A Sampling-Based Tree Planner for Systems with Complex Dynamics

Ioan A. Şucan and Lydia E. Kavraki

***Abstract*—This paper presents a kinodynamic motion planner, Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (`KPIECE`), specifically designed for systems with complex dynamics, where integration backward in time is not possible and speed of computation is important. A 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. An important characteristic of this discretization is that it 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 that `KPIECE` provides significant computational gain over existing state-of-the-art methods and allows solving some harder, previously unsolvable problems. For some problems `KPIECE` is shown to be up to two orders of magnitude faster than existing methods and use up to forty times less memory. 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.**

***Index Terms*—sampling-based motion planning, kinodynamic planning, planning with differential constraints, physics simulation**

I. 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 region while respecting a set of constraints [1]–[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 with kinematic and dynamic constraints (e.g., [5]). Although the motion planning problem stemmed from artificial intelligence and robotics, its applications have expanded to other domains such as graphics, computational biology and verification [6]–[9]. Exploration robots, tour guides, surgical robots, digital actors and folding proteins are only a few examples of cases where motion planning is used.

An important class of algorithms that can solve the motion planning problem is that of sampling-based tree planners [2], [3], [10]–[12]. These are algorithms that explore the state space of the robotic system by growing a tree of valid motions from the start state of the system towards a goal region, using a model of motion.

This work focuses on the problem of motion planning under differential constraints for complex, physical systems. Our interest is in general algorithms applicable to a variety of systems, without relying on particular properties of systems. Tree planners that address this problem have been previously introduced (e.g., [13]–[18]). However, better planners are needed as systems become more complex and there is a need to account for friction, mass, inertia, etc. Many complex systems of practical interest can be modeled by simulation of rigid body dynamics [19]. This renders tree planners that make use of bi-directional search or lazy collision checking [12] inapplicable (for reasons later explained in Section II-C). To quickly compute motion plans for systems with complex dynamics, two approaches can be followed: *(1)* ignore the complexities of the system and only compute geometric paths (sequences of states), in the hope that a controller can follow the paths by keeping velocities sufficiently low (e.g., [20]); *(2)* improve the exploration capabilities of the tree planner through means that only depend on forward propagating the model of motion – numerically evaluating motions only forward in time. While the first approach allows for the implementation of algorithms that can make use of many techniques for speeding up the planning process [12], including bi-directional search and lazy collision checking, the execution of rapid motions or of motions that must account for payload, friction, etc., cannot be correctly planned. In consequence, the latter approach is followed here.

This paper proposes `KPIECE` (Kinodynamic Planning by Interior-Exterior Cell Exploration), a new tree sampling-based motion planning algorithm, specifically designed for systems with complex dynamics. `KPIECE` achieves significant computational advantages over existing algorithms by combining novel ideas with ones that have been shown to perform well in previous work [12]. `KPIECE` is innovative in the sense that while it is able to handle high dimensional systems with complex dynamics, it reduces both runtime and memory requirements by making better use of information collected during the planning process. Intuitively, this information is used to decrease the amount of forward propagation the algorithm needs. Our experience has shown that it is typical for sampling-based planners to spend more than 90% of their computation performing forward propagation. For this reason, significant computational improvement over previous methods is obtained. Furthermore, with decreased amount of forward propagation, fewer states are generated in the tree of motions, which leads to memory savings. Conducted experiments show that up to two orders of magnitude speedup can be obtained when comparing to previous work. This speedup is accompanied by significantly reduced memory consumption: up to a factor of 40 reduction. Furthermore,

I.A. Şucan and L.E. Kavraki are with the Department of Computer Science, Rice University, Houston, TX, 77005, USA. E-mail: {isucan, kavraki}@rice.edu.

tackling more complex problems, ones that could not be previously addressed, becomes possible.

Contributions. The main contribution of this work is the exploration strategy of KPIECE. The core components of this strategy are: *(1)* using projections from the state space to a lower dimensional Euclidean space that can be discretized to help in guiding the exploration, *(2)* keeping track of the boundary of the explored space efficiently, so exploration can be further biased towards unexplored regions, *(3)* combining collected exploration information so that regions to be explored further can be deterministically selected and *(4)* evaluating progress of exploration. As discussed in Section II, the exploration strategy of tree planners is an issue that has attracted a lot of attention (e.g., [13]–[18], [21]–[32]). The presented strategy is particularly beneficial for problems with complex dynamics. The performance of KPIECE was tested on three different robots: a car moving in plane, a blimp moving in space and a chain modular robot. Propagation forward in time for the models of these three robots was computed using physics-based simulation of rigid body dynamics [19]. To the authors' knowledge, simulation of rigid body dynamics is a viable alternative at this time, in terms of accuracy of modeling (e.g., [33]). As mentioned above, the results of computing motion plans for these robots lead to significant computational improvements.

Since motion planning is usually part of a more complex task, it is generally desirable to have fast methods for the computation of motion plans. Although with core components alone KPIECE produces excellent results, a number of extensions that further improve performance are possible. These include using multiple resolutions of discretization when projecting to the lower dimensional Euclidean space, goal biasing, and using shared-memory parallelism (taking advantage of multiple processing cores). The combination of speedup that can be achieved and use of physics-based simulation could make KPIECE fast and accurate enough to be applicable in real-time motion planning for complex reactive robotic systems.

An initial version of this work appeared in [34]. This paper presents a more developed version of the algorithm, and significantly more experiments. In particular, the updated algorithm incorporates use of random projections [35] as a fall-back mechanism in case a projection from the state space to a Euclidean space is not specified in the input. Goal biasing is introduced in a manner that makes use of the algorithm's underlying data representations. New experiments have been conducted. Comparisons to more existing algorithms are included. In addition, a bi-directional version of the algorithm is also developed and the performance of KPIECE is evaluated when planning solely with geometric constraints as well.

Organization of the paper. Section II describes background and related work, Section III describes the core of the proposed algorithm and Section IV presents corresponding experiments. Section V extends the algorithm to its general form and includes additional experiments. Conclusions and future work are in Section VI.

II. Background and Previous Work

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]. A number of complete algorithms were developed for various forms of the problem and it was eventually shown to be PSPACE-complete [36], [37]. The developed algorithms are computationally prohibitive and difficult to implement. Techniques such as cell decomposition methods and potential fields [1]–[3] were studied as well, but few were successful at solving problems where the state space is high dimensional [38].

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 [39]. However, for the simplified case of a point mass robot, a polynomial algorithm exists [40]; the reconfiguration of modular robots under kinodynamic constraints is possible in $\Theta(\sqrt{n})$ time under certain assumptions [41].

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 techniques with weak completeness guarantees. There are multiple directions of research that investigate such techniques. 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].

A. Problem Definition

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 a differentiable manifold representing the state space, $Q_{free} \subseteq Q$ is the valid part of 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 explicitly defined). 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) [42] physics-based simulator. Instead of providing a set of equations of motion to be integrated, a model of a robot and its environment need to be specified.

B. Sampling-based Motion Planning

Much of the recent progress in motion planning is attributed to the development of sampling-based algorithms [2], [3].

Several of these sampling-based planning algorithms are probabilistically complete [2], which means that if a solution exists, it will be eventually found.

One of the earliest and most influential algorithms in sampling-based motion planning was the Probabilistic RoadMap (PRM) [43]. This method provided a coherent framework for many earlier works that used sampling and opened new directions for research. Among these new directions, a notable development is that of *tree-based planners* [15], [16], [21], [23], and later efforts, such as [17], [18], [22], [24]–[32]. As the name suggests, tree-based planners grow a tree of motions in the state space of the robotic system. 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 appropriate for planning under differential constraints because implementations that only use forward propagation are possible. If backward propagation is available, more efficient bi-directional tree planners can also be used, but in this case the steering problem [44], [45] needs to be solved as well. There are two fundamental issues tree planners must consider in their expansion:

– *In which parts of the tree expansion should continue*: There are various ways to guide the tree expansion (e.g., [15]–[18], [21]–[23], [32], [46]). Rapidly-exploring Random Trees (RRT) expand from states closest to randomly produced states [16], [21], Expansive Space Trees (EST) and Single-query Bi-directional probabilistic roadmap planner with Lazy collision checking (SBL) attempt to detect less explored regions and expand from them [15], [23], [24]. A more recent development is the idea of a Path-Directed Subdivision Tree (PDST) [47]. PDST uses an adaptive subdivision of the state space and a deterministic priority scheme to guarantee coverage, avoiding the use of a metric.

– *How this expansion should continue*: RRT [16], [21] suggests a Voronoi bias, by expanding toward random states. However, this can become problematic when planning with differential constraints and controls that achieve specific states cannot be easily computed. Methods that discretize the control set in order to achieve better coverage and reduced planning time have been introduced as well [13], [48]. Existence of narrow passages that need to be crossed by valid solutions can significantly reduce the performance of planners. Techniques that improve sampling in narrow passages [49] or identify the direction of narrow passages [50] have also been developed. Recently, the idea of combining two layers of planning has been introduced (SyCLoP, presented as DSLX in earlier work) [46]. SyCLoP is a meta-planner that uses discrete paths (top layer) in a discretization of the workspace to guide the continuous tree exploration (bottom layer). The planner at the bottom layer can be chosen among many tree-based planners, including the one presented in this paper.

In this work, we focus on the first aspect of the tree expansion: deciding which parts of the tree merit further expansion. Developments addressing the second aspect of tree expansion carry over to the work presented here. KPIECE tries to push the current limits of planning under differential constraints. A new strategy for guiding the tree expansion, based on a discretization of the state space, is presented. More details about this follow in Section III.

