shed in Algorithmic Foundation of Robotics VIII (Proceedings of Workshop on the Algorithmic Foundations of Robotics). 57:449-464 (2009)

Kinodynamic Motion Planning by Interior-Exterior Cell Exploration

Ioan A. Şucan[1] and Lydia E. Kavraki[1]

Department of Computer Science, Rice University, {isucan, kavraki}@rice.edu

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 physics-based simulation is necessary. A multiple-level grid-based discretization is used to estimate the coverage of the state space. The coverage estimates help the planner detect the less explored areas of the state space. The planner also keeps track of the boundary of the explored region of the state space and focuses exploration on the less covered parts of this boundary. Extensive experiments show KPIECE provides computational gain over state-of-the-art methods and allows solving some harder, previously unsolvable problems. A shared memory parallel implementation is presented as well. This implementation provides better speedup than an embarrassingly parallel implementation by taking advantage of the evolving multi-core technology.

1 Introduction

Over the last two decades, motion planning [4, 15, 17] has grown from a field that considered basic geometric problems to a field that addresses planning for complex robots with kinematic and dynamic constraints [5, 25]. Applications of motion planning have also expanded to other fields such as graphics and computational biology [16].

Much of the recent progress in motion planning is attributed to the development of sampling-based algorithms [4, 17]. A sampling-based motion planning algorithm can only be probabilistically complete [9, 12], which means if a solution exists, it will be eventually found. One of the first successful sampling-based motion planners was the Probabilistic Roadmap Method (PRM) [10]. This method provided a coherent framework for many earlier works that used sampling and opened new directions for research [2]. In the case of realistic robots, taking dynamic constraints into account (kinodynamic motion planning) is a necessity. Sampling-based tree planners such as Rapidly-exploring Random Trees (RRT) [11, 19], Expansive Space Trees (EST) [6, 7] have been successfully used to solve such problems. These planners build a tree of motions in the state space of the robot and attempt to reach the goal state. Many

variations of these planners exist as well (*e.g.*, [8, 18, 22]). More recent planners have been designed specifically for planning with complex dynamic constraints [14, 21]. The Path-Directed Subdivision Tree (`PDST`) planner [13, 14] has been used in the context of physics-based simulation as well.

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

The rest of the paper is organized as follows: Section 2 presents the motion planning problem in more detail, Section 3 contains a description of the proposed algorithm, and Section 4 presents experiments using `KPIECE`. The parallel implementation is discussed in Section 5. Conclusions and future work are in Section 6.

2 Problem Definition

An instance of the motion planning problem addressed here can be formally defined by the tuple $S = (Q, U, I, F, f)$ where Q is the state space, U is the control space, $I \subset Q$ is the set of initial states, and $F \subset Q$ is the set of

final states. The dynamics are described by a forward propagation routine $f : Q \times U \rightarrow TgQ$, where TgQ is the tangent space of Q (f does not need to be explicit). A solution to a motion planning problem instance consists of a sequence of controls $u_1, \ldots, u_n \in U$ and times $t_1, \ldots, t_n \in \mathbb{R}^{\geq 0}$ such that $q_0 \in I$, $q_n \in F$ and $q_k, k = 1, \ldots, n$, can be obtained sequentially by integration of f.

For the purposes of this work, the function f is computed by a physics simulator. In particular, an open source library called Open Dynamics Engine (`ODE`) [24], is used. Instead of equations of motion to be integrated, a model of a robot and its environment needs to be specified. Although simulation incurs more computational costs than simple integration, the benefits outweigh the costs: increased accuracy is available since physics simulators take into account more dynamic properties of the robot (such as gravity, friction) and constructing models of systems is easier and less error prone than deriving equations of motion. Limited numerical precision will still be a problem regardless of how f is computed. However, as robotic systems become more complex, physics-based simulation becomes a necessity.

It is sometimes the case that due to the high dimensionality of the state space Q, a projection space $\mathcal{E}(Q)$ is used for various computations the motion planning algorithm performs ($\mathcal{E}$ can be the identity transform). Finding such a projection $\mathcal{E}$ is a research problem in itself. In this work it is assumed that such a projection $\mathcal{E}$ is available, when needed. Section 4.1 presents simple cases of $\mathcal{E}$ used in this paper.

