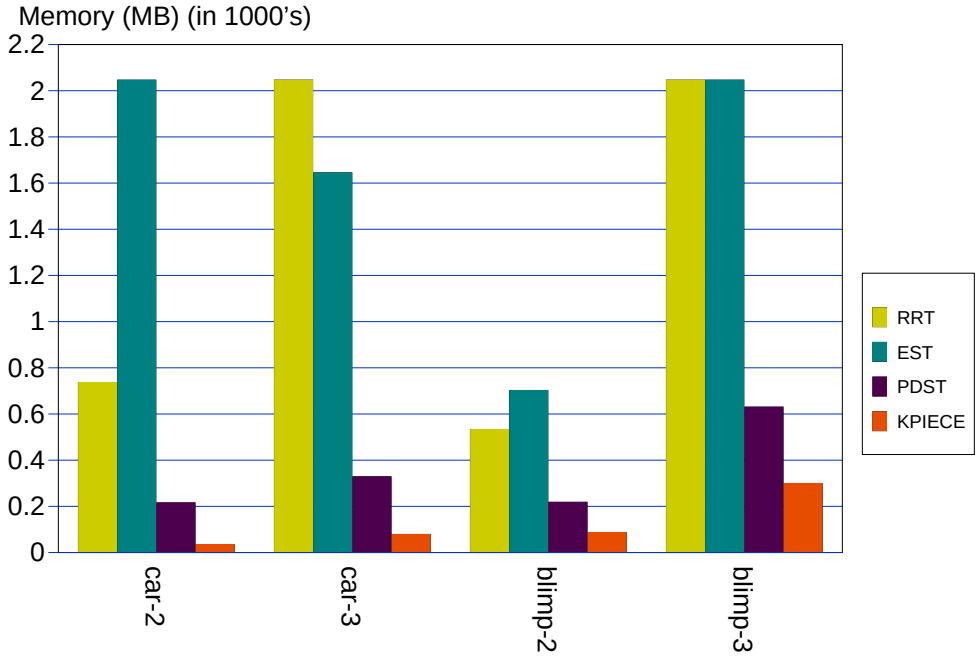

Figure 4.7: Memory usage for different algorithms on the car and blimp models. Notice the similarity to Figure 4.6. Since for every simulation step a new state is produced and the number of motions in the built tree is directly proportional to the number of states, the fewer simulation steps there are, the fewer motions will need to be stored in memory.

Table 4.3: Speedup achieved by `KPIECE` when using a two-level discretization relative to the automatically computed one-level discretization. For chain1-10 and chain2-10, a solution was found only with the two-level discretization so no speedup is reported.

chain1-2:	1.0	chain1-7:	1.9	chain2-5:	0.8	car-1:	1.3	blimp-1:	2.1
chain1-3:	1.1	chain1-8:	1.9	chain2-6:	0.9	car-2:	1.0	blimp-2:	1.4
chain1-4:	0.7	chain1-9:	5.0	chain2-7:	1.5	car-3:	0.9	blimp-3:	1.4
chain1-5:	0.8			chain2-8:	0.7				
chain1-6:	2.2			chain2-9:	1.2				