C. Physics-Based Simulation

Physics-based simulation (of rigid body dynamics) can be performed with libraries known as physics-based simulators. These libraries approximate the dynamics of robots in their environments usually assuming all bodies are rigid, Coulomb models of friction and instantaneous impacts [19], [51].

Figure 1 shows the operation of a typical physics-based simulator. As in the case of numerical integration of motion models, 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-based simulation, this discretization interval typically does not vary and is referred to as the *simulation step*. When the system is at a particular state q_t, applying the control u_t takes the system to state q_{t+1}, within one simulation step. Simulating a system for some duration T requires taking a number of simulation steps. Large values for the simulation step will produce less accurate results while small values for the simulation step will require more computation time. Thus, the value of the simulation step affects both the quality of solution paths and the runtime of the motion planner. In this work we only use default values for the simulation step, as indicated by the developers of the physics simulator.

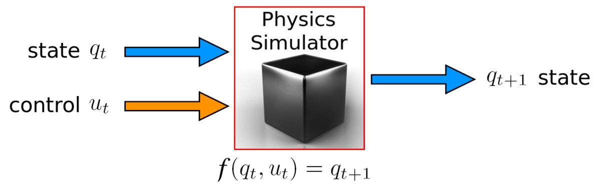


Fig. 1. The operation of a physics-based simulator: one simulation step.

One essential difference between integration of motion models and physics-based 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. This prevents planning with bi-directional tree planners, which are often computationally very efficient.

An additional difference is that physics-based simulation is significantly more computationally expensive than integration of motion models, as contacts need to be computed for every simulation step. The necessity of contact computation also makes lazy collision checking unwarranted. Even so, the benefits of simulation outweigh the costs: increased accuracy is available since physics-based 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 remains a problem for all known approaches.

This work uses physics-based simulation as a black box only and does not attempt to improve simulation algorithms.

Currently, many options exist when it comes to physics-based simulation libraries, both commercial and free. Among the better-known ones are `NVIDIA PhysX`, `ODE`, `Vortex`, `bullet`, etc. For the purposes of this work, the `ODE` (Open Dynamics Engine) library was used [42]. `KPIECE` is not tied to `ODE`, and other physics simulators could have been used instead. `ODE` was chosen because it is a piece of software widely used for simulating robotic systems (e.g., [52]).

III. Algorithm

`KPIECE` is a sampling-based algorithm that explores the state space of the robotic system by growing a tree of motions. Each motion in the tree is defined as $\nu = (s, u, t)$, where $s \in Q$ is the starting state of the motion and $u \in U$ is the control applied at that state, for a duration $t \in R^{\geq 0}$.

From a high-level perspective, `KPIECE` proceeds iteratively, as described in Algorithm 1: at each iteration, an existing motion from the tree is selected [line 3]; a new motion starting at a state along the selected motion is produced and added to the tree [line 4]; information gathered in the expansion process is incorporated for future selections of motions [line 7]; this process continues until some termination criterion is met.

The above description is intended to be solely an overview of `KPIECE`. The steps Algorithm 1 are common to many other sampling-based algorithms that use trees. What makes such algorithms different is how these steps are carried out. In the case of `KPIECE`, this accounts for up to two orders of magnitude speedup with respect to previous work.

Various aspects of `KPIECE` are discussed in the following sections. For clarity, we approach the description in an incremental fashion. A more detailed description of the core of the algorithm is given in Section III-E. Extensions to the algorithm are presented in Section V.

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

```
1: $T$ = InitializeTree($q_{start}$)
2: for $i \leftarrow 1...N_{iterations}$ do
3:     $\nu$ = SelectMotion($T$)
4:     ExpandTree($T, \nu$)
5:     if solution is found then
6:         return solution
7:     EvaluateProgress()
8: return no solution
```

A. The Tree of Motions

The tree is initialized with a motion $\nu_0 = (s, null, 0)$ that consists of the robot's starting state and a control that has no effect, applied for 0 duration. Although motions are in fact continuous segments, they are computed by a forward propagation function (as in Section II-A), with fixed step size. This means that intermediate states along each motion are generated at a fixed resolution. We call this resolution the *propagation step size*. The choice of fixing the propagation step size is made to avoid inconsistencies that may otherwise arise when using physics simulation. In that case, the propagation step size is the same as the simulation step defined in Section II-C. As a result, for every motion, the duration of the control is $t = m \cdot r$, where $r \in \mathbb{R}^+$ is the propagation step size and $m \in \mathbb{N}$ is the number of steps.

The controls applied from s are selected uniformly at random from U. The duration of the control is obtained by sampling a value for m. 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. Different methods of control selection are desirable for systems that are not stable for instance, as in this case random selection of controls will likely not lead to valid states. Using some generic forms of control such as LQR is also possible [53].

For a motion ν, let $States(\nu)$ be the set of states along the motion ν, at the propagation step size, as they are generated by forward propagation. $States(\nu)$ is not stored by `KPIECE`, but it is generated as needed. See Figure 2 for an example. New motions expanded from an existing motion ν can start at any state in $States(\nu)$. Let $AS = \bigcup_\nu States(\nu)$ be the set of all states that the tree of motions consists of, with respect to the used propagation step size. AS is not computed by `KPIECE`, but it is a notion we use to explain the execution of the algorithm.

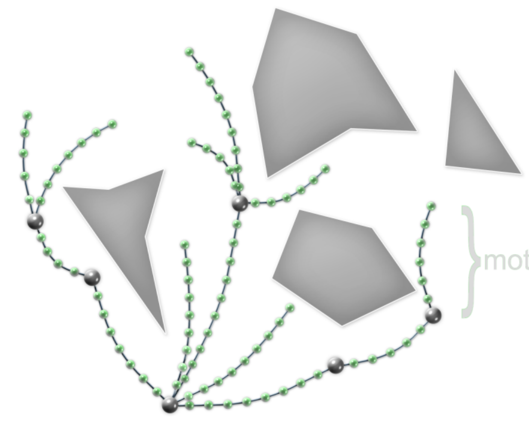


Fig. 2. Tree of motions as grown by `KPIECE`. The states at the start of motions are depicted as larger vertices. The motion is computed by forward integration at fixed step size. Intermediate states are depicted as smaller vertices. The intermediate states are not stored by `KPIECE`.

Not unlike other sampling-based planners that employ trees, `KPIECE` tries to reach the goal as quickly as possible, but also eventually explore entirely the reachable regions of the valid state space Q_{free}, so that a solution is found if one exists. In order to achieve this, `KPIECE` carefully selects motions for further expansion [line 3 in Algorithm 1]. An important part of the selection strategy is estimating the coverage of the state space that the tree of motions achieves.

B. Estimating State Space Coverage

As Q can be high dimensional and its topology is not known to the algorithm, we define the following continuous mapping, to help with the estimation of coverage:

$$Proj : Q \to \mathbb{R}^k.$$

We call $Proj$ a projection from Q to a Euclidean space. This is an input to the algorithm. $\mathbb{R}^k$ is the projection space and k is the dimension of the projection. We will first discuss how to use a projection, and later how to generate it.

Define $Coord : \mathbb{R}^k \to \mathbb{Z}^k$, where $\mathbb{Z}$ is the set of integers:

$$\begin{aligned} Coord(\mathbf{p}) &= Coord((p_1, \ldots, p_k)) \\ &= \left(\left\lfloor \frac{p_1 - o_1}{d_1} \right\rfloor, \ldots, \left\lfloor \frac{p_k - o_k}{d_k} \right\rfloor \right) = \mathbf{z}, \end{aligned}$$

where $\lfloor \cdot \rfloor$ denotes truncation to nearest smaller integer, $\mathbf{p} = (p_1, \ldots, p_k) \in \mathbb{R}^k$, $\mathbf{o} = (o_1, \ldots, o_k) \in \mathbb{R}^k$ is an arbitrary point designated as the origin, $d_i \in \mathbb{R}^+, i \in \{1, \ldots, k\}$ and $\mathbf{z} \in \mathbb{Z}^k$. $Coord$ discretizes $\mathbb{R}^k$ into k-dimensional cubes of uniform size, each with sides of lengths $d_1, \ldots, d_k$.

For every $\mathbf{z} \in \mathbb{Z}^k$, define the corresponding cell in Q to be:

$$Cell(\mathbf{z}) = \{q \in Q \mid Coord(Proj(q)) = \mathbf{z}\}.$$

Motions added to the tree of motions are said to be part of a cell if all their states are included in the cell:

$$Motions(\mathbf{z}) = \{\nu \mid q \in States(\nu) \text{ implies } q \in Cell(\mathbf{z})\}.$$

The invariant that every motion is part of a single cell is maintained. This is achieved by splitting motions before adding them to the tree of motions, such that they are not part of multiple cells. When a motion ν is to be added, $States(\nu)$ is generated. For every $q \in States(\nu)$, $Coord(Proj(q))$ is computed. With this information, it can be decided which parts of the motion go to which cells. Since $States(\nu)$ is an approximation of the motion ν, this computation is not exact, but it is sufficient for our purposes.

It is now possible to define the coverage achieved by a tree of motions in Q. For every cell coordinate $\mathbf{z} \in \mathbb{Z}^k$, the coverage of $Cell(\mathbf{z})$ is

$$Coverage(\mathbf{z}) = \sum_{\nu=(s,u,t) \in Motions(\mathbf{z})} \left(1 + \frac{t}{r}\right),$$

where r is the propagation step size. Since t is an integer multiple of r, the value of the coverage represents the number of states in $Cell(\mathbf{z})$ that are also in AS: $Coverage(\mathbf{z}) = |AS \cap Cell(\mathbf{z})|$, where $|\cdot|$ denotes the cardinality of a set.

At this point we make the assumption that coverage estimates for cells are relevant for the coverage of Q. We do not prove this is the case from a mathematical point of view, but experimental results shown later support this hypothesis.

