Algorithmic Foundations of Robotics VI, Springer, STAR 17, p. 297–312, 2005

Fast Tree-Based Exploration of State Space for Robots with Dynamics

Andrew M. Ladd and Lydia E. Kavraki

Rice University, Houston TX 77005, USA

Abstract. This paper presents a new motion planning algorithm which we call the Path-Directed Subdivision Tree exploration planner (`PDST-EXPLORE`). It is a sampling-based method which builds a tree and takes a substantially different approach from other exploration planners such as `RRT` [18] and `EST` [12]. `PDST-EXPLORE` is a general purpose planner but is designed to overcome difficulties inherent in planning for robots with non-trivial dynamics. Specifically, our planner represents samples as path segments rather than individual states and uses non-uniform subdivisions of the state space to estimate coverage. This change avoids many of the problems that previous sampling-based planners have had with milestone placement, metrics and coverage estimation. We use a deterministic update schedule together with randomized path generation to adaptively strike a balance between greedy exploration and methodical search. We have obtained a proof of probabilistic completeness for the planner which assumes very little about the specific robot system that the planner operates on. Finally, we have implemented the planner for planar kinodynamic point robots, differential drive robots and blimp-like robots. The experimental results demonstrate the efficiency of the planner's implementation as well as its robustness in covering the entire reachable free space.

1 Introduction

This paper treats the problem of developing efficient planning techniques for robot systems with non-trivial dynamics operating in the presence of obstacles [18]. There are many interesting examples of dynamic motion planning problems such as car-like robots [16,3], tractor-trailer robots [16], snakeboard [19], helicopter robots [10], and spaceship robots [7]. We present a general purpose planner that is designed to address the challenges posed by robot systems with non-trivial second-order dynamics. In these cases, planning is complicated by constrained motion, drift and the importance of minimizing a path cost such as time or fuel usage. This paper relates the design and implementation of the Path-Directed Subdivision Tree exploration planner (`PDST-EXPLORE`). It is a tree-based planner which departs from previous frameworks, such as Rapidly-Exploring Random Trees `RRT` [18] and Expansive Spaces Trees `EST` [12], in order to avoid well-known weaknesses of these approaches [4,1].

There are various techniques applicable to dynamics planning problems in the literature. We group these into several families: exact methods, path

conversion methods and sampling-based methods. Exact methods which produce optimal paths are known but are **PSPACE** [9]. Polynomial time algorithms that produce approximately optimal paths are known for acceleration bounded point robots and for curvature constrained planning [20]. These exact algorithms produce optimal or approximately optimal solutions, however the main barrier to usage is that they are not practical algorithms and are only applicable to specific instances. Path conversion techniques make use of structure [2] to lift solutions obtained for simplified systems to feasible solutions for the given problem [22,21]. Although path conversion methods often offer a computationally efficient way of generating dynamic motions, the resulting paths tend to be of low quality, having many redundant or stopping motions. In the area of sampling-based planners, we divide existing planners into two categories: roadmap methods, such as the Probabilistic Roadmap (`PRM`) planner, and tree-based planners. The roadmap family of planning techniques [13] use global sampling and local planners to build roadmaps. In the dynamics case, implementing the local planner primitive, as it used in `PRM`, requires solving the steering problem [6,15], which is typically computationally expensive. The `PRM` approach offers a probabilistically complete solution [14] but the large number of calls to the steering primitive is prohibitive. Finally, tree-based planners, such as the (`RRT`) planner [18,5] and the (`EST`) planner [12], make use of forward integration of controls (propagation) to incrementally construct trees in the state space (exploration planning). Typically, steering primitives are then used to connect pairs of trees (connection planning). The tree-based methods enjoy the computational advantage of the relatively cheap propagation primitive and are thus the starting point for the research developed in this paper.

We choose to focus on improving the basic instance of a tree-based planner: the exploration planner. The purpose of an exploration planner is to capture the structure of the reachability set of the root state by incrementally constructing a tree of paths beginning from the given root. Exploration planners are usually employed as primitives in larger schemes, such as in bidirectional planning [18] or for the Probabilistic Roadmap of Trees planner [1], and dominate the run-time [18]. Our objective is to improve the state-of-the-art of exploration planning for dynamic planning problems. Exploration planners for such systems, namely `RRT` and `EST`, incrementally construct a tree in reachability space by alternating between two actions, selection and propagation, at each iteration of the planner. We refer to this design as the select-propagate architecture. In the selection phase, a state from the sample set is chosen using an algorithmic procedure guided by a data structure which is constructed and maintained on-line. This data structure is designed to estimate the current coverage of the sample set and to bias the sampling toward areas of the space which have few or no samples. Propagation generates new branches in the tree extending from the state chosen by the selection phase.