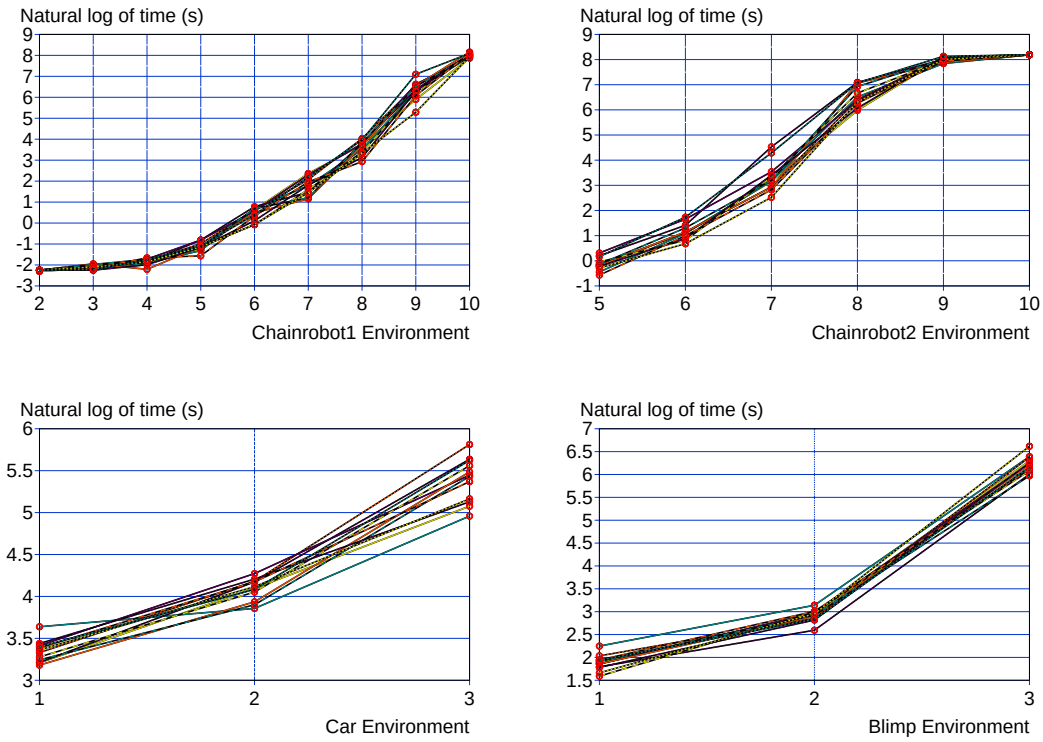

Figure 4.8: Logarithmic runtimes with twelve different discretizations for the chain1, chain2, car, and blimp.

tion, for some problems, better results can be obtained using multiple levels of discretization. To show this, for each robot, twelve discretizations are defined. First, a one-level discretization (consists only of $\mathcal{L}_1$) is computed as discussed in Section 3.4. Two more one-level discretizations with half and double the cell volume of the computed discretization's cells are then constructed (cell sides shortened and lengthened proportionally, in each dimension). For each of these three one-level discretizations, three more two-level discretizations (consist of $\mathcal{L}_1$, $\mathcal{L}_2$) are defined: ones that have the same $\mathcal{L}_1$, but $\mathcal{L}_2$ consists of cells with sizes of 10, 15, and 20 times the cell sizes

of $\mathcal{L}_1$. Table 4.3 shows the speedup obtained when employing the best of the nine defined two-level discretizations. As we can see, in most cases there are benefits to using two discretization levels. Experiments with more than two levels of discretization were conducted as well, but the performance started to decrease and the results are not presented here. The defined discretizations can also be used to evaluate the sensitivity of `KPIECE` to the defined grid sizes. As shown in Figure 4.8, the runtimes of the algorithm for the different discretizations are relatively close to one another (within a factor of 2.3). This implies that the algorithm is not overly sensitive to the defined discretization and thus approximating good cell sizes is sufficient. Nevertheless, finding a good number of levels of discretization and good cell sizes remains an open problem.

4.3 Discussion on Algorithm Components

In the previous section we have shown the computational benefits of using `KPIECE` over other algorithms. There are a few key details that make `KPIECE` distinct: the sampling of a chain of cells, the grouping of cells into interior and exterior, and the progress evaluation, based on increase in coverage. While the sampling of cell chains is an inherent part of the algorithm, the other two features can be easily disabled. This allows us to evaluate the contribution of these components individually.

Figure 4.9 shows that both progress evaluation and cell distinction contribute to reducing the runtime of `KPIECE`. While these components do not seem to help for easier problems (blimp-1), their contribution is important for harder problems