As the tree of motions increases, and the number of states in AS increases, KPIECE keeps track of the minimal set of cells that covers AS. We say $C \subset \mathbb{Z}^k$ covers AS if $AS \subseteq \bigcup_{\mathbf{z} \in C} Cell(\mathbf{z})$. We say C is minimal if there is no subset $D \subsetneq C$ such that D covers AS. When the algorithm starts, AS has only one state. One cell is sufficient to cover AS – the cell that contains the starting state. Let $M_c \subset \mathbb{Z}^k$ denote the minimal cover of AS. Throughout the execution of the algorithm, the cardinality of M_c increases. Cells included in M_c are called *instantiated cells*. M_c is called a *discretization* of the space covered by the tree of motions. In Section V we discuss how to extend this discretization to multiple levels.

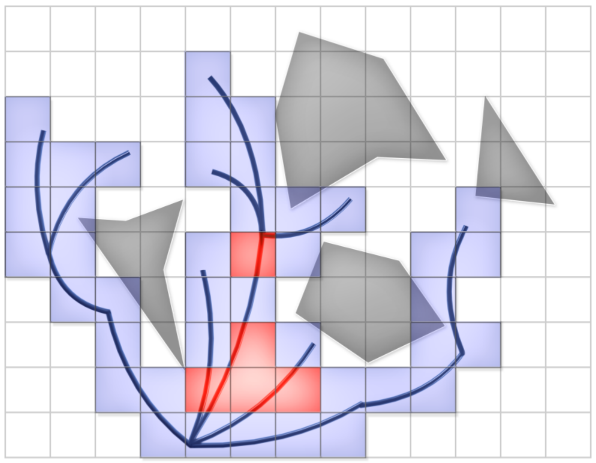


Fig. 3. Representation of a tree of motions and its minimal cover. Interior cells are differentiated from exterior cells.

C. Distinguishing Interior and Exterior Cells

For every $\mathbf{z} = (z_1, \ldots, z_k) \in \mathbb{Z}^k$, the neighbors of $Cell(\mathbf{z})$ are $Neighbors(\mathbf{z}) =$

$$\begin{aligned} \{\ Cell(\mathbf{w}) \in M_c \ \mid\ & \mathbf{w} = (z_1, \ldots, z_{i-1}, y, z_{i+1}, \ldots z_k), \\ & \text{for } y = z_i - 1 \text{ or } y = z_i + 1\ \}. \end{aligned}$$

The maximum cardinality of $Neighbors(\mathbf{z})$ is $2k$. A distinguishing feature of KPIECE is the notion of interior and exterior cells. A cell is considered exterior if it has less than $2k$ neighboring cells. Cells with $2k$ neighboring cells are considered interior. The reason for making this distinction is that focusing the exploration on exterior cells allows the motion planner to cover the state space faster. As the algorithm progresses and new cells are created, some exterior cells become interior (see Figure 3). When larger parts of the state space have been explored, most cells have become interior. However, for high dimensional spaces, to avoid having only exterior cells, the definition of interior cells can be relaxed and cells can be considered interior before the cardinality of the set of neighbors reaches $2k$. For the purposes of this work, this relaxation was not needed.

D. Importance of Cells

In Section III-E we show that KPIECE needs to first select a cell in order to find motions from which to continue the expansion of the exploration tree. This section describes the notion of *importance* associated to cells, a notion used in the selection of cells. The following pieces of information are considered when selecting a cell $Cell(\mathbf{z})$:

- The coverage $Coverage(\mathbf{z})$ (work in the same spirit suggested in e.g., [46]),
- The number of times $Cell(\mathbf{z})$ was previously selected (suggested in e.g., [23]),
- The cardinality of $Neighbors(\mathbf{z})$,
- The iteration at which $Cell(\mathbf{z})$ was added to M_c (suggested in e.g., [47]),

- A measure of the progress in exploration achieved when expanding from $Cell(\mathbf{z})$ (work in the same spirit suggested in e.g., [32]).

KPIECE prefers expanding from cells that are less covered rather than from cells that are well covered. Cells that have been selected for expansion fewer times are preferred over cells that have been selected many times. Cells that have fewer neighbors and cells that have been instantiated more recently are preferred, as these are more likely to be closer to unexplored areas of the space. Cells that have led to good progress in exploration are preferred over cells that have led to slower progress (e.g., if a cell contains many motions that place the robot in front of a wall, it is possible expanded motions will often hit the wall).

The considerations mentioned above for selecting cells are heuristics that have been shown to work well in practice. KPIECE combines their use in the notion of *importance*, since no one heuristic can be identified as better than the others. The importance of a cell $Cell(\mathbf{z})$ is defined as:

$$Importance(\mathbf{z}) = \frac{\log(\mathcal{I}) \cdot \texttt{score}}{\mathcal{S} \cdot (1 + |Neighbors(\mathbf{z})|) \cdot Coverage(\mathbf{z})},$$

where $\mathcal{I}$ is the number of the iteration at which $Cell(\mathbf{z})$ was added to M_c, $\mathcal{S}$ is the number of times $Cell(\mathbf{z})$ was selected for expansion (initialized to 1) and score reflects the exploration progress achieved when expanding from $Cell(\mathbf{z})$ (initialized to 1). The definition we propose for importance represents what worked well in our experiments. However, it is possible that other definitions can lead to better performance for certain applications. KPIECE prefers expanding from cells with higher importance. With careful bookkeeping, the importance of a cell can be computed in constant time, since all the values it depends on can be made readily available.

To make the definition of importance complete, the use of score needs to be explained. Adding a motion to the tree of motions may increase the coverage of the space. The update to score proceeds as follows:

- Assume a motion was selected for expansion from $Motions(\mathbf{z})$ (Algorithm 1, line 3).
- Let total coverage $C_{before} = \sum_{\mathbf{z} \in M_c} Coverage(\mathbf{z})$, and $T_{before} =$ current time.
- Algorithm 1 proceeds with lines 4 through 6.
- Let total coverage $C_{after} = \sum_{\mathbf{z} \in M_c} Coverage(\mathbf{z})$, and $T_{after} =$ current time.
- Line 7 of Algorithm 1 consists of the following steps:

$$\begin{aligned} P &= \alpha + \beta \cdot \left(\frac{C_{after} - C_{before}}{T_{after} - T_{before}} \right) \\ \texttt{score} &= \texttt{score} \cdot \min(P, 1), \end{aligned}$$

  for score corresponding to $Cell(\mathbf{z})$.

The purpose of score is to reflect how much progress has been made when expanding from $Cell(\mathbf{z})$. Based on the increase in total coverage and the time spent achieving this increase in coverage, a penalization value P is computed. P is used as a multiplicative factor for score. To avoid entering an infinite loop where the cell with highest importance is always the same, score must never be multiplied by a value larger than 1, hence the use of $\min(P, 1)$. α and β are implementation specific constants that help defining P. P is intended to be smaller than 1 for expansions that did not provide significant increase in coverage. If $P \geq 1$, the score is not be changed. If the coverage increase is 0, $P = \alpha$, so α must always be larger than 0 so that the score does not become 0. $\beta \in \mathbb{R}^+$ is chosen such that P ends up being larger than 1 only for expansions that have led to significant progress. More details on the selection of parameters introduced in this section follow in Section III-G.

E. KPIECE *Algorithm*

A more detailed description of KPIECE is given in Algorithm 2. The algorithm begins by initializing the tree of motions with a motion of 0 duration that consists solely of the robot's starting state [lines 1–3]. This motion is added to a special data structure called Grid. Grid associates $Motions(\mathbf{z})$ to every $\mathbf{z} \in M_c$ and takes care of the bookkeeping necessary to update the importance of cells as the algorithm is running.

In order to expand the tree of motions, KPIECE needs to select an existing motion from that tree. Grid is used to identify areas of the state space that are considered more important – using the notion of importance defined in Section III-D [line 5].

KPIECE randomly decides whether to expand from an interior or exterior cell from M_c. A strong bias towards exterior cells is usually employed (75% of the time, in this paper). This decision effectively separates M_c in two disjoint sets: $M_{c,int}$ and $M_{c,ext}$ (the set of interior cells and the set of exterior cells, respectively). Subsequently, KPIECE deterministically selects the cell $Cell(\mathbf{c})$ with maximum importance from either $M_{c,int}$ or $M_{c,ext}$. This operation can be performed quickly by maintaining $M_{c,int}$ and $M_{c,ext}$ as heaps. A motion ν from $Motions(\mathbf{c})$ is then picked according to a half-normal distribution. The half-normal distribution $h(\sigma^2)$ is used because motions that have been added more recently are preferred for expansion [line 6]. $h(\sigma^2)$ corresponds to the normal distribution (mean = 0 and variance = σ^2) folded about the y-axis at 0; it returns a value larger than 0, with a high probability of being close to 0. For a set $Motions(\mathbf{c})$ with m motions, numbered from 0 to $m-1$, where the 0^{th} motion is the most recently added one, a randomly selected motion ν is the $\lfloor h((m/3)^2) \rfloor^{th}$ motion. A state s along ν is then chosen uniformly at random from $States(\nu)$ [line 7]. Expanding the tree of motions continues from s [line 9]. Because $States(\nu)$ is not stored it may be necessary to recompute s, but the memory savings outweigh the computational costs.

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]. Information gained during the expansion step is incorporated in the score of the selected cell $\mathbf{c}$, as described in Section III-D [lines 14,15].