PDST-EXPLORE is a general purpose exploration planner but focuses on the case of motion planning with second-order dynamics as it is particularly challenging for exploration planners [4]. It is these challenges that inspired the PDST-EXPLORE planner and motivate its design. The efficiency of exploration planners is strongly tied to the implementation of the selection phase during expansion. The RRT approach uses Voronoi bias computed with proximity queries to guide expansion [18] and the EST approach maintains local density estimates to achieve the same [12]. Both approaches determine a probability distribution over the set of all current samples. The randomized (or quasi-random) selection approach is the foundation of probabilistic (or resolution) completeness for both methods. During propagation it is usually desirable to create multiple new samples along the propagated path (checkpointing) [4,12]. It is now well-known that the performance of these methods when applied to dynamic planning is very sensitive to the metric that determines proximity or density [4]. In particular, the drift present in second-order systems invalidates inherent assumptions in the way the metric is used [4]. Although specific remedies to this difficulty have been proposed [4], the metric sensitivity issue still impacts the efficiency of the planner. It has been empirically observed that RRT and EST enjoy rapid initial convergence but global convergence is much slower [1]. We posit that this is due to the randomized (or quasi-random) selection schedule leading to redundant growth as coverage increases.

PDST-EXPLORE preserves the features of exploration planning that have been successful: the use of cheap propagation primitives and the select-propagate architecture for growing the tree. Our contribution is a new selection algorithm and corresponding introspection data structure which are radically different from the ones typically employed in sampling-based planning. Namely, we propose the following design changes:

1. The samples are path segments and not states.
2. Metrics are not employed during selection and their properties are not used to prove completeness.
3. We use a deterministic, greedy selection schedule.

Outline This paper is organized as follows: in Section 2 we define the motion model that we use and state the problem that we are addressing. In Section 3, we describe the PDST algorithm and state a theorem about probabilistic completeness. We have tested the efficiency of our planner on the planar kinodynamic point robot, the differential drive robot and a blimp robot in 3-D. The planar workspaces we used are maze-like environments with features of various scale and character. Such workspaces are very challenging for existing planners [18,12,4,5]. Our experiments and results are detailed in Section 4, together with a novel application of Maneuver Automata theory to trajectory generation [11]. We conclude in Section 5 with a brief discussion.

2 Problem Statement

The state space of the robot is denoted by Q and is a smooth m-dimensional manifold. Let U be the set of controls for the robot. The motion of the robot system is governed by a differential equation. This equation is of the form $\dot{q} = f(q, u)$ for $q \in Q$ and $u \in U$. The function f is smooth in q. In addition to the constraint imposed by f, the motion of the robot may be further constrained by inequality constraints of the form $g(q, \dot{q}) \leq 0$ where g is smooth. Any path through Q that satisfies these constraints is *feasible*. In addition to the dynamic constraints, there are constraints determined by obstacles in Q. The set of states which are free is an m-dimensional submanifold of Q which is denoted Q_{free}. A state $q \in Q_{\text{free}}$ is a free state and state $q \notin Q_{\text{free}}$ is in collision. A feasible path which lies entirely in Q_{free} is said to be *collision-free*.

We give a formal definition of an exploration planner which is convenient for exposition and to state our theorem of probabilistic completeness. We define the exploration planning problem as follows: given $q_0 \in Q_{\text{free}}$ and $A \subset Q_{\text{free}}$, the exploration planner must compute a collision-free path from q_0 to any $q_f \in A$. This formulation captures how `RRT` and `EST` can be used a uni-directional planners and allows to analyze coverage of the reachability set of q_0.

3 The `PDST-EXPLORE` Planner

Like `RRT` and `EST`, `PDST-EXPLORE` is a sampling-based planner which uses select-propagate architecture to incrementally construct a tree. However, we propose radical changes to the way samples are represented and to the implementation of the selection algorithm. We will begin this section by giving a brief overview of and introduction to the operation of our planner.

In exploration planners to date, the sample set is represented as a set of states connected by paths defining a tree. In the `PDST-EXPLORE` planner, we represent the samples as a set of paths which are connected at branch states. By switching to this representation, the selection algorithm adaptively places branch states as needed. This seemingly superficial change avoids the need for check-pointing and drastically reduces the number of stored sample objects.