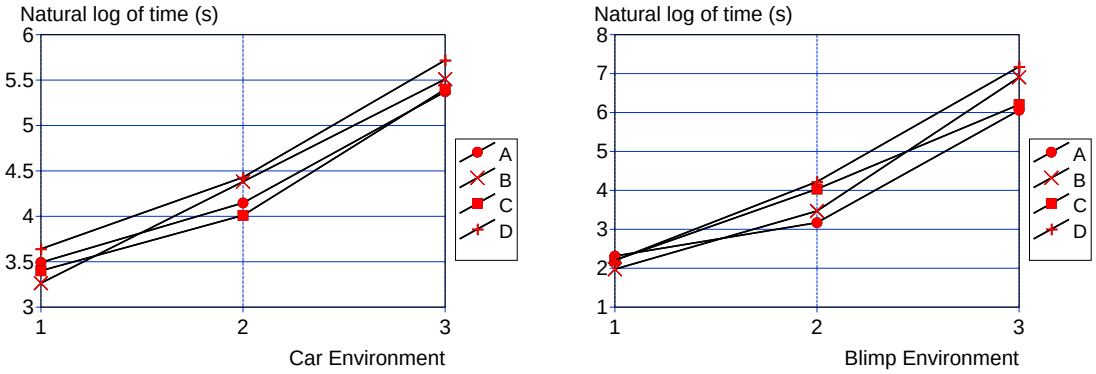

Figure 4.9: Logarithmic runtime for `KPIECE` with various components disabled, on 2-dimensional and 3-dimensional projections (car and blimp) with the automatically computed one-level discretization. `A` = no components disabled, `B` = no cell distinction, `C` = no progress evaluation, `D` = no cell distinction and no progress evaluation.

(car-3, blimp-3). In particular, the cell distinction seems to be the more important component as the problems get harder. This is to be expected, since the distinction allows the algorithm to focus exploration on the boundary of the explored space, while ignoring the larger, already explored interior volume.

4.4 Parallel Implementation

The presented algorithm was also implemented in a shared memory parallel framework. While previous work has shown significant improvements with embarrassingly parallel setups [56, 57], this work attempts to take the emerging multi-core technology into account and use it as an advantage. Instead of running the algorithm multiple times and stopping when one of the active instances found a solution as in [56, 57], `KPIECE` uses multiple threads to build the same tree of motions (threads

can continue expanding from cells instantiated by other threads). Synchronization points are used to ensure correct order of execution. This execution format will become more important in the next few years as the number of computing cores and memory bandwidth increase. Since each computing thread starts from a different random seed, the chances of all seeds being unfavourable decrease. If a single thread finds a path through a narrow passage, the rest of the threads will immediately use this information as well. This setup also seems to reduce the variance in the average runtime of the algorithm. It is important to note this proposed parallelization scheme can be applied to other sampling-based algorithms as well.

Table 4.4: Speedup achieved by `KPIECE` with multiple threads for 2-dimensional and 3-dimensional projections (car and blimp). `KPIECE` was configured with an automatically computed one-level discretization, as described in Section 3.4.

Threads	car-1	car-2	car-3	blimp-1	blimp-2	blimp-3
2	1.7	2.0	2.6	2.3	1.9	1.4
3	2.8	2.7	3.0	2.9	3.0	2.2
4	3.9	3.6	4.4	3.5	3.2	3.1

Table 4.5: Speedup achieved by `KPIECE` in embarrassingly parallel mode.

Threads	car-1	car-2	car-3	blimp-1	blimp-2	blimp-3
2	1.3	1.5	1.6	1.5	1.6	1.3
3	1.5	1.8	1.8	1.8	1.9	1.4
4	1.7	2.1	2.0	2.2	3.0	1.5

All experiments presented in previous chapters were conducted when using the planner in single-threaded mode. Table 4.4 shows the speedup achieved by the motion planner when using one to four threads on a four-core machine. The achieved speedup is super-linear in some cases, a known characteristic of sampling-based mo-

tion planners. When comparing to the speedup obtained with an embarrassingly parallel setup, shown in Table 4.5, we notice that lower runtimes are obtained with the shared memory setup. In addition, total memory requirements in the suggested setup do not increase significantly as the number of processors is increased.

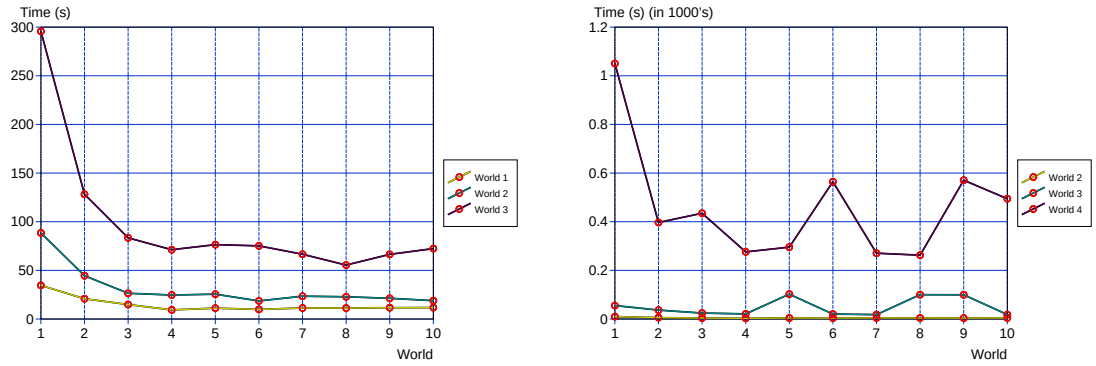

Figure 4.10: Runtime with different number of threads on a four-core machine, for 2-dimensional and 3-dimensional projections (car and blimp).