3 Algorithm

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

3.1 Discretization

During the course of its run, the motion planner must decide which areas of the state space merit further exploration. As the size of the tree of motions increases, making this decision becomes more complex. There are various strategies to tackle this problem (*e.g.*, [7, 14, 19, 21, 22]). The approach taken

in this work is to construct a discretization that allows the evaluation of the coverage of the state space. This discretization consists of k levels $\mathcal{L}_1, ..., \mathcal{L}_k$, as shown in Fig. 1. Each of these levels is a grid where cells are polytopes of fixed size. The number of levels and cell sizes are predefined, however, cells are instantiated only when they are needed. The purpose of these grids is to cover the area of the space that corresponds to the area spanned by the tree of motions. Each of the levels provides a different resolution for evaluating the coverage. Coarser resolution (higher levels) can be used initially to find out roughly which area is less explored. Within this area, finer resolutions (lower levels) can then be employed to more accurately detect less explored areas. The following is a formal definition of a k-level discretization:

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

The tree of motions exists in the state space Q, but since the dimension of this space may be too large, the discretization is typically imposed on a projection of the state space, $\mathcal{E}(Q)$. The use of such a projection $\mathcal{E}(Q)$ was also discussed in [13, 21]. An important result we show in this paper is that simple projections work for complex problems. For any motion μ, each level of discretization contains a cell that μ is part of. A motion μ is considered to be part of a grid cell p if there exists a state s along μ such that the projection $\mathcal{E}(s)$ is inside the bounding box of cell p. If a motion spans more than one cell at the same level of discretization, it is split into smaller motions such that no motions cross cell boundaries. This invariant is maintained to make sure each motion is accounted for only once. For every motion μ, there will be exactly one cell at every level of discretization that μ is part of. This set of cells forms a tuple $\mathbf{c} = (p_1, ..., p_k), p_i \subset p_{i+1}, p_i \in \mathcal{L}_i$ and will be referred to as the "cell chain" for μ. Since cells in $\mathcal{L}_1$ will determine whether a motion is split, we augment the definition of the discretization:

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

For all $p \in \mathcal{L}_1$ we say p contains $\mathcal{M}_p$ and for all $p \in \mathcal{L}_i, i > 1$ we say p contains $\mathcal{D}_p$. While the discretization spans the potentially very large projection space $\mathcal{E}(Q)$, cells are instantiated only when a motion that is part of them is found, hence the grids are not fully instantiated. This allows the motion planner to limit its use of memory to reasonable amounts. The size of the grid cells is discussed in Section 3.4.

A distinguishing feature of `KPIECE` is the notion of interior and exterior cells. A cell is considered exterior if it has less than $2n$ instantiated neighboring cells (diagonal neighboring cells are ignored) at the same level of discretization, where n is the dimension of $\mathcal{E}(Q)$. Cells with $2n$ neighboring cells are

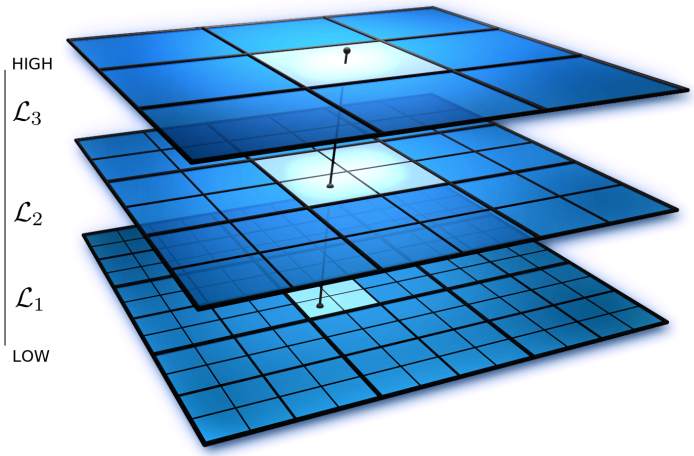

Fig. 1. An example discretization with three levels. The line intersecting the three levels defines a cell chain. Cell sizes at lower levels of discretization are integer multiples of the cell sizes at the level above.

considered interior (there can be no more than $2n$ non-diagonal neighboring cells in an n-dimensional space). As the algorithm progresses and new cells are created, some exterior cells will become interior. When larger parts of the state space are explored, most cells will be interior. However, for very high dimensional spaces, to avoid having only exterior cells, the definition of interior cells can be relaxed and cells can be considered interior before all $2n$ neighboring cells are instantiated. For the purposes of this work, this relaxation was not necessary.