Previous exploration planners construct and maintain a biased probability distribution over the samples and then select the state to propagate from by randomly selecting from this distribution. The distribution bias is determined by proximity [18] or density [12]. In `PDST-EXPLORE`, priorities are assigned to each sample. The priority of a given sample is initialized to the iteration number the sample was created on and doubles each time that sample is selected. The selection algorithm for `PDST-EXPLORE` selects the sample with the lowest weighted priority at each iteration. The weighting for the priority are determined by a partition of the state space into cells. At the end of iteration, one

cell is subdivided and samples contained in that cell are subdivided as well. The cell subdivision rule, priority scheme and sample growth are designed to satisfy probabilistic completeness, to cover the reachability set greedily and to support efficient implementation.

3.1 Space and Sample Representation in PDST-EXPLORE

In this subsection we describe how the PDST-EXPLORE planner represents the state space and the samples. The space representation is a non-uniform cell partition of state space which is used to estimate coverage and to guide expansion. The samples are collision-free path segments which define a tree.

Definition of Subdivision Scheme PDST-EXPLORE maintains a complete subdivision of the state space into a set of cells. The subdivision is refined after each iteration. Each cell has a volume. Refining a cell creates two new cells, the union of which is the original cell, each with non-zero volume. This final property is necessary for our proof of correctness. A straightforward implementation of the cell partition can be obtained with a Binary Space Partition Tree [8].

Formally, a cell C is a subset of Q. The measure or volume of a cell is given by $\mu(C)$. The function μ is a measure in the formal sense and is normalized so that $\mu(Q) = 1$. A subdivision, S, is a finite partition of Q into cells of non-zero volume, $S = \{C_1, ..., C_K\}$. Cell subdivision is deterministic and consists of splitting a cell C into two parts, $\texttt{left}(C)$ and $\texttt{right}(C)$, which form a disjoint partition of C and each have non-zero measure. A subdivision S is refined by subdividing a single cell. The measure μ and refinement rules are called a subdivision scheme. It turns out that the subdivision scheme need not satisfy any additional properties beyond the ones we have detailed in order to permit the proof of probabilistic completeness. For this reason, the choice of the subdivision scheme can be made to optimize the efficiency of the planner.

Definition of Sample (Mass) A change proposed in this paper is to use paths rather than states as samples. We introduce the term *mass* to refer to a sample consisting of a non-empty path segment together with some additional structure. Masses also have a non-negative scalar *weight* and can be subdivided. It is possible to generalize mass to mean arbitrary subsets of state space but we eschew this in favor of clarity of exposition.

A mass is a collision-free path π with given duration T. Since the masses define a tree structure we need to define the root of the tree and the path from the root to each state contained in a mass. Specifically, every mass M has a root state $\texttt{root}(M) \in Q$ and associated with a path from $\texttt{root}(M)$ to each state along π. If q is state along the path π, we write $\texttt{path}(M, q)$ for the collision-free sub-path from $\texttt{root}(M)$ to q.

Each mass M has a weight $\nu(M) \geq 0$. The function ν is a measure in the formal sense. In our implementations, we have used $\nu(M) = |\pi| = T$, the duration of the path. Other notions of length or measure could be used. If π' is a sub-path of the path π, then the mass $M' = \pi'$ is a sub-mass of M if $\texttt{root}(M) = \texttt{root}(M')$ for any state q along both π and π', $\texttt{path}(M, q) = \texttt{path}(M', q)$.

Propagation The `propagate` operation creates a new mass by randomly extending from an existing mass. In our experiments, we have implemented `propagate` using two operations: `choose` and `generate`. The `choose` operation chooses a random state from the mass. For mass $M = \pi$ with duration T, $\texttt{choose}(M)$ generates a random parameter t uniformly from $[0, T]$ and returns $\pi(t)$. The operation `generate` creates a new mass by integrating a randomly chosen control function starting from the given state. Let M be a mass and let q be the result of $\texttt{choose}(M)$, the call to $\texttt{generate}(M, q)$ chooses $\kappa : [0, \infty] \to U$ from a distribution of control functions and integrates it starting from q to obtain a new path. The design of such distributions of control functions has been described in the literature [18,12,10,5]. The new path is collision detected and a collision-free sub-path, π', is obtained. The new mass M' is created from π' and the canonical path to $\texttt{root}(M)$ is constructed from the path from the root to q, $\texttt{path}(M, q)$.