F. Selection of Projections

So far we have described how projections are used by KPIECE. How this projection is defined is left to the user. Our experience has shown that projections are easy to specify, Algorithm 2. KPIECE(q_{start}, $N_{iterations}$)

```
1: Let $\nu_0$ be the motion of duration 0 containing solely $q_{start}$
2: Create an empty Grid data-structure $G$
3: $G$.AddMotion($\nu_0$)
4: for $i \leftarrow 1...N_{iterations}$ do
5:     $\mathbf{c}$ = $G$.Select(0.75)
6:     Select $\nu \in Motions(\mathbf{c})$ using a half-normal distribution
7:     Select $s$ along $\nu$
8:     Sample random control $u \in U$ and simulation time $t \in \mathbb{R}^+$
9:     Check if any motion $(s, u, t_\circ)$, $t_\circ \in (0, t]$ is valid (forward propagation)
10:    if a motion is found then
11:        Construct the valid motion $\nu_\circ = (s, u, t_\circ)$ with $t_\circ$ maximal
12:        If $\nu_\circ$ reaches the goal region, return path to $\nu_\circ$
13:        $G$.AddMotion($\nu_\circ$)
14:    $P$ = $\alpha$ + $\beta$ · (ratio of increase in coverage to time spent in simulation)
15:    Multiply the score of $Cell(\mathbf{c})$ by min($P$,1)
16: return no solution
```

as KPIECE is robust to multiple projections for each of the problems considered. In Section IV-A we show some example projections used in our experiments. The purpose of the projection is to provide a space in which coverage is to be estimated, such that the space is representative for the problem being solved. For example, if we are planning for a car moving in plane, a representative space for estimating coverage is the plane in which the car is moving (2-dimensional projection). For a manipulator arm, the position in space of the tip of the arm is representative (3-dimensional projection). For systems where velocity is important, for example an inverted pendulum, a representative space is that of the velocity of the pendulum and its angle (2-dimensional projection). If the pendulum has multiple links, the projection can consist of the norm of angular velocities and the position of the tip of the pendulum in the plane of motion (3-dimensional projection).

Most of the time, finding an input projection can be very intuitive. As already mentioned, multiple different projections work well. For example, for a manipulator arm, the projection that only considers the first two angles of the arm (closest to base) also leads to good results. In some cases however, for example in the case of reconfigurable robots, defining a projection can be hard. In such cases, or when the user is simply unwilling to set a projection, random projections can be used as a fall-back. The inspiration to use random projections comes from a theorem by Johnson and Lindenstrauss [54] which states:

For any $\varepsilon > 0$, any n points from a l_2 metric can be embedded in a l_2 metric of dimension $O(\log n/\varepsilon^2)$, with $(1+\varepsilon)$ distortion.

This means that distances between states in the state space with the l_2 norm are approximately preserved in the projection space with the l_2 norm. Since l_2 norm is usually not an appropriate metric for the state space Q, we do not rely on the mathematical foundation provided by this theorem. Random projections have been empirically shown to work well in the context of estimating state space coverage: it was shown that finding a random projection that leads to good performance is easy for systems with state spaces of moderate dimension (up to 10 dimensions) [35].

In this paper, a random projection is sampled every time planning needs to be performed and a user-defined projection is not available. To generate a random projection from an n-dimensional state space Q to $\mathbb{R}^k$, k vectors in $\mathbb{R}^n$ are randomly sampled element by element according to a normal distribution (with mean 0 and variance 1). The vectors are made orthonormal, as suggested in [35]. For a state $x \in Q$ and a random linear projection $\mathbf{V}$,

$$\mathbf{V} = (\mathbf{v}_1, ..., \mathbf{v}_k),\ \ \mathbf{v}_i \in \mathbb{R}^n,$$

the projection of x is $p \in \mathbb{R}^k$, with

$$p = \mathbf{V}^T x,$$

assuming all vectors are column vectors.

G. Selection of Parameters

In the description of KPIECE a number of parameters have been introduced:

- The α and β parameters for progress evaluation,
- The sizes of cells in the discretization.

Parameters for progress evaluation. The value for α represents the penalization of a cell's score, as a multiplicative factor, if no progress is made from that cell. This suggests α should be lower than 1 but not close to 0. A value of $\alpha = 0.7$ worked well in our implementation and we believe this can be considered constant for other implementations as well. The value for β depends on how the coverage of the cell is computed. β is used to scale the increase in coverage so that it can be added to α. For our experiments $\beta = 5$ worked well.

Sizes of cells. The sizes of cells depend on the projection used. We have identified a set of guidelines for determining cell sizes. While KPIECE is running, it 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 even exist), there are ranges that 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 relatively small. In this work values between 1 and 4 were observed for well performing projections.
- 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 are good, too large or too small. This information is reported for each dimension of the projection 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. To start the iteration, an initial guess is needed. A simple way to compute an initial guess is to sample a number of states from the state space (e.g., 1000) and compute the bounding box for the corresponding projections. A simple initial guess would be to set the cell sizes to be 10% of the size of the bounding box, in each dimension.

IV. Experiments

In this section we present experimental results for different planning problem instances. We begin by describing the employed robot models and the experimental setup (Sections IV-A and IV-B). We then compare the implementation of `KPIECE` as presented thus far, using a discretization with automatically computed parameters as shown in Section III-G, against a number of existing algorithms (Section IV-C). Further experiments show the influence of using random projections (Section IV-D). Various components of `KPIECE` are then disabled to expose their importance in the overall algorithm (Section IV-E). The results of running `KPIECE` in a kinematic context are then presented (Section IV-F).

A. Robot Models

Three different robots were used in benchmarking `KPIECE`, 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.

a) Modular Robot: The model[1] characterizes the CKBot modules [55]. 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 [33]. 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. 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 state space for a chain modular robot with m modules is $Q = \{\mathbf{x} \mid \mathbf{x} = ((x_1, \dot{x}_1), ..., (x_m, \dot{x}_m))\}$, where x_i is the angle position of module i, $i \in \{1, .., m\}$. The employed projection was $\mathit{Proj} : Q \rightarrow \mathbb{R}^3$. In the evaluation of Proj, the first two dimensions are the (x, z) coordinates of the last module (x, z is the plane observed in Figure 4) 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]This model was created in collaboration with Mark Yim, GRASP Laboratory of Robotics Research and Education, `yim@grasp.upenn.edu`.

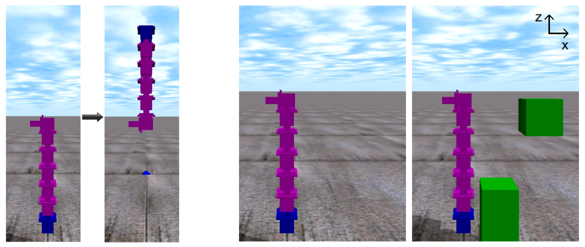


Fig. 4. Left: start and goal configurations. Right: environments used for the chain robot (7 modules). Experiments were conducted for 2 to 10 modules. In the case without obstacles, the environments are named chain1-*x* where *x* stands for the number of modules used in the chain. In the case with obstacles, the environments are named chain2-*x*.

b) 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). The desired velocities indicate what velocities the car is intended to achieve and 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 state space for this model is $Q = \{\mathbf{x} \mid \mathbf{x} = (x, y, \theta, v, \dot{\theta})\}$, where (x, y) denote the center of the car chassis, θ is the car's orientation and v is the velocity along the orientation. The employed projection was $\mathit{Proj} : Q \rightarrow \mathbb{R}^2$. Proj evaluates to the (x, y) coordinates of the center of the car body. The environments the system was tested in are shown in Figure 5.

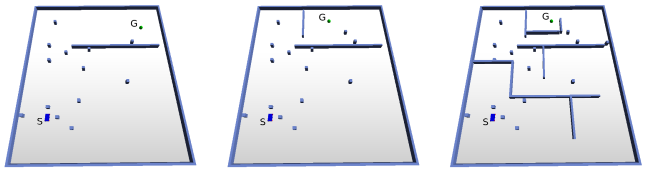


Fig. 5. Environments used for the car robot (car-1, car-2, car-3). Start and goal configurations are marked by "S" and "G". The small cubes represent obstacles.

c) Blimp Robot: The third robot that was tested was a blimp robot [17]. 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 with respect to the direction of forward motion. Since `ODE` does not include air friction, a Stokes model of drag was implemented for the blimp (the drag force is $F_{drag} = -b\mathbf{v}$ where $\mathbf{v}$ is the linear velocity of the blimp and b is the drag coefficient). The state space for this model is $Q = \{\mathbf{x} \mid \mathbf{x} = (x, y, z, \theta, v, \dot{z}, \dot{\theta})\}$, where (x, y, z) denote the center of the blimp, θ is the blimp's orientation and v is the forward velocity along the orientation. The employed projection was $\mathit{Proj} : Q \rightarrow \mathbb{R}^3$. Proj evaluates to the (x, y, z) coordinates of the center of the blimp. The environments the system was tested in are shown in Figure 6.

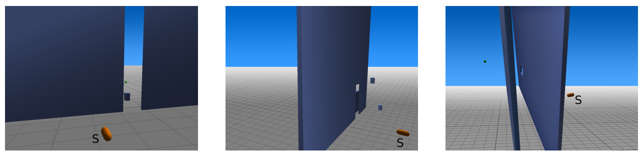


Fig. 6. Environments used for the blimp robot (blimp-1, blimp-2, blimp-3). Start configurations are marked by "S". The blimp has to pass between the walls and through the hole(s), respectively. The small cubes represent obstacles.

B. Experimental Setup and Compared Motion Planners

KPIECE was benchmarked against general well-known efficient algorithms (RRT, EST, PDST) with three different robotic systems, in different environments. All these algorithms are implemented using uni-directional exploration – single trees, since backward exploration is not possible with physics engines. For the implementations of RRT [16], EST [15] and PDST [47], the OOPSMP framework was used [56]. 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. 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. In addition, placing of intermediate states along motions as they were added in the exploration tree has also been attempted (this variant of the algorithm is marked as RRTi). For EST, the nodes to expand from were selected based on a grid subdivision of the state space, as this strategy has been shown to work well [24].