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

3.2 Algorithm Execution

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

At every iteration, a cell chain $\mathbf{c} = (p_1, ..., p_k)$ is sampled. This means $p_i \in \mathcal{L}_i$ will have to be selected, from p_k to p_1, as will be shown later. It is important to note here that "samples" in the case of `KPIECE` are chains of cells. This can be regarded as a natural progression (selection of "volumes") from the selection of states ("points") as in the case of `RRT` and `EST`, and selection of motions ("curves") as in the case of `PDST`. Our experiments show that selecting chains of cells benefits from the better estimates of coverage that can be maintained for cells at each level, as opposed to estimates for single

motions or states. Sampling a cell chain $\mathbf{c} = (p_1, ..., p_k)$ is a k-step process that proceeds as follows: the decision to expand from an interior or exterior cell is made [line 5], with a bias towards exterior cells. An instantiated cell, either interior or exterior, is then deterministically selected from $\mathcal{L}_k$, according to the cell importance (higher importance first). The idea of deterministic selection was inspired by [14], where it has been successfully used. The importance of a cell p, regardless of the level of discretization it is part of, is computed as:

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

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

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

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

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

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

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

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

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

Algorithm 2 AddMotion(s, u, t)

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

3.3 Implementation Details

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

3.4 Computing the Discretization

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

While KPIECE is running, we can keep track of averages of how many motions per cell there are, how many parts a motion is split into before it is added to the discretization, and the ratio of interior to exterior cells. While we do not know how to compute optimal values for these statistics (if they exist), there are certain ranges that may work better than others. In particular, the authors have observed that for good 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.
- Average number of parts in which a motion is split should be larger than 1 but not higher than 4.
- As the algorithm progresses, at least some interior cells need to be created.
- The average number of samples per cell should be in the range of tens to hundreds.

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

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

4 Experiments

The presented algorithm was benchmarked against well-known efficient algorithms (RRT, EST, PDST) with three different robotic systems, in different environments. For modeling the robots, the ODE [24] physics-based simulator was used. For the implementations of RRT [19] and EST [6], the OOPSMP framework was used [20]. A plugin for linking OOPSMP with the ODE simulator was developed by the authors. The authors did their best to tune the parameters of both RRT and EST. For RRT, a number of different metrics were tested for each robot and experiments are presented with the metric that performed best. In addition, random controls were selected instead of attempting to find controls that take the robotic system toward a desired state, as this strategy seemed to provide better results. For EST, the nodes to expand from were selected both based on their degree [7] and based on a grid subdivision of the state space [22]. Experiments are shown for the selection strategy that performed best. PDST and KPIECE were implemented by the authors. A projection was defined for each robot and the same projection was used for both PDST and KPIECE. In addition to the projection, KPIECE needs a discretization

to be defined for each robot. When comparing with other algorithms, only discretizations computed as shown in Section 3.4 were used. Separate experiments are shown when using empirically chosen discretizations with multiple levels. Explanations on how these multiple levels were chosen are given later in this section. No goal biasing was used for any of the algorithms. However, separate experiments are shown for RRT with biasing ($\texttt{RRT}_b$). All implementations are in C++ and were tested on the Rice Cray XD1 Cluster, where each machine runs at 2.2 Ghz and has 8 GB RAM. For each system and each of its environments, each algorithm was executed 50 times. The best two and worse two results in terms of runtime were discarded and the results of the remaining 46 runs were averaged. The time limit was set to one hour and the memory limit was set to 2 GB. If an execution exceeded the time or memory limit, it was considered successful with execution time equal to the time limit.

4.1 Robots

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

Modular Robot

The model for this robot was implemented in collaboration with Mark Yim[1], and characterizes the CKBot modules [23]. Each CKBot module contains one motor. An ODE model for serially linked CKBot modules has been created [5]. The task is to compute the controls for lifting the robot from a vertical down position to a vertical up position for varying number of modules, as shown in Fig. 2. 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. This is why the planner has to find swinging motions to solve the problem. The employed projection $\mathcal{E}$ was a 3-dimensional one, the first two dimensions being the (x, z) coordinates of the last module (x, z is the plane observed in Fig. 2) 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 Fig. 2.