3.2 `PDST-EXPLORE` Description

In this subsection, we describe the operation of the `PDST-EXPLORE` algorithm given as Algorithm 1. We split the pseudo-code of the algorithm into three parts: initialization (lines 1-4), select and propagate (lines 6-9) and the update code (lines 10-30). As we describe the operation of Algorithm 1, we will detail the calculation of various quantities and the data invariant maintained during the execution. Certain aspects of the interaction between the priority scheme, selection and space partitioning might seem somewhat arbitrary. These design decisions were made for two reasons: observed experimental efficiency and to permit the proof of Theorem 1.

Initialization and Invariants We now describe the initialization phase of `PDST-EXPLORE` (lines 1-4). An input of q_0, A and N is provided. The state $q_0 \in Q_{\text{free}}$ will be the root state for all masses created by the run. The subset $A \subset Q_{\text{free}}$ is the goal set for the run and planner will halt if a generated mass has non-empty intersection with A. The integer N determines the number of iterations that the algorithm runs for. Before we continue, some key invariants need to be discussed to shed light on the design of `PDST-EXPLORE`.

The algorithm maintains a few structures during the run. Fundamentally, these are a subdivision of Q called S, a finite set of masses written $\mathbf{M}$ and

Algorithm 1 PDST-EXPLORE(q_0, A, N)

```
 1: Set the subdivision to S = {Q}.
 2: Create a singleton mass M_0 = {q_0}.
 3: Set the mass set M = {M_0}.
 4: Set priority(M_0) = 1.
 5: for Each iteration = 1, ..., N do
 6:    Choose C_s ∈ S, the cell with the lowest priority.
 7:    Choose M_s ∈ masses(C), the mass in C_s with the lowest priority.
 8:    Set q to choose(M).
 9:    Set M_c to generate(M_s, q).
10:    if M_c = ∅ then
11:       priority(M_s) := 2(priority(M_s) + iteration).
12:    else
13:       if M_c ∩ A ≠ ∅ then
14:          Take q_f ∈ M_c such that q_f ∈ M_c.
15:          return path(M_c, q_f).
16:       end if
17:    else
18:       priority(M_s) := 2(priority(M_s) + 1).
19:       while M_c ≠ ∅ do
20:          C_next := stab(S, M_c)
21:          M_r := C_next ∩ M_c.
22:          M_c := M_c − M_r.
23:          if not the first time through and if density(C_next) > avgdensity(S)
             then
24:             break out of the while loop.
25:          else
26:             M := M ∪ {M_r}.
27:             priority(M_r) := iteration.
28:          end if
29:       end while
30:       Subdivide cell C_s.
31:    end if
32: end for
33: return FAILED.
```

a positive integer priority for each $M \in \mathbf{M}$ denoted $\texttt{priority}(M)$. We discuss low-level implementations for these structures later in this section. The subdivision S and $\mathbf{M}$ always satisfy the invariant property that for every $M \in \mathbf{M}$, there is a unique $C \in S$ such that $M \subset C$. In other words, each mass element lies uniquely in a some cell of the subdivision. When a cell is subdivided, the masses contained in that cell are also subdivided.

During initialization, the structures are created and assigned trivial values. The subdivision is initialized to the trivial subdivision $S = \{Q\}$. A special singleton mass M_0 is created. We have $M_0 = \{q_0\}$ and is associated

with the trivial path at q_0. The mass set is initialized by $\mathbf{M} = \{M_0\}$. The initial priority of M_0 is set to 1.

Select and Propagate Selection and propagation occur on lines 6-9 and execute at the beginning of each iteration. The selection step chooses a mass M_s from the current mass set $\mathbf{M}$. The mass M_s has the lowest priority among all masses contained in the cell $C_s \in S$ with the lowest priority. The priority of cell C, $\texttt{priority}(C)$, is determined as a function of priorities of the masses contained in C and the volume of C. The intuition is that small cells are less important than large cells.

$$\texttt{priority}(C) = \frac{\min \{\texttt{priority}(M) : M \subset C \text{ and } M \in \mathbf{M}\}}{\mu(C)}.$$

If there are no $M \in \mathbf{M}$ such that $M \subset C$ then $\texttt{priority}(C) = \infty$.

After selecting the mass M_s, a new mass is propagated from the old mass. The new mass, which we call M_c, is created by calling a randomized operator $\texttt{propagate}(M_s)$. It is possible that $M_c = \emptyset$ in which case we say that propagation failed.

Update After selection and propagation occur, the mass set, priorities and subdivision are updated. The update takes place on lines 10-30. There are three cases: if $M_c = \emptyset$, if $M_c \cap A \neq \emptyset$ and the final case where M_c is incrementally inserted into $\mathbf{M}$ according to S.