KPIECE was implemented by the authors. A projection was defined for each robot and the same projection was used for EST, PDST and KPIECE. In addition to the projection, KPIECE needs a discretization to be defined for each robot. When comparing with other algorithms, discretizations with cell sizes computed as shown in Section III-G were used.

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. We only average results for successful runs and we also report the success rate.

C. Comparative Analysis

Table I shows significant computational gains for KPIECE in terms of runtime, when compared to other algorithms such as RRT, EST, and PDST. The success rate of KPIECE is also better than for the other algorithms – 100% for almost all experiments, even when the other algorithms drop to 0%. Figure 7 and Figure 10 show actual runtimes from the data in Table I. As the dimensionality of the problem increases,

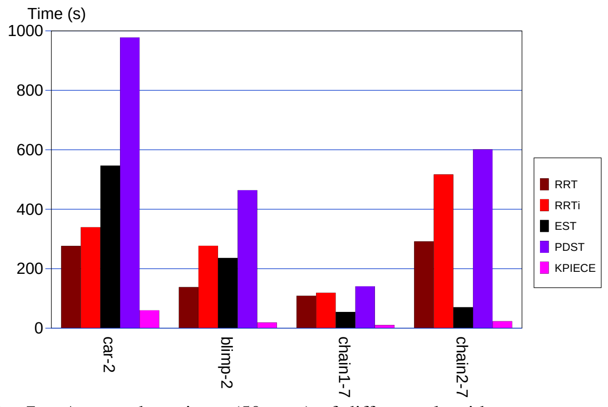


Fig. 7. Averaged runtimes (50 runs) of different algorithms on some of the tested models. The achieved speedup is shown in Table I.

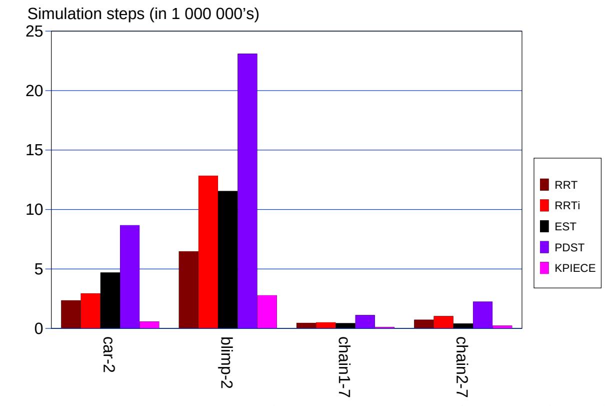


Fig. 8. Averaged number of simulation steps (50 runs) for different algorithms. Notice the similarity to Figure 7. This similarity serves to prove that the runtime of sampling-based planning algorithms is dominated by physics-based simulation, so minimizing the number of simulation steps leads to speedup.

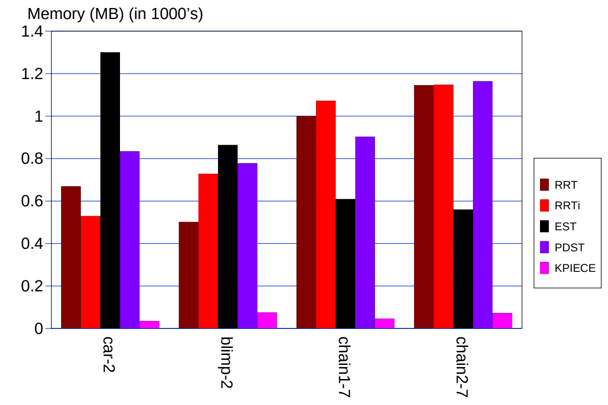


Fig. 9. Average memory usage for different algorithms (50 runs). Notice the similarity to Figure 8. Since for every simulation step it is likely a new state is produced, and the number of motions in the built tree is directly proportional to the number of states, the fewer simulation steps there are, the fewer motions will need to be stored in memory.

TABLE I
COMPARISON OF ALGORITHMS FOR FOUR DIFFERENT PROBLEMS. *spd* DENOTES AVERAGE SPEEDUP OF `KPIECE` WITH RESPECT TO A PARTICULAR ALGORITHM, *mem* DENOTES AVERAGE MEMORY USAGE IN MB AND *suc* DENOTES SUCCESS RATE.
N/A STANDS FOR UNAVAILABLE AVERAGES (0% SUCCESS RATE).

	`RRT`			`RRTi`			`EST`			`PDST`			`KPIECE`		
Problem	*spd*	*mem*	*suc*	*spd*	*mem*	*suc*	*spd*	*mem*	*suc*	*spd*	*mem*	*suc*	*spd*	*mem*	*suc*
car-1	3.91	255	1.00	5.38	232	1.00	17.75	1074	1.00	30.66	665	1.00	1.00	23	1.00
car-2	4.60	670	1.00	5.65	530	1.00	9.10	1301	0.63	16.27	834	0.76	1.00	35	1.00
car-3	5.95	1790	0.11	7.87	1513	0.30	N/A	N/A	0.00	N/A	N/A	0.00	1.00	64	1.00
blimp-1	3.07	171	1.00	4.01	179	1.00	5.34	309	1.00	19.13	535	1.00	1.00	46	1.00
blimp-2	7.21	502	1.00	14.46	729	0.91	12.32	864	0.89	24.21	779	0.72	1.00	76	1.00
blimp-3	1.43	1645	0.24	1.71	1458	0.15	1.38	1613	0.13	N/A	N/A	0.00	1.00	347	1.00
chain1-2	0.54	0	1.00	0.57	0	1.00	0.62	0	1.00	1.32	0	1.00	1.00	3	1.00
chain1-3	0.49	0	1.00	0.54	0	1.00	0.59	0	1.00	1.25	0	1.00	1.00	3	1.00
chain1-4	0.88	0	1.00	0.89	0	1.00	1.45	0	1.00	2.00	0	1.00	1.00	4	1.00
chain1-5	4.00	21	1.00	3.62	21	1.00	4.00	22	1.00	6.05	32	1.00	1.00	4	1.00
chain1-6	10.29	173	1.00	6.47	107	1.00	10.60	211	1.00	6.76	91	1.00	1.00	11	1.00
chain1-7	10.28	1000	0.33	11.23	1073	0.52	5.14	610	0.26	13.27	903	0.89	1.00	46	1.00
chain1-8	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	1.00	150	1.00
chain1-9	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	1.00	726	0.96
chain1-10	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	1.00	1970	0.36
chain2-5	23.49	73	1.00	36.84	84	1.00	149.38	392	1.00	74.55	141	1.00	1.00	6	1.00
chain2-6	62.40	568	0.96	51.78	407	1.00	125.41	1555	0.15	152.74	927	0.67	1.00	13	1.00
chain2-7	12.45	1146	0.57	22.06	1148	0.48	3.01	560	0.04	25.65	1164	0.17	1.00	73	1.00
chain2-8	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	3.24	1083	0.02	1.00	718	0.96
chain2-9	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	1.00	1621	0.14
chain2-10	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00

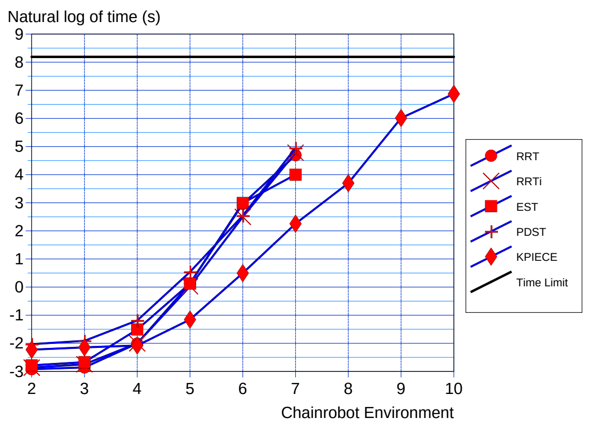


Fig. 10. Averaged runtimes (50 runs) of different algorithms on the modular robot model with no obstacles, using varying number of modules (chain1-x). The achieved speedup is shown in Table I.

`KPIECE` does better and speedups of up to two orders of magnitude are observed (e.g., for chain2-6). For simple problems however, other algorithms can be faster (e.g., `RRT` and `RRTi` for chain1-4). In fact, it is not recommended that a more elaborate algorithm such as `KPIECE` is used when simple algorithms such as `RRT` are sufficient.

As robots become more complex, the size of a state increases and many sampling-based planners reach the memory limit. Table I and Figure 9 show that the memory footprint of `KPIECE` is reduced by as much as 10 to 40 times, compared to that of the other algorithms.

The presented speedup values are consistent with the time spent performing simulations, as shown in Figure 8, which serves to prove that the computational improvements are obtained by minimizing the usage of the physics-based simulator. Since physics-based simulation takes up around 90% of the execution time, computational gain will be observed in the case of purely geometric planning as well, where simulation is replaced by collision detection. With less time spent performing simulations, there are fewer samples to store in the built tree. This leads to significant savings in memory consumption, as indicated in Figure 9.

D. Using Random Projections

The experiments shown in the previous section employed user-defined projections, as described above. In Section III-F it was discussed that if a user-defined projection is not available, random projections can be used as an alternative. To exhibit the behaviour of `KPIECE` when using random projections, we repeat some of our experiments, with random projections set instead of user-defined projections.

We show the results of using 2-, 3- and 4-dimensional projections in Table II. On one hand, the performance of the algorithm degrades as the dimension of the projection becomes larger. For instance, for the car model, a 4-D projection performs worse than a 3-D projection. On the other hand, for systems that are higher dimensional, the increased costs of maintaining a projection with more dimensions are outweighed by the better approximation of coverage, which leads to better-informed exploration and thus lower runtimes, as shown by the blimp model. For the CKBot model we only show experiments for up to 8 modules as for 9 or more modules the failure rate is more than 90% when using random projections.