Car Robot

A model of a car [4] was created as well. 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

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

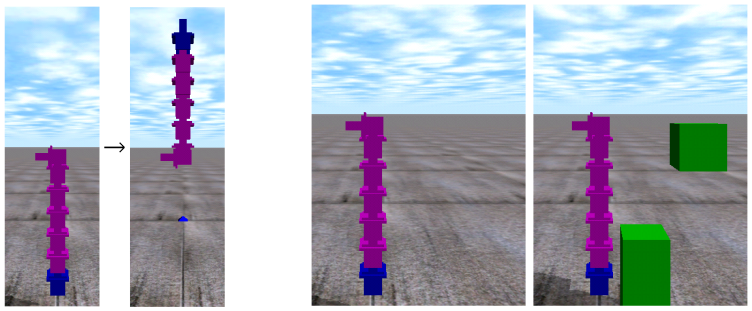

Fig. 2. Left: start and goal configurations. Right: environments used for the chain robot (7 modules). Experiments were conducted for 2 to 10 modules. In the case without obstacles, the environments are named ch1-x where x stands for the number of modules used in the chain. In the case with obstacles, the environments are named ch2-x.

the forward velocity and steering velocity (as recommended by the developers of the library). These desired velocities go together with a maximum allowed force. The end result is that the car will not be able to achieve the desired velocities instantly, due to the limited force. In effect, this makes the system a second order one. The employed projection $\mathcal{E}$ was the (x, y) coordinates of the center of the car body. The environments the system was tested in are shown in Fig. 3.

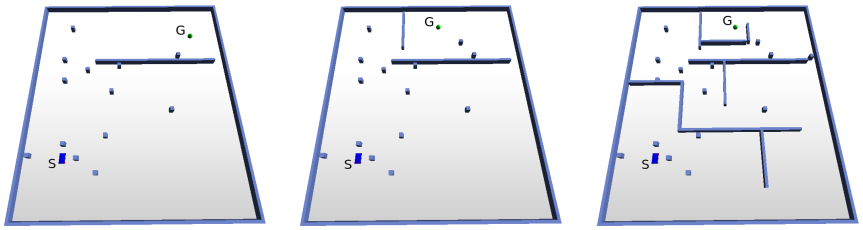

Fig. 3. Environments used for the car robot (cr-1, cr-2, cr-3). Start and goal configurations are marked by "S" and "G".

Blimp Robot

The third robot that was tested was a blimp robot [14]. The motion in this case is executed in a 3D environment. This robot is particularly constrained in its motion: the blimp must always apply a positive force to move forward (slowing down is caused by friction), it must always apply an upward force to lift itself vertically (descending is caused by gravity) and it can turn left or right along the direction of forward motion. Since `ODE` does not include air friction, a Stokes model of drag was implemented for the blimp. The employed projection $\mathcal{E}$ was the (x, y, z) coordinates of the center of the blimp. The environments the system was tested in are shown in Fig. 4.

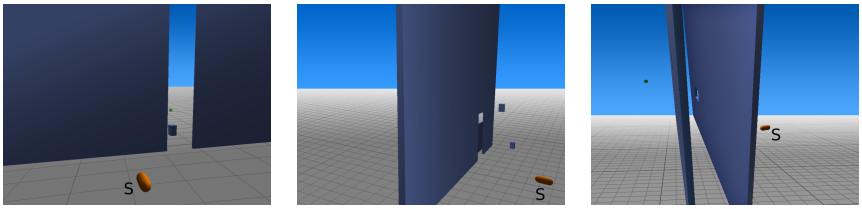

Fig. 4. Environments used for the blimp robot (bl-1, bl-2, bl-3). Start configurations are marked by "S". The blimp has to pass between the walls and through the hole(s), respectively.

4.2 Results

Table 1. Speedup achieved by `KPIECE` over other algorithms for four different problems. If one of the other algorithms was unable to solve the problem in at least 10% of the cases, "—" is reported. `KPIECE` was configured with an automatically computed one-level discretization, as described in Section 3.4.