Due to the fact that using multiple random seeds increases the chances of the motion planner benefiting from at least one favourable seed, using more threads than available processing units may reduce runtime as well. To evaluate this possibility, we run `KPIECE` with up to 10 threads on a four-core machine (shown in Figure 4.10). Once we use more threads than number of cores, speedup decreases drastically, but does not come to a complete halt. Based on the conducted experiments, it seems to be the case that using a number of threads up to double the number of cores provides small benefits.

Chapter 5

Conclusions and Future Work

We have presented KPIECE, a sampling-based motion planning algorithm designed for high dimensional systems with complex dynamics, where physics-based simulation is needed. The algorithm is general, provides significant computational improvements and has a reduced memory footprint, when compared to previous work. All these features are essential for extending the range of solvable motion planning problems.

KPIECE does not need a distance metric or a way to sample states. It does however require a projection of the state space and the specification of a discretization. At this point it is recommended that the projection is defined by the user. As shown in the experiments, even simple intuitive projections work for complex problems. Initial experiments were conducted using Principal Component Analysis (PCA) to automatically extract projections that can be used for estimating the coverage instead of using the complete state space. The results are promising and it is likely this step of the problem can be made automatic. However, this is beyond the scope of this

thesis and is left for future work.

The discretization is an additional requirement when compared to other state-of-the-art algorithms. The algorithm's performance is not drastically affected by the discretization and a method to automatically compute one-level discretizations was presented. When using an automatically computed one-level discretization, `KPIECE` was compared to other popular algorithms, and shown to provide significant computational speedup. In addition, the provided shared memory parallel implementation seems to give better results than the embarrassingly parallel setup.

`KPIECE` is the result of a combination of ideas. Some of these ideas are new, some are inspired by previous work. In previous work, we have encountered state [32, 35] and motion sampling [51]; `KPIECE` takes this further and uses cell chain sampling. We have also seen progress evaluation [49], deterministic sample selection [51], use of physics-based simulation [50], and use of additional data-structures for estimation of coverage [37, 49, 51]. `KPIECE` implements variants of these ideas, combined with new ideas like distinction between interior and exterior cells, to obtain an algorithm that works well in a parallel framework. The result is a more accurate and efficient method that can solve problems previous methods could not.

It is conjectured that `KPIECE` is probabilistically complete: in a bounded state space, the number of cells is finite. Since with every selection, the importance of a cell can only decrease, every cell will be selected infinitely many times during the course of an infinite run. Every motion in a cell has positive probability of being selected, which makes the number of selections of each motion in the tree of motions be infinite as well. By the completeness of `PDST` [50], `KPIECE` is likely to be probabilistically complete.

A formal proof is left for future work.

Further work is needed for better automatic computation of the employed discretization. Automatic computation of the used state space projection would be beneficial as well, not only for `KPIECE`, but for other algorithms that require such a projection. Furthermore, it would be interesting to push the limits of this method to harder problems.

Bibliography

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

[2] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, June 2005.

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

[4] J. T. Schwartz and M. Sharir, "On the piano movers' problem: General techniques for computing topological properties of real algebraic manifolds," *Communications on Pure and Applied Mathematics*, vol. 36, pp. 345–398, 1983.

[5] I. A. Şucan, J. F. Kruse, M. Yim, and L. E. Kavraki, "Kinodynamic motion planning with hardware demonstrations," in *Intl. Conf. on Intelligent Robots and Systems*, September 2008, pp. 1661–1666.

[6] R. Gayle, A. Sud, M. Lin, and D. Manocha, "Reactive deformation roadmaps: motion planning of multiple robots in dynamic environments," in *Intl. Conf. on Intelligent Robots and Systems*, 2007, pp. 3777–3783.

[7] J.-C. Latombe, "Motion planning: A journey of robots, molecules, digital actors, and other artifacts," *Intl. Journal of Robotics Research*, vol. 18, no. 11, pp. 1119–1128, 1999.

[8] S. Thomas, X. Tang, L. Tapia, and N. M. Amato, "Simulating protein motions with rigidity analysis." *Journal of Computational Biology*, vol. 14, no. 6, pp. 839–855, 2007.