The experimental results indicate that user-defined projections may lead to better performance. This is expected as some user intuition and knowledge about the problem is inserted in

TABLE II
SLOW-DOWN (*sld*) AND SUCCESS RATE (*suc*) OF KPIECE WHEN USING RANDOM PROJECTIONS, AS OPPOSED TO USING USER-DEFINED PROJECTIONS. REGARDLESS OF THE PROJECTION, AUTOMATICALLY COMPUTED DISCRETIZATIONS ARE USED, AS DESCRIBED IN SECTION III-G. N/A STANDS FOR UNAVAILABLE AVERAGES (0% SUCCESS RATE).

	2D		3D		4D	
Problem	*sld*	*suc*	*sld*	*suc*	*sld*	*suc*
car-1	1.14	1.00	1.22	1.00	1.27	1.00
car-2	1.30	1.00	1.21	1.00	1.72	1.00
car-3	7.80	0.76	2.28	1.00	3.83	1.00
blimp-1	1.87	1.00	0.89	1.00	0.99	1.00
blimp-2	13.17	0.96	1.12	1.00	0.98	1.00
blimp-3	4.35	0.39	2.14	0.87	1.46	1.00
chain1-5	1.09	1.00	1.10	1.00	1.18	1.00
chain1-6	5.78	0.98	3.34	1.00	7.00	1.00
chain1-7	29.61	0.28	15.87	0.85	15.70	0.76
chain1-8	N/A	0.00	17.55	0.33	22.24	0.28
chain1-9	N/A	0.00	5.33	0.02	5.63	0.02
chain2-5	4.52	0.72	4.84	1.00	3.39	1.00
chain2-6	16.12	0.72	9.94	0.98	7.78	1.00
chain2-7	15.75	0.09	7.36	0.63	17.51	0.83
chain2-8	16.45	0.02	4.04	0.41	4.89	0.33
chain2-9	N/A	0.00	N/A	0.00	2.50	0.07

the choice of the projection. For the cases when user-defined projections are not specified, random projections can be used without significant degradation of performance for systems of moderate dimension (up to 10).

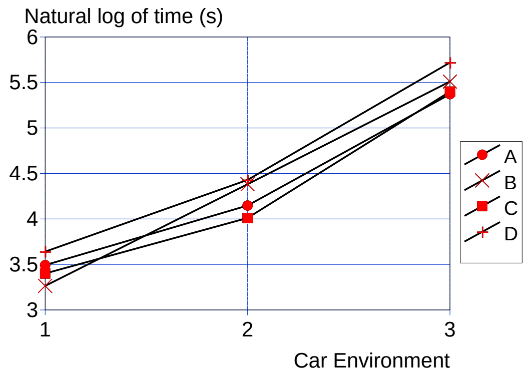
E. Discussion on Algorithm Components

In the previous sections, we have shown the computational benefits of using KPIECE over other algorithms. Two features of KPIECE that can be easily disabled are the grouping of cells into interior and exterior, and the progress evaluation based on increase in coverage (updates to score). Disabling these components allows us to evaluate their contribution individually.

Figure 11 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 (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. However, due to the randomized nature of sampling-based algorithms it is difficult to make absolute statements about the performance of such algorithms in general.

F. Planning in a Kinematic Context

KPIECE was designed to be used for motion planning with differential constraints. However, this does not mean the algorithm cannot be used for kinematic problems as well. Furthermore, comparisons with certain types of algorithms, such as ones that use lazy collision checking or bi-directional search, cannot be performed for the problems with differential constraints presented above. To shed some light on



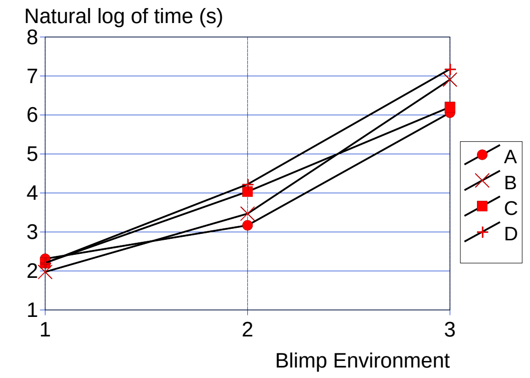


Fig. 11. Logarithmic runtime for KPIECE with various components disabled, on 2-dimensional and 3-dimensional projections (car and blimp) with the automatically computed one-level discretization. A = no components disabled, B = no cell distinction, C = no progress evaluation, D = no cell distinction and no progress evaluation.

the performance of KPIECE for kinematic problems, two experiments with only kinematic constraints are included. The first experiment is that of moving an arm with 7 degrees of freedom (The PR2 arm from Willow Garage) from a position above a table to a position under the table, as show in Figure 12. The second experiment is that of moving a rigid body from a start configuration to a goal configuration in a complex environment, as shown in Figure 13. The sampling of controls was replaced with sampling of random states and connecting to the random states using a local planner. The projection used in the first experiment was a two dimensional one, consisting of the joint values at the first two joints of the arm. For the second experiment, the projection was the position of the rigid body in space (ignoring orientation).

TABLE III
RUNTIMES OF KINEMATIC VERSIONS OF THE ALGORITHMS.

Algorithm	Arm Plan Time(ms)	Rigid Body Plan Time (ms)
RRT	456	3248
EST	187	3907
KPIECE	166	698
RRTConnect	21	1508
SBL	29	3943
LBKPIECE	37	1146

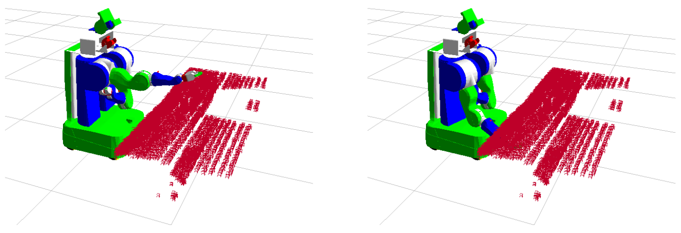


Fig. 12. Move the right arm from above to below the table: start state (left) and goal state (right). The representation of the table is as observed using a laser scanner.

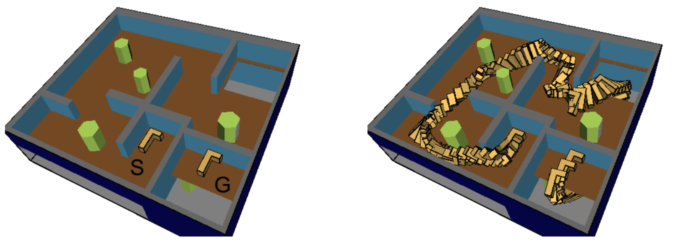


Fig. 13. Move the "L"-shaped rigid body from start to goal, indicated by "S" and "G", respectively.

Table III shows the runtimes of various algorithms in this context. KPIECE is still faster than RRT and EST, but the speedup is not as significant as in the previous examples. For comparison, runtimes of bi-directional search algorithms, SBL [24] and RRTConnect [22], are included. LBKPIECE is a lazy bi-directional implementation of KPIECE, with a connection strategy similar to that of SBL. For the arm problem, the bi-directional versions are an order of magnitude faster, with RRTConnect outperforming the other algorithms. For the rigid body problem, since the start and goal states are close in the workspace, bi-directional algorithms no longer perform as well. In fact KPIECE performs best due to its ability to expand towards unexplored space. The implementations of these algorithms are part of the Open Motion Planning Library (OMPL) [57].

V. Algorithm Extensions

This section introduces the additional improvements of KPIECE, with corresponding experimental results.

A. Multiple Levels of Discretization

The $Proj$ and $Coord$ functions introduced in Section III-B allow discretizing Q in spaces $Cell(\mathbf{z})$, for $\mathbf{z} \in \mathbb{Z}^k$. While this is useful for evaluating coverage, the number of cells may increase significantly while KPIECE is running. This can lead to increased computational requirements. Even though KPIECE was shown to significantly outperform competing algorithms, we propose a means to further speed up the algorithm: addition of coarser levels of discretization.

We generalize the definition of $Coord$:

$$\begin{aligned} Coord(\mathbf{p}, L) &= Coord((p_1, \dots, p_k), L) \\ &= \left(\left\lfloor \frac{p_1 - o_1}{d_{L,1}} \right\rfloor, \dots, \left\lfloor \frac{p_k - o_k}{d_{L,k}} \right\rfloor \right) = \mathbf{z}, \end{aligned}$$

where $\mathbf{p}$ and $\mathbf{o}$ are the same as in the original definition of $Coord$. L is a positive integer specifying the *level of discretization*. $d_{1,i},\ i \in \{1, \dots, k\}$ are the side lengths of the k-dimensional cubes at the lowest level of discretization ($L = 1$). For $L > 1$, $d_{L,i},\ i \in \{1, \dots, k\}$ are defined such that $d_{L,i} = d_{L-1,i} \cdot g_{L,i}$ for $g_{L,i}$ a positive integer. In effect, this definition specifies how to discretize a projection space $\mathbb{R}^k$ into multiple levels of discretization. Each level has cells of uniform volume. A cell at a higher level of discretization consists of an integer number of cells at the next lower level of discretization. Figure 14 shows this pictorially. The lowest level of discretization corresponds to the discretization as described in Section III-B. The higher levels of discretization consist of coarser cells. The intuition behind using multiple levels of discretization is that higher levels of discretization allow the planner to quickly identify the general area that needs to be further explored, and lower levels of discretization allow the planner to focus on the most promising locations within that general area.