	RRT	RRT_b	EST	PDST
ch1-2	1.1	3.5	2.2	2.5
ch1-3	0.8	2.1	1.0	3.6
ch1-4	1.5	3.9	1.8	9.6
ch1-5	4.1	3.7	14.4	15.6
ch1-6	13.4	9.6	946.8	42.5
ch1-7	58.5	196.3	—	238.1
ch1-8	—	—	—	—

	RRT	RRT_b	EST	PDST
ch2-5	18.3	23.4	—	13.0
ch2-6	35.0	255.7	—	23.0
ch2-7	45.7	124.7	—	81.3
ch2-8	—	—	—	5.9
ch2-9	—	—	—	—

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

In terms of runtime, when compared to other algorithms such as `RRT`, `EST`, and `PDST`, Table 1 shows significant computational gains for `KPIECE`. In particular, as the dimensionality of the problem increases, `KPIECE` does better. For simple problems however, other algorithms can be faster (*e.g.*, `RRT` for ch1-3). The presented speedup values are consistent with the time spent performing simulations, which serves to prove that the computational improvements are obtained by minimizing the usage of the physics-based simulator. Since physics simulation takes up around 90% of the execution time, computational gain will be observed in the case of purely geometric planning as well, where forward integration is replaced by collision detection.

Table 2. Speedup achieved by `KPIECE` when using a two-level discretization relative to the automatically computed one-level discretization. For ch1-10 and ch2-10, a solution was found only with the two-level discretization so no speedup is reported.

ch1-2: 0.9	ch1-7: 2.2	ch2-5: 0.9	cr-1: 1.0	bl-1: 1.3
ch1-3: 1.1	ch1-8: 2.0	ch2-6: 1.1	cr-2: 1.0	bl-2: 1.1
ch1-4: 0.9	ch1-9: 1.3	ch2-7: 1.7	cr-3: 0.7	bl-3: 1.8
ch1-5: 0.7		ch2-8: 0.5		
ch1-6: 2.5		ch2-9: 1.2		

While the results shown in Table 1 are computed with a one-level discretization, for some problems, better results can be obtained using multiple

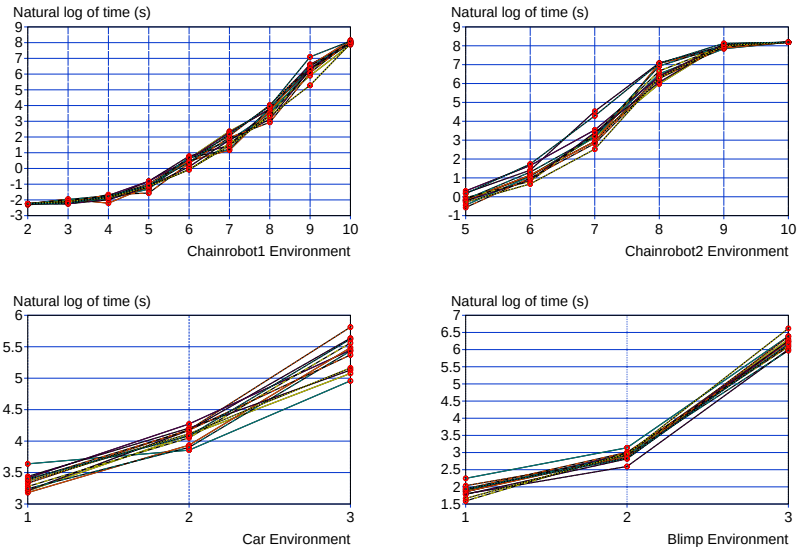

Fig. 5. Logarithmic runtimes with twelve different discretizations for the ch1, ch2, cr, and bl.

levels of discretization. To show this, for each robot, twelve discretizations are defined. First, a one-level discretization (consists only of $\mathcal{L}_1$) is computed as discussed in Section 3.4. Two more one-level discretizations with half and double the cell volume of the computed discretization's cells are then constructed (cell sides shortened and lengthened proportionally, in each dimension). For each of these three one-level discretizations, three more two-level discretizations (consist of $\mathcal{L}_1$, $\mathcal{L}_2$) are defined: ones that have the same $\mathcal{L}_1$, but $\mathcal{L}_2$ consists of cells with sizes of 10, 15, and 20 times the cell sizes of $\mathcal{L}_1$. Table 2 shows the speedup obtained when employing the best of the nine defined two-level discretizations. As we can see, in most cases there are benefits to using two discretization levels. Experiments with more than two levels of discretization were conducted as well, but the performance started to decrease and the results are not presented here. The defined discretizations can also be used to evaluate the sensitivity of `KPIECE` to the defined grid sizes. As shown in Fig. 5, the runtimes of the algorithm for the different discretizations are relatively close to one another (within a factor of 2.3). This implies that the algorithm is not overly sensitive to the defined discretization and thus approximating good cell sizes is sufficient. Nevertheless, finding good discretizations remains an open problem.