If the call to `propagate` fails ($M_c = \emptyset$) then the priority of M_s is adjusted to $2(\texttt{priority}(M_s) + \texttt{iteration})$. The addition of the iteration counter is to penalize the rate of selection for that mass beyond the normal doubling scheme since propagation failed. If $M_c \cap A \neq \emptyset$ then there is $q_f \in M_c$ such that $q_f \in A$. The exploration planner has succeeded and can return $\texttt{path}(M_c, q_f)$.

If the propagation succeeded but M_c did not intersect A, then M_c is partitioned according to S and incrementally inserted into $\mathbf{M}$. The insertion proceeds until it reaches a cell of above average density. The density thresholding technique helps limit the creation of redundant samples. At least one insertion occurs regardless of density. The insertion loop is on lines 10-30. After the insertion is complete, the selected cell C_s is subdivided. Additionally, the priority of M_s is updated to $2(\texttt{priority}(M_s) + 1)$ to penalize its selection.

The density of a cell is determined by the weight of the masses in the cell and the volume of the cell. The density of a cell C is determined by the sum of the masses contained in a given cell divided by the volume of the cell

$$\texttt{density}(C) = \sum_{M \in \mathbf{M} : M \subset C} \frac{\nu(M)}{\mu(C)}.$$

The average density of the subdivision S, $\texttt{avgdensity}(S)$, is determined by cells in the subdivision which are non-empty $S_p = \{C \in S : \text{ there is } M \in$

$\mathbf{M}$ such that $M \subset C\}$. Specifically, the average density is the sum of the densities of the non-empty cells divided by the number of non-empty cells,

$$\texttt{avgdensity}(S) = \sum_{C \in S_p} \frac{\texttt{density}(C)}{|S_p|}.$$

The insertion loop proceeds by determining a cell C_{next} such that $M_c \cap C_{\text{next}}$ is non-empty and then inserting $M_i = M_c \cap C_{\text{next}}$ into $\mathbf{M}$. After the insertion, M_c is set to the remainder $M_c - M_i$ and the loop proceeds. After the first insertion, the insertion loop ends if $M_c = \emptyset$ or if $\texttt{density}(C_{\text{next}}) > \texttt{density}(S)$. The choice of C_{next} is determined by operation called `stab` such that $C_{\text{next}} = \texttt{stab}(M_c, S)$. The stabbing operation finds the cell containing the first state in M_c, i.e. if $M_c = \pi$ then $\texttt{stab}(M_c, S)$ returns the cell $C \in S$ such that $\pi(0) \in C$. This operation is well-defined since the cells in S are a partition of the state space. After insertion has completed, the cell C_{next} is subdivided. The cell C_{next} is removed from the subdivision is removed from S and the cells $C_l = \texttt{left}(C_{\text{next}})$ and $C_r = \texttt{right}(C_{\text{next}})$ are added. Then each $M \in \mathbf{M}$ that was contained in C_{next} is split into $M_l \subset C_l$ and $M_r \subset C_r$. The mass M is discarded and M_l and M_r are added to $\mathbf{M}$. Empty masses are discarded.

Data Structures Obtaining a general implementation of `PDST-EXPLORE` is fairly straightforward. The mass representation, subdivision scheme and their associated operations can be made abstract. A generic binary space partition tree is then the primary data structure and the masses are referenced by the leaf nodes (cells). A hash table is used to store priorities and the cell priorities and densities are stored at each cell. All non-empty cells are placed in a priority queue and sorted by increasing priority. We use binary heap with hash table back-pointers to implement the priority queue and its operations. The stabbing operations are implemented top-down. The algorithmic overhead in `PDST-EXPLORE` at the mth iteration is proportional to $O(DS+\log(m))$, where D is the maximum depth of the tree, S is the number of stab operations and $\log(m)$ is incurred by the binary heap. In practice, we have observed that the bulk of the run-time is spent in path generation and collision detection.

Probabilistic Completeness The general formulation of `PDST-EXPLORE` that we provide supports an abstract proof of probabilistic completeness similar to one made for `PRM` [14]. We begin by defining a simple random walk exploration planner which we call `RANDOM-WALK-EXPLORE`. Given input (q_0, A), the planner begin by setting M to singleton mass $\{q_0\}$. At each iteration: if M intersects A then return success and report the path, otherwise set $M := \texttt{propagate}(M)$. If $M = \emptyset$ then return failure otherwise loop again. We can now state the Theorem below.

Theorem 1. *For a given robot system and given a formulation for the mass and the subdivision scheme, the exploration planner* `RANDOM-WALK-EXPLORE` *succeeds with strictly positive probability on an input* (q_0, A) *if and only if the exploration planner* `PDST-EXPLORE` *eventually succeeds on input* (q_0, A) *with probability* 1.