The definitions for $Cell(\mathbf{z})$, $Motions(\mathbf{z})$, $Neighbors(\mathbf{z})$ and $Importance(\mathbf{z})$ can clearly be generalized to $Cell(\mathbf{z}, L)$, $Motions(\mathbf{z}, L)$, $Neighbors(\mathbf{z}, L)$ and $Importance(\mathbf{z}, L)$.

Let $M_c(L) \subset \mathbb{Z}^k$ be the minimal cover of AS (as defined in Section III-A) consisting of cells from level L: $AS \subseteq \bigcup_{\mathbf{z} \in M_c(L)} Cell(\mathbf{z}, L)$.

For a cell $Cell(\mathbf{z}, L)$, define the contained cells to be:

$$\begin{aligned} IncCells(\mathbf{z}, L) &= \emptyset, \text{ if } L = 1, \\ IncCells(\mathbf{z}, L) &= \{\mathbf{y} \in \mathbb{Z}^k \mid Cell(\mathbf{y}, L-1) \subset Cell(\mathbf{z}, L) \\ &\qquad \text{and } \mathbf{y} \in M_c(L-1)\}, \text{ if } L > 1. \end{aligned}$$

For generalizing coverage, we use the following definition:

$$\begin{aligned} Coverage(\mathbf{z}, 1) &= \text{previous } Coverage(\mathbf{z}), \\ Coverage(\mathbf{z}, L) &= |IncCells(\mathbf{z}, L)|, \text{ for } L > 1. \end{aligned}$$

This means that the coverage at the first level of discretization ($L = 1$) is the same as originally defined. For higher levels of discretization ($L > 1$), the coverage of a cell is simply the number of instantiated cells it contains from the next, lower level of discretization.

We define an $\mathcal{L}$-level discretization to be:

$$\mathcal{H}_{\mathcal{L}} = \{ M_c(1), \dots, M_c(\mathcal{L}) \}.$$

The coordinates of the cells that make up $\mathcal{H}_{\mathcal{L}}$ are stored in a multi-level hash data structure called Grid. At the lowest level, Grid contains a hash that associates $Motions(\mathbf{z}, 1)$ to each cell coordinate $\mathbf{z} \in M_c(1)$. For higher levels $L > 1$, Grid contains a hash that associates $IncCells(\mathbf{z}, L)$ to each cell coordinate $\mathbf{z} \in M_c(L)$.

The selection of cells (Algorithm 2, line 5) becomes selection of cell chains, as shown in Algorithm 3. In a discretization with $\mathcal{L}$ levels, KPIECE selects a *chain of cells* $\mathbf{c} = (Cell(\mathbf{z}_1, 1), Cell(\mathbf{z}_2, 2), \dots, Cell(\mathbf{z}_{\mathcal{L}}, \mathcal{L}))$ such that $Cell(\mathbf{z}_1, 1) \subset Cell(\mathbf{z}_2, 2) \subset \dots \subset Cell(\mathbf{z}_{\mathcal{L}}, \mathcal{L})$. $\mathcal{L}$ is fixed during the execution of the algorithm. KPIECE identifies the most important cell at the coarsest level of discretization, $Cell(\mathbf{z}_{\mathcal{L}}, \mathcal{L})$, and then proceeds recursively within that cell, and identifies the most important cell in $IncCells(\mathbf{z}_{\mathcal{L}}, \mathcal{L})$.

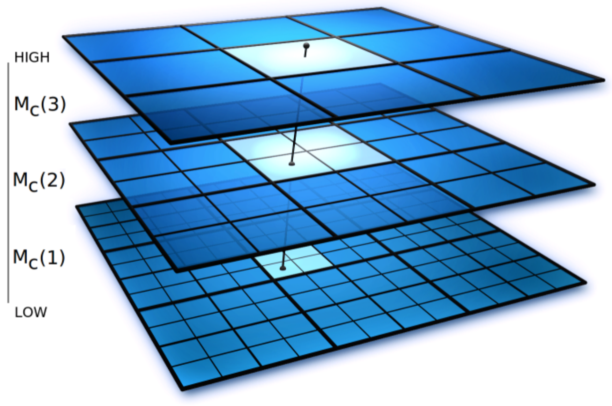


Fig. 14. An example discretization with three levels ($\mathcal{H}_3$). The line intersecting the three levels defines a cell chain. Cell sizes at lower levels of discretization are integer multiples of the cell sizes at the level above.

Algorithm 3. SELECT($bias$)

```
1:  c = ()
2:  $\mathcal{L}$ = the number of levels in the discretization
3:  $type$ = uniform random number in $(0, 1)$
4:  if $type \geq bias$ then
5:      c[$\mathcal{L}$] = exterior cell coordinate from $M_c(\mathcal{L})$ with highest importance
6:  else
7:      c[$\mathcal{L}$] = interior cell coordinate from $M_c(\mathcal{L})$ with highest importance
8:  for $i \leftarrow \{\mathcal{L} - 1...1\}$ do
9:      $type$ = uniform random number in $(0, 1)$
10:     if $type \geq bias$ then
11:         c[$i$] = exterior cell coordinate from $IntCells(\mathbf{c}[i+1], i+1)$ with highest importance
12:     else
13:         c[$i$] = interior cell coordinate from $IntCells(\mathbf{c}[i+1], i+1)$ with highest importance
14: return c
```

This will be a cell $Cell(\mathbf{z}_{\mathcal{L}-1}, \mathcal{L} - 1)$. Preference is given to exterior cells. This process continues until a cell from the finest level of discretization is selected: $Cell(\mathbf{z}_1, 1)$. A motion ν from $Motions(\mathbf{z}_1, 1)$ is then picked according to a half-normal distribution.

The update to `score` in Algorithm 2 [lines 14,15] does not change, but it is now performed for every cell in a selected chain.

TABLE IV
SPEEDUP (*spd*) ACHIEVED BY KPIECE WHEN USING A DISCRETIZATION $\mathcal{H}_2$ RELATIVE TO THE AUTOMATICALLY COMPUTED DISCRETIZATION $\mathcal{H}_1$.

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

While the results shown in Table I are computed with a discretization $\mathcal{H}_1$ (one level of discretization), for some problems, better results can be obtained using multiple levels of discretization. To show this, for each robot, twelve simple discretizations are defined. First, a $\mathcal{H}_1$ discretization (consisting only of $M_c(1)$) is computed as discussed in Section III-G. Two more $\mathcal{H}_1$ 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 $\mathcal{H}_1$ discretizations, three more $\mathcal{H}_2$ discretizations (consisting of $M_c(1)$, $M_c(2)$) are defined: ones that have the same cell sizes for $M_c(1)$, but $M_c(2)$ consists of cells with sizes increased by a factor of 10, 15, and 20 times, in each dimension, with respect to the cell sizes of $M_c(1)$.

Table IV shows the speedup obtained when employing the best of the nine defined $\mathcal{H}_2$ discretizations, compared to the automatically computed discretization $\mathcal{H}_1$, computed as in Section III-G. 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 cell sizes. As shown in Figure 15, the runtimes of the algorithm for the twelve different discretizations (both $\mathcal{H}_1$ and $\mathcal{H}_2$) are relatively close to one another (mostly within a factor of 2). This suggests that the algorithm is not overly sensitive to the defined discretization and thus approximating good cell sizes and number of levels of discretization is sufficient.

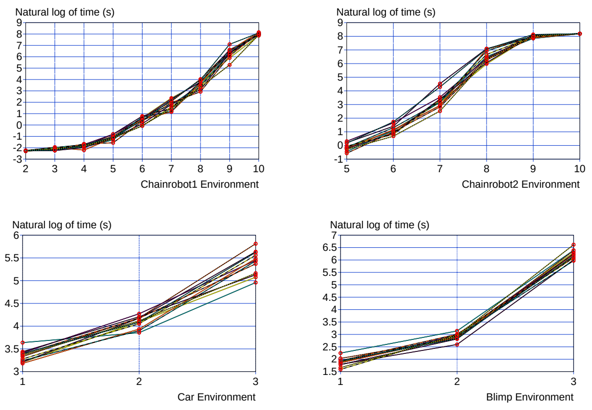


Fig. 15. Logarithmic runtimes with twelve different discretizations for the chain1, chain2, car, and blimp.

The results from Table IV and Figure 15 suggest that carefully choosing the number of levels of discretization and the cell sizes at these levels, can lead to computational benefits (e.g., blimp-1, blimp-2, blimp-3). However, KPIECE is robust to these settings and very good results can be obtained without spending significant effort on tuning parameters for the employed discretization. For automatic computation of these parameters, parameter sweeps can be conducted.

B. Goal Biasing

Goal biasing is a standard technique used by sampling-based 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 [12]. 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 $h(s)$ instead of 1 in Algorithm 2, where s is the last state of the motion ν that required the creation of the cell (we say $h^*(\nu) = h(s)$). 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, the progress evaluation component of the algorithm becomes essential, 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 ν, $h^*(\nu)$ is computed; $\mathcal{B}$ is a limited-size set (at most 30, in our implementation) of motions with the largest values of $h^*(\nu)$. 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. This constraint prevents having all motions in $\mathcal{B}$ be almost the same.

Adding biasing for `KPIECE` does often improve the runtime, as shown in Table V. The fact that progress evaluation is also present limits the negative effects biasing has in certain cases (e.g., blimp-3).

TABLE V
SPEEDUP (*spd*) ACHIEVED BY `KPIECE` WITH BIASING RELATIVE TO `KPIECE`, BOTH USING AN AUTOMATICALLY COMPUTED DISCRETIZATION.