[9] M. S. Apaydin, D. L. Brutlag, C. Guestrin, D. Hsu, J.-C. Latombe, and C. Varma, "Stochastic roadmap simulation: an efficient representation and algorithm for analyzing molecular motion." *Journal of Computational Biology*, vol. 10, no. 3-4, pp. 257–281, 2003.

[10] H. J. Reif, "Complexity of the mover's problem and generalizations," in *IEEE Symposium on Foundations of Computer Science*, 1979.

[11] J. Canny, "Some algebraic and geometric computations in pspace," in *Annual ACM Symposium on Theory of Computing*. Chicago, Illinois, United States: ACM Press, 1988, pp. 460–469.

[12] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM*, vol. 40, no. 5, pp. 1048–1066, 1993.

[13] P. Cheng, G. Pappas, and V. Kumar, "Decidability of motion planning with differential constraints," in *IEEE Intl. Conf. on Robotics and Automation*, 2007, pp. 1826–1831.

[14] J. H. Reif and S. Slee, "Optimal kinodynamic motion planning for self-reconfigurable robots between arbitrary 2d configurations," in *Robotics: Science and Systems*, Atlanta, GA, June 2007.

[15] S. Lindemann and S. M. LaValle, "Current issues in sampling-based motion planning," in *Robotics Research: The Eleventh International Symposium*. Berlin: Springer-Verlag, 2005, pp. 36–54.

[16] S. Carpin, "Randomized motion planning - a tutorial," *Intl. Journal of Robotics and Automation*, vol. 21, no. 3, pp. 184–196, 2006.

[17] K. I. Tsianos, I. A. Şucan, and L. E. Kavraki, "Sampling-based robot motion planning: Towards realistic applications." *Computer Science Review*, vol. 1, no. 1, pp. 2–11, August 2007.

[18] R. Smith, "Open dynamics engine," http://www.ode.org.

[19] J. T. Schwartz and M. Sharir, "On the piano movers' problem: Iii. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers," *Intl. Journal of Robotics Research*, vol. 2, no. 3, pp. 46–75, 1983. [Online]. Available: http://ijr.sagepub.com/cgi/content/abstract/2/3/46

[20] M. Sharir and E. Ariel-Sheffi, "On the piano movers' problem: Iv. various decomposable two-dimensional motion-planning problems," *Communications on Pure and Applied Mathematics*, vol. 37, no. 4, pp. 479–493, 1983.

[21] J. Barraquand and J.-C. Latombe, "Robot motion planning : A distributed representation approach," *Intl. Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, 1991.

[22] L. E. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan, "Randomized query processing in robot path planning," *Journal of Computer and System Sciences*, vol. 57, no. 1, pp. 50–60, 1998.

[23] A. Ladd and L. Kavraki, "Measure theoretic analysis of probabilistic path planning," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 229–242, 2004.

[24] S. G. Ming C. Lin, "Collision detection between geometric models: A survey," in *IMA Conference on Mathematics of Surfaces*, 1998.

[25] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, August 1996.

[26] J. Barraquand, L. E. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan, "A random sampling scheme for robot path planning," *Intl. Journal of Robotics Research*, vol. 16, no. 6, pp. 759–774, 1997.

[27] L. Kavraki, M. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 166–171, Feb. 1998.

[28] F. Lamiraux, E. Ferr, and E. Vallee, "Kinodynamic motion planning: Connecting exploration trees using trajectory optimization methods," in *IEEE Intl. Conf. on Robotics and Automation*, vol. 4, April 2004, pp. 3987–3992.

[29] P. Cheng, E. Frazzoli, and S. LaValle, "Improving the performance of sampling-based planners by using a symmetry-exploiting gap reduction algorithm," in *IEEE Intl. Conf. on Robotics and Automation*, vol. 5, April 2004, pp. 4362–4368.

[30] C. Nissoux, T. Simeon, and J.-P. Laumond, "Visibility based probabilistic roadmaps," in *Intl. Conf. on Intelligent Robots and Systems*, vol. 3, 17–21 Oct. 1999, pp. 1316–1321.

[31] R. Bohlin and L. Kavraki, "Path planning using lazy prm," in *IEEE Intl. Conf. on Robotics and Automation*, vol. 1, April 2000, pp. 521–528.

[32] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE Intl. Conf. on Robotics and Automation*, 2000.