Although we do not have space to present the proof of this Theorem as it is fairly involved, we can briefly describe the argument. For the input (q_0, A), both exploration planners can only succeed with strictly positive probability if there a sequence of calls to `propagate` which creates mass in A with strictly positive probability. Using measure-theoretic techniques similar to those applied to our analysis of `PRM` [14], we prove the existence of a finite stationary sampling sequence. It remains to show that `PDST-EXPLORE` eventually makes progress, thus decreasing the length of stationary sequence required to generate mass in A by one. Then, applying induction, we can conclude that the planner succeeds. The technical difficulties in the proof arise as a result of the deterministic selection schedule and the subdivision of the masses and cells. Since the effect of random results to `propagate` on the sequence of mass selections is difficult to judge directly, a non-deterministic model of `propagate` can be taken and it can be shown that some sub-mass of any generated mass is selected infinitely often over the course of an infinite run. From this observation, the rest of proof can be constructed after formalizing the effect of subdivision on the masses.

4 Experimental Results

In this section, we begin by describing a novel extension to Maneuver Automata theory [11] which uses `PRM` sampling to generate Maneuver Automata. The generated Maneuver Automata are then used for trajectory generation. We continue by outlining the specific robot systems we have implemented `PDST` for. Finally, we describe our experiments and present our results.

4.1 Maneuver Automata

We propose using the Maneuver Automata [11] to implement the `generate` function used in our exploration planner. This is novel extension to the Maneuver Automata literature and can applied to any planner that uses propagation for a robot that satisfies a certain symmetry property. The advantage we gain is that we can restrict ourselves to nice family of motions and eliminate numerical integration from the call to `generate`. These techniques only apply when special structure exists in the motion of the robot, however the class of applicable robot systems is an important one. A sufficient condition occurs when the state space Q is a direct product of a Lie Group G and a

shape manifold Z. Furthermore, the Lie Group G operating on Q must preserve path feasibility [11]. If this is the case, we say that G is a symmetry group for Q and Maneuver Automata theory can be applied.

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

The graph $MA = (V, E)$ is a kind of roadmap in shape space. Continuing this analogy, we can sample MA using `PRM` techniques. So, given $z_1, z_2 \in Z$, we need a local planning primitive which can compute a control function which defines a maneuver motion to connect z_1 to z_2 . The local planner and the sampling distribution over Z can be used to implement `PRM`. It seems likely that Quasi-Random-Lattice methods would be particularly effective for this task [17].

4.2 Robot Systems

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

2-D Kinodynamic Robot The state space for the 2-D kinodynamic robot is denoted by $Q = \mathbb{R}^2 \times \mathbb{R}^2$. A state $q = (x, y, \dot{x}, \dot{y})$, where (x, y) is the robot's position in the plane and $(\dot{x}, \dot{y})$ is the robot's velocity. The symmetry group we use for this robot is $G = \mathbb{R}^2$, the group of 2-D translations. The shape manifold for the robot is $Z = \mathbb{R}^2$ and represents positionless velocities. Every $z = (\dot{x}, \dot{y}) \in Z$ defines a trim primitive α_z and corresponds to the straight line at that fixed velocity. Maneuvers between trim primitives consist of the robot acceleration toward the different velocity state at maximum acceleration.

We impose a constraint that velocity is bounded, i.e. $||(\dot{x}, \dot{y})|| \leq v_{\max}$. The subdivision scheme builds a *kd*-tree in the state space by equal splits on the first and second dimensions.