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

C. 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 [58], [59], 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 [58], [59], `KPIECE` uses multiple threads to build the same tree of motions (threads can continue expanding from cells instantiated by other threads). Synchronization points are employed to ensure correct order of execution. Since there is only one copy of the exploration tree, there is no communication necessary, as would be the case with a distributed approach. Shared memory parallelization takes advantage of the increase in number of computing cores and memory bandwidth. 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. This proposed parallelization scheme can be applied to other sampling-based algorithms as well (e.g., [57]).

TABLE VI
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 III-G.

#	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 VII
SPEEDUP ACHIEVED BY `KPIECE` IN EMBARRASSINGLY PARALLEL MODE.

#	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 sections were conducted when using the planner in single-threaded mode. Table VI 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 motion planners [60]. When comparing to the speedup obtained with an embarrassingly parallel setup, shown in Table VII, 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.

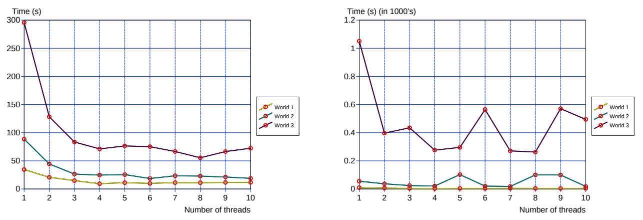


Fig. 16. Runtime with different number of threads on a four-core machine, for 2-dimensional and 3-dimensional projections (car and blimp).

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 16). 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.

VI. Discussion and Future Work

We have presented KPIECE, a sampling-based motion planning algorithm that can solve complex planning problems within significantly reduced runtime (up to two orders of magnitude speedup), and with substantial memory savings (up to a factor of 40). Furthermore, KPIECE enables the solution of problems that could not be tackled before.

KPIECE uses a projection of the state space to a lower dimensional Euclidean space and a specification of a discretization. As shown in the experiments, even simple intuitive projections work for complex problems. If a projection is not defined by the user, the algorithm performs well with random projections for systems of moderate dimension.

The performance of KPIECE is not drastically affected by the choice of discretization and a method to automatically compute $\mathcal{H}_1$ discretizations was presented. Parameter sweeps can be used to decide when further levels of discretization are beneficial. When using automatically computed $\mathcal{H}_1$ discretizations, 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.

For future work, it would be interesting to prove the probabilistic completeness of KPIECE, to develop a principled approach for automatically computing discretizations with more levels of discretization, and to explore the possibility of using multiple projection spaces simultaneously. Another possible direction is that of using PCA to identify vectors along which to project, but extensive experiments have not yet been conducted. This use of PCA has been shown to work locally to help in the exploration of narrow passages [50].

Acknowledgements

An earlier version of this work appeared in [34]. The authors would like to thank Mark Yim and Jonathan Kruse for their help in defining the CKBot ODE model and the members of the Kavraki Lab for providing valuable comments. 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, NSF DUE 0920721 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.

References

[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] L. Zhang, J. Pan, and D. Manocha, "Motion planning of human-like robots using constrained coordination," in *IEEE-RAS International Conference on Humanoid Robots*, Paris, December 2009.

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

[7] 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.

[8] 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.

[9] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Hybrid systems: from verification to falsification by combining motion planning and discrete search," *Formal Methods in System Design*, vol. 34, pp. 157–182, 2009.

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

[11] 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.

[12] I. A. Şucan and L. E. Kavraki, "On the implementation of single-query sampling-based motion planners," in *IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, May 2010, pp. 2005–2011.

[13] P. Cheng and S. M. LaValle, "Reducing metric sensitivity in randomized trajectory design," in *IEEE International Conference on Robotics and Automation*, Seoul, Korea, May 2001, pp. 43–48.

[14] J. Bruce and M. Veloso, "Real-time multi-robot motion planning with safe dynamics," *Multi-Robot Systems: From Swarms to Intelligent Automata Volume III*, pp. 159–170, 2005.

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

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

[17] 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.

[18] A. Shkolnik, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," in *IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009, pp. 2859–2865.

[19] D. E. Stewart, "Rigid-body dynamics with friction and impact," *SIAM Review*, vol. 42, no. 1, pp. 3–39, 2000.

[20] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. J. Kuffner., "Bispace planning: Concurrent multi-space exploration," in *Robotics: Science and Systems*, Zurich, Switzerland, June 2008.

[21] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep. 11, 1998.

[22] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, San Francisco, California, April 2000, pp. 995–1001.

[23] D. Hsu, J. C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *IEEE International Conference on Robotics and Automation*, vol. 3, Albuquerque, New Mexico, April 1997, pp. 2719–2726.

[24] G. Sánchez and J. C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," *International Journal of Robotics Research*, vol. 6, pp. 403–417, 2003.

[25] R. Bohlin and L. Kavraki, "Path planning using Lazy PRM," in *IEEE International Conference on Robotics and Automation*, San Francisco, California, April 2000, pp. 521–528.

[26] E. Ferre and J.-P. Laumond, "An iterative diffusion algorithm for part disassembly," in *IEEE International Conference on Robotics and Automation*, New Orleans, Louisiana, April 2004, pp. 3149–3154.

[27] 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.

[28] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato, "RESAMPL: A region-sensitive adaptive motion planner," in *Algorithmic Foundation of Robotics VII*. Springer, STAR 47, 2008, pp. 285–300.

[29] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle, "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain," in *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, April 2005, pp. 3856–3861.

[30] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *IEEE International Conference on Robotics and Automation*, Orlando, Florida, May 2006, pp. 1243–1248.

[31] 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.

[32] B. Burns and O. Brock, "Single-query motion planning with utility-guided random trees," in *IEEE International Conference on Robotics and Automation*, Rome, Italy, April 2007, pp. 3307–3312.

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

[34] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Algorithmic Foundation of Robotics VIII*. Springer, STAR 57, 2009, pp. 449–464.

[35] ——, "On the performance of random linear projections for sampling-based motion planning," in *International Conference on Intelligent Robots and Systems*, St. Louis, Missouri, October 2009, pp. 2434–2439.

[36] J. H. Reif, "Complexity of the mover's problem and generalizations," in *IEEE Symposium on Foundations of Computer Science*, 1979, pp. 421–427.

[37] 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.

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

[39] P. Cheng, G. Pappas, and V. Kumar, "Decidability of motion planning with differential constraints," in *IEEE International Conference on Robotics and Automation*, Rome, Italy, April 2007, pp. 1826–1831.

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

[41] J. H. Reif and S. Slee, "Optimal kinodynamic motion planning for self-reconfigurable robots between arbitrary 2d configurations," in *Robotics: Science and Systems*, W. Burgard, O. Brock, and C. Stachniss, Eds. Atlanta, Georgia: MIT Press, June 2007.

[42] http://sourceforge.net/projects/opende/, seen May 19^{th} 2011.

[43] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. 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.

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

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

[46] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Transactions on Robotics*, vol. 26, pp. 469–482, June 2010.

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

[48] M. Kalisiak and M. v. d. Panne, "RRT-blossom: RRT with a local flood-fill behavior," in *IEEE International Conference on Robotics and Automation*, Orlando, Florida, May 2006, pp. 1237–1242.

[49] H. Kurniawati and D. Hsu, "Workspace importance sampling for probabilistic roadmap planning," in *International Conference on Intelligent Robots and Systems*, Sendai, Japan, September 2004, pp. 1618–1623.

[50] S. Dalibard and J.-P. Laumond, "Control of probabilistic diffusion in motion planning," in *Algorithmic Foundations of Robotics VIII*. Springer, STAR 57, 2009, pp. 467–481.

[51] E. R. Johnson and T. D. Murphey, "Scalable variational integrators for constrained mechanical systems in generalized coordinates," *IEEE Transactions on Robotics and Automation*, vol. 25, pp. 1249–1261, 2009.

[52] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *International Conference on Intelligent Robots and Systems*, Sendai, Japan, September 2004, pp. 2149–2154.

[53] R. Tedrake, "LQR-trees: Feedback motion planning on sparse randomized trees," in *Robotics: Science and Systems*, Seattle, USA, June 2009.

[54] W. B. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space," *Contemporary Mathematics*, vol. 26, pp. 189–206, 1984.

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

[56] http://kavrakilab.org/OOPSMP/.

[57] "The Open Motion Planning Library (OMPL)," http://ompl.kavrakilab.org, 2010.

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

[59] 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.

[60] D. Challou, M. Gini, V. Kumar, and C. Olson, "Very fast motion planning for dexterous robots," *IEEE International Symposium on Assembly and Task Planning*, pp. 201–206, August 1995.

Ioan A. Şucan is a Ph.D. candidate in computer science at Rice University, Houston, TX. He is studying algorithmic robotics under Dr. Lydia E. Kavraki. His research interests include motion planning under differential constraints, multi-threaded search algorithms and task and motion planning for manipulation. He received the B.S. degree in electrical engineering and computer science from Jacobs University, Bremen, Germany, in 2006 and the M.S. degree in computer science from Rice University, Houston, TX, in 2008.

Lydia E. Kavraki received the Ph.D. degree in computer science from Stanford University, Stanford, CA, in 1995. She is currently the Noah Harding Professor of computer science and bioengineering with Rice University, Houston, TX. She is the author/coauthor of more than 150 technical papers and a robotics textbook: Principles of Robot Motion (MIT Press, 2005). Her interests include motion planning for continuous and hybrid systems, planning with temporal specifications, mobile manipulation, networked multiagent systems, and applications of robotics methods to biology. Prof. Kavraki is a Fellow of the Association for the Advancement of Artificial Intelligence and the American Institute for Medical and Biological Engineering. She is the recipient of the ACM Grace Murray Hopper Award and the Early Academic Career Award from the IEEE Robotics and Automation Society. Information about her work can be found at http://www.kavrakilab.org.