4.3 Discussion of Experimental Results

In the previous section we have shown the computational benefits of using `KPIECE` over other algorithms. There are a few key details that make `KPIECE` distinct: the sampling of a chain of cells, the grouping of cells into interior and

exterior, and the progress evaluation, based on increase in coverage. While the sampling of cell chains is an inherent part of the algorithm, the other two features can be easily disabled. This allows us to evaluate the contribution of these components individually.

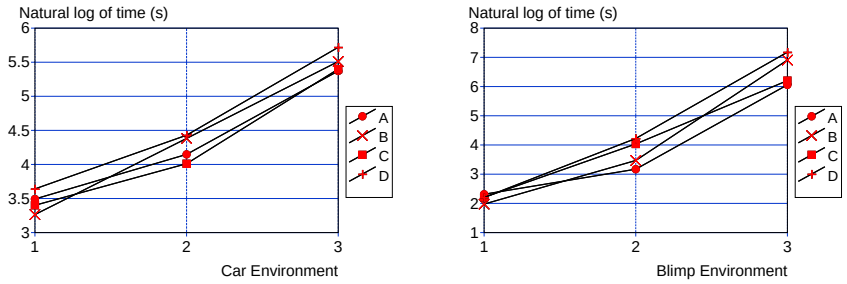

Fig. 6. Logarithmic runtime for `KPIECE` with various components disabled, on 2-dimensional and 3-dimensional projections (cr and bl) with the automatically computed one-level discretization. `A` = no components disabled, `B` = no cell distinction, `C` = no progress evaluation, `D` = no cell distinction and no progress evaluation.

Fig. 6 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 (bl-1), their contribution is important for harder problems (cr-3, bl-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.

5 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 [1, 3], 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 [1, 3], `KPIECE` uses multiple threads to build the same tree of motions (threads can continue expanding from cells instantiated by other threads). Synchronization points are used to ensure correct order of execution. This execution format will become more important in the next few years as the number of computing cores and memory bandwidth increase. Since each computing thread starts from a different random seed, the chances of all seeds being unfavourable decrease. If a single thread finds a path through a narrow passage, the rest of the threads will immediately use this information as well. This setup also reduces the variance in the average runtime of the algorithm. It is important to note this proposed parallelization scheme can be applied to other sampling-based algorithms as well.

All experiments presented in previous sections were conducted when using the planner in single-threaded mode. Table 3 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. When comparing to the speedup obtained with an embarrassingly parallel setup, shown in Table 4, we notice that better runtimes are obtained with our suggested setup. In addition, total memory requirements in our suggested setup do not increase significantly as the number of processors is increased.

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

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

Table 4. Speedup achieved by `KPIECE` in embarrassingly parallel mode.

Threads	cr-1	cr-2	cr-3	bl-1	bl-2	bl-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

6 Conclusions and Future Work

We have presented `KPIECE`, a sampling-based motion planning algorithm designed for complex systems where physics-based simulation is needed. This algorithm does not need a distance metric or a way to sample states. It does however require a projection of the state space and the specification of a discretization. At this point we recommend that the projection is defined by the user. As shown in our experiments, even simple intuitive projections work for complex problems. The discretization is an additional requirement when compared to other state-of-the-art algorithms. The algorithm's performance is not drastically affected by the discretization and a method to automatically compute one-level discretizations was presented. When using an automatically computed one-level discretization, `KPIECE` was compared to other popular algorithms, and shown to provide significant computational speedup. In addition, the provided shared memory parallel implementation seems to give better results than the embarrassingly parallel setup.

`KPIECE` is the result of a combination of ideas. Some of these ideas are new, some are inspired by previous work. In previous work, we have encountered state [7, 11] and motion sampling [14]; `KPIECE` takes this further and uses cell chain sampling. We have also seen progress evaluation [21], deterministic sample selection [14], use of physics-based simulation [13], and use

of additional data-structures for estimation of coverage [14, 21, 22]. `KPIECE` implements variants of these ideas, combined with new ideas like distinction between interior and exterior cells, to obtain an algorithm that works well in a parallel framework. The result is a more accurate and efficient method that can solve problems previous methods could not.