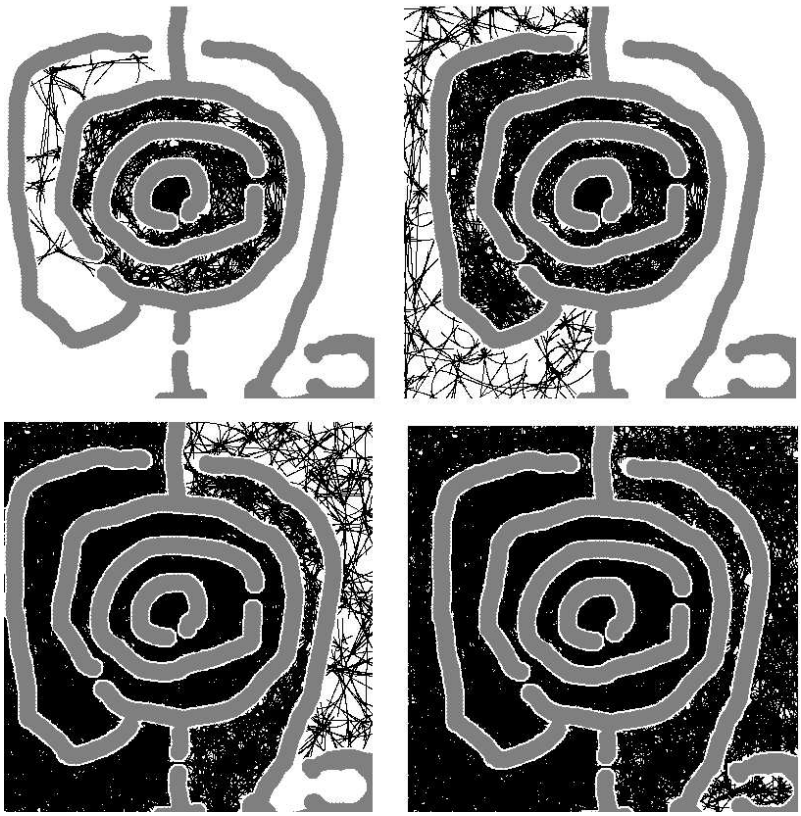

Fig. 1. Execution snapshots of `PDST-EXPLORE` for a differential drive robot

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

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

where $\omega = \frac{v_r - v_l}{L}$, $v_f = \frac{v_l + v_r}{2}$ and L is the length of the wheel base of the robot. Let $z^1 = (v_l^1, v_r^1)$ and $z^2 = (v_l^2, v_r^2)$. The maneuver primitive that

brings z^1 to z^2 is $[\pi_{z^1 z^2}]$. It is determined by $a_{\max}$, the maximum wheel acceleration for the robot. Let

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

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

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

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

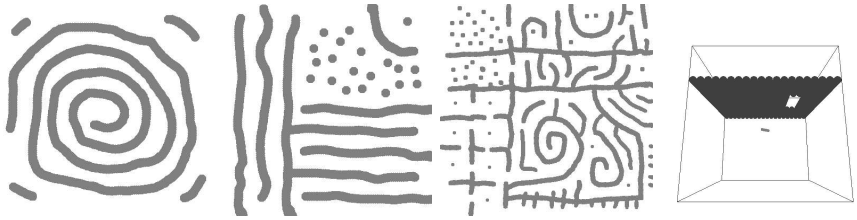

Fig. 2. Workspaces (from left to right) spiral-1, varied-1, varied-2 and slot

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

4.3 PDST Experiments

In Figure 2, we depict some of the workspaces we used for the experiments. In each experiment, a maneuver automata was built offline which took less than two seconds for the kinodynamic robot and differential drive robot and between 50 and 70 seconds for the blimp example. During the experiment, the maneuver automata was loaded off the disk and the PDST-EXPLORE planner was run until the measured dispersion [17] in the free space became very small. Dispersion was measured on a high-resolution cell grid. Cells containing an obstacle were not considered in the dispersion measure. The threshold we used ensured that over 0.999 of the space was covered. The number of iterations was then reported. In every example, 384 trials were carried out. The number of iterations required to solve the problem tended to be very similar to the mean number of iterations with the occasional outlier requiring between two and six times more iterations. In Figure 1, we show snapshots of the execution of the exploration of the free space for a differential drive robot in the workspace chambers-1. The time costs and collision detect calls were very consistent across multiple runs. The raw data is presented in Figure 3. Cost in time per iteration is roughly $O(n \log n)$ experimentally, which is expected because of the binary heap.

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

Fig. 3. Average running times to obtain full coverage.

5 Discussion

In this paper, we have presented the PDST-EXPLORE algorithm, stated its probabilistic completeness theorem, discussed motion generation for planning using Maneuver Automata and then implemented our ideas for kinodynamic point robots in 2-D, second-order differential drive robots and a second-order blimp-like robots. In our experiments we have demonstrated that PDST-EXPLORE produces full coverage of the space efficiently. Many of our examples were for complicated and varied maze-like environments. In previous studies for planning with second-order dynamics that have reported run-times, times on the scale on a hour have been reported for finding a path

in simple environments using robot systems similar to those implemented for this paper [18,5].

There is a great of flexibility in how the PDST-EXPLORE planner can be applied that we have not evaluated in this paper. Time or other cost variables could be incorporated into the space to encourage cost optimization. Our preliminary experiments in this direction have been promising. The cell subdivision scheme, space measure and mass measure can be varied significantly without taking away completeness since they can be chosen to reduce running time by improving convergence speed and by reducing overhead of the geometric computations. For example, the use of non-uniform measures or cell subdivision in a projective space such as task space might be useful for some applications.