[33] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Intl. Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[34] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Intl. Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, March 2002.

[35] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *IEEE Intl. Conf. on Robotics and Automation*, vol. 3, April 1997, pp. 2719–2726.

[36] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Motion planning for humanoid robots," in *In Proc. 20th Int'l Symp. Robotics Research (ISRR'03)*, 2003.

[37] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," *Intl. Journal of Robotics Research*, pp. 403–407, 2003.

[38] E. Ferre and J.-P. Laumond, "An iterative diffusion algorithm for part disassembly," in *IEEE Intl. Conf. on Robotics and Automation*, New Orleans, USA, 2004, pp. 3149–3154.

[39] E. Plaku, K. Bekris, B. Chen, A. Ladd, and L. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Transactions on Robotics and Automation*, vol. 21, no. 4, pp. 597–608, Aug. 2005.

[40] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato, "A machine learning approach for feature-sensitive motion planning," Texas A&M University, Tech. Rep., 2004.

[41] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato, "Resampl: A region-sensitive adaptive motion planner," in *Intl. Workshop on the Algorithmic Foundations of Robotics*, New York City, July 2006.

[42] L. Jaillet, A. Yershova, S. M. LaValle, and T. Siméon, "Adaptive tuning of the sampling domain for dynamic-domain rrts," in *Intl. Conf. on Intelligent Robots and Systems*, 2005.

[43] A. Yershova, L. Jaillet, T. Siméon, and S. LaValle, "Dynamic-domain rrts: Efficient exploration by controlling the sampling domain," in *IEEE Intl. Conf. on Robotics and Automation*, Barcelona, Spain, 2005.

[44] A. M. Ladd and L. E. Kavraki, "Motion planning in the presence of drift, underactuation and discrete system changes," in *Robotics: Science and Systems*, Boston, MA, June 2005, pp. 233–241.

[45] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *IEEE Intl. Conf. on Robotics and Automation*, 2006.

[46] J. P. v. d. Berg and M. H. Overmars, "Path planning in repetitive environments," in *Methods and Models in Automation and Robotics*, 2006, pp. 657–662.

[47] B. Burns and O. Brock, "Single-query motion planning with utility-guided random trees," in *IEEE Intl. Conf. on Robotics and Automation*, Rome, Italy, 2007.

[48] K. E. Bekris and L. E. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in *IEEE Intl. Conf. on Robotics and Automation*, 2007, pp. 704–710.

[49] E. Plaku, M. Y. Vardi, and L. E. Kavraki, "Discrete search leading continuous exploration for kinodynamic motion planning," in *Robotics: Science and Systems*, W. Burgard, O. Brock, and C. Stachniss, Eds. Atlanta, Georgia: MIT Press, June 2007, pp. 313–320.

[50] A. M. Ladd, "Direct motion planning over simulation of rigid body dynamics with contact," Ph.D. dissertation, Rice University, Houston, Texas, December 2006.

[51] A. M. Ladd and L. E. Kavraki, "Fast tree-based exploration of state space for robots with dynamics," in *Algorithmic Foundations of Robotics VI.* Springer, STAR 17, 2005, pp. 297–312.

[52] Wikipedia, "Phisics engine," http://en.wikipedia.org/wiki/Physics_engine.

[53] I. A. Şucan, J. F. Kruse, M. Yim, and L. E. Kavraki, "Reconfiguration for modular robots using kinodynamic motion planning," in *ASME Dynamic Systems and Control Conference*, Michigan, Ann Arbor, October 2008.

[54] E. Plaku, K. E. Bekris, and L. E. Kavraki, "OOPS for Motion Planning: An Online Open-source Programming System," in *IEEE Intl. Conf. on Robotics and Automation*, Rome, Italy, 2007, pp. 3711–3716.

[55] J. Sastra, S. Chitta, and M. Yim, "Dynamic rolling for a modular loop robot," *Intl. Journal of Robotics Research*, vol. 39, pp. 421–430, January 2008.

[56] N. M. Amato and L. K. Dale, "Probabilistic roadmap methods are embarrassingly parallel," in *IEEE Intl. Conf. on Robotics and Automation*, Detroit, Michigan, USA, May 1999, pp. 688–694.

[57] S. Caselli and M. Reggiani, "Randomized motion planning on parallel and distributed architectures," *Proceedings of the Seventh Euromicro Workshop on Parallel and Distributed Processing*, pp. 297–304, February 1999.