It is conjectured that `KPIECE` is probabilistically complete: in a bounded state space, the number of cells is finite. Since with every selection, the importance of a cell can only decrease, every cell will be selected infinitely many times during the course of an infinite run. Every motion in a cell has positive probability of being selected, which makes the number of selections of each motion in the tree of motions be infinite as well. By the completeness of `PDST` [13], `KPIECE` is likely to be probabilistically complete. A formal proof is left for future work.

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

Acknowledgements

This work was supported in part by NSF IIS 0713623 and Rice University funds. The experiments were run on equipment obtained by NSF CNS 0454333 and NSF CNS 0421109 in partnership with Rice University, AMD and Cray. The authors would like to thank Mark Yim and Jonathan Kruse for their help in defining the CKBot `ODE` model, Marius Şucan for drawing the example discretization and Mark Moll, Konstantinos Tsianos and Nurit Haspel for reading the paper and providing valuable comments.

References

1. N. M. Amato and L. K. Dale. Probabilistic roadmap methods are embarrassingly parallel. In *IEEE Intl. Conf. on Robotics and Automation*, pages 688–694, Detroit, Michigan, USA, May 1999.
2. J. Barraquand, L. E. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for robot path planning. *Intl. Journal of Robotics Research*, 16(6):759–774, 1997.
3. S. Caselli and M. Reggiani. Randomized motion planning on parallel and distributed architectures. *Proceedings of the Seventh Euromicro Workshop on Parallel and Distributed Processing*, pages 297–304, February 1999.
4. 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.
5. I. A. Şucan, J. F. Kruse, M. Yim, and L. E. Kavraki. Kinodynamic motion planning with hardware demonstrations. In *Intl. Conf. on Intelligent Robots and Systems*, pages 1661–1666, September 2008.

6. D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *Intl. Journal of Robotics Research*, 21(3):233–255, March 2002.
7. D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *IEEE Intl. Conf. on Robotics and Automation*, volume 3, pages 2719–2726, April 1997.
8. L. Jaillet, A. Yershova, S. M. LaValle, and T. Siméon. Adaptive tuning of the sampling domain for dynamic-domain rrts. In *Intl. Conf. on Intelligent Robots and Systems*, 2005.
9. L. E. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. *Journal of Computer and System Sciences*, 57(1):50–60, 1998.
10. L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, August 1996.
11. J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE Intl. Conf. on Robotics and Automation*, 2000.
12. A. Ladd and L. Kavraki. Measure theoretic analysis of probabilistic path planning. *IEEE Transactions on Robotics and Automation*, 20(2):229–242, 2004.
13. A. M. Ladd. *Direct Motion Planning over Simulation of Rigid Body Dynamics with Contact*. PhD thesis, Rice University, Houston, Texas, December 2006.
14. A. M. Ladd and L. E. Kavraki. Fast tree-based exploration of state space for robots with dynamics. In *Algorithmic Foundations of Robotics VI*, pages 297–312. Springer, STAR 17, 2005.
15. J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.
16. J.-C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *Intl. Journal of Robotics Research*, 18(11):1119–1128, 1999.
17. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.
18. S. M. LaValle and J. Kuffner. Rapidly-exploring random trees: Progress and prospects. *New Directions in Algorithmic and Computational Robotics*, pages 293–308, 2001.
19. S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Intl. Journal of Robotics Research*, 20(5):378–400, May 2001.
20. E. Plaku, K. E. Bekris, and L. E. Kavraki. OOPS for Motion Planning: An Online Open-source Programming System. In *IEEE Intl. Conf. on Robotics and Automation*, pages 3711–3716, Rome, Italy, 2007.
21. E. Plaku, M. Y. Vardi, and L. E. Kavraki. Discrete search leading continuous exploration for kinodynamic motion planning. In *Robotics: Science and Systems*, pages 3751–3756, Atlanta, Georgia, 2007.
22. G. Sánchez and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. *Intl. Journal of Robotics Research*, pages 403–407, 2003.
23. J. Sastra, S. Chitta, and M. Yim. Dynamic rolling for a modular loop robot. *Intl. Journal of Robotics Research*, 39:421–430, January 2008.
24. R. Smith. Open dynamics engine. http://www.ode.org.
25. K. I. Tsianos, I. A. Şucan, and L. E. Kavraki. Sampling-based robot motion planning: Towards realistic applications. *Computer Science Review*, 1(1):2–11, August 2007.