The PDST-EXPLORE is only one component of an efficient planning framework for robots with dynamics. We envision using the Probabilistic Roadmap of Trees (PRT) [1] to continue developing this framework toward general purpose single and multiple-query planners for dynamic systems. In pursuit of this goal, we plan on studying path optimality issues, connection planning and adapting PDST-EXPLORE to higher dimensional systems. Most importantly, we plan to apply our techniques to more complex dynamical robot systems to determine the limitations of our approach.

Acknowledgements

Work on this paper has been partially supported by NSF 0308237, NSF 0205671, NSF EIA 0216467, an FCAR fellowship to A. Ladd and a Sloan Fellowship to L. Kavraki. We also thank AMD for donating hardware. A. Ladd thanks S. LaValle, E. Frazzoli, P. Cheng, O. Brock and J. Phillips for valuable input on planning and dynamics. Finally, we thank E. Plaku, K. Bekris and D. Ruths for their feedback on the manuscript.

References

1. M. Akinc, K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plaku, and L. Kavraki. Multiple query probabilistic roadmap planning using single query primitives. In *Int. Symp. Robotics Research*, 2003.
2. F. Bullo, A. D. Lewis, and K. Lynch. Controllable kinematic reductions for mechanical systems: concepts, computational tools and examples. In *Mathematical Theory of Networks and Systems*, Notre Dame, IN, August 2002.
3. L. G. Bushnell, D. M. Tilbury, and S. S. Sastry. Steering three-input nonholonomic systems: the fire truck example. *IJRR*, 14(4):366–381, 1995.
4. P. Cheng and S. LaValle. Reducing metric sensitivity in randomized trajectory design. In *IROS*, 2001.
5. P. Cheng and S. LaValle. Resolution-complete rapidly-exploring random trees. In *ICRA*, 2002.
6. P. Cheng and S. LaValle. Exploiting group symmetries to improve precision in kinodynamic and nonholonomic planning. In *IROS*, 2003.

7. P. Cheng, Z. Shen, and S. M. LaValle. RRT-based trajectory design for autonomous automobiles and spacecraft. *Control Sciences*, 11(3-4):51–78, 2001.
8. M. de Berg, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 1997.
9. B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *J. of the ACM*, 40:1048–1066, 1993.
10. E. Feron, E. Frazzoli, and M. Dahleh. Real-time motion planning for agile autonomous vehicles. In *AIAA Conf. on Guidance, Navigation and Control*, 2000.
11. E. Frazzoli. Maneuver-based motion planning for non-linear systems with symmetries. Submitted to Transactions on Robotics and Automation.
12. D. Hsu, R. Kindel, J. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *IJRR*, 2001.
13. L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *TRA*, 12(4):566–580, June 1996.
14. A. Ladd and L. Kavraki. Measure theoretic analysis of PRM. *TRA*, pages 222–229, 2004.
15. F. Lamiraux, E. Ferre, and E. Vallee. Kinodynamic motion planning: Connecting exploration trees using trajectory optimization methods. In *ICRA*, 2004.
16. J. P. Laumond. Feasible trajectories for mobile robots with kinematic and environment constraints. In *Preprints of the International Conference on Intelligent Autonomous Systems*, pages 346–354, Amsterdam, The Netherlands, 1986. Elsevier Science Publishers, B.V.
17. S. LaValle, M. Branicky, and S. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *International Journal of Robotics Research*, 2003.
18. S. LaValle and J. Kuffner. Rapidly exploring random trees: Progress and prospects. In B. Donald, K. Lynch, and D.Rus, editors, *WAFR*, pages 293–308. A.K. Peters, 2001.
19. A. Lewis, J. P. Ostrowski, R. M. Murray, and J.W.Burdick. Nonholonomic mechanics and locomotion: the snakeboard example. In *ICRA*, pages 2391–2400, May 1994.
20. J. Reif and H. Wang. Non-uniform discretization approximations for kinodynamic motion planning. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 97–112. A K Peters, Wellesley, MA, 1997.
21. S. Sekhavat, P. Svestka, J.-P. Laumond, and M. H. Overmars. Multilevel path planning for nonholonomic robots using semiholonomic subsystems. *IJRR*, 17:840–857, 1998.
22. P. Švestka and M. Overmars. Motion planning for car-like robots using a probabilistic learning approach. *Int. J. of Robotics Research*, 16(2):119–143, 1997.