16-741: Mechanics of Manipulation – Paper Review

Randomized Kinodynamic Planning

Steve M. LaValle & James J. Kuffner

Philipp Michel
pmichel@cs.cmu.edu

Abstract

LaValle & Kuffner [12] present an application of a randomized technique to the problem of kinodynamic planning. Their algorithm constructs Rapidly-exploring Random Trees (RRTs) in a high dimensional state space that encompasses both first order constraints resulting from the physically-based system dynamics as well as global kinematic constraints due to obstacles in the environment. By coupling and simultaneously addressing the problems of finding a basic path in configuration space and finding a controller satisfying the system dynamics while exploiting the benefits of randomized techniques, trajectories for a broad range of systems operating in cluttered environments can be computed in an efficient manner.

1 Introduction

Most traditional approaches to path planning decouple the problem into two components. They first perform basic path planning in configuration space, ignoring the system dynamics. Subsequently, during so-called smoothing, they seek to find a controller and system-specific trajectory that satisfies the dynamics and roughly follows the path computed previously [11]. As such, the actual path planning component is purely kinematic, with the system dynamics essentially being ignored during the planning stage.

This has several negative consequences. Firstly, such separation of kinematics and dynamics during planning means that it is possible for the kinematic path planner to generate paths that are in fact unexecutable by the system due to inherent control limits. Secondly, in order to guarantee a collision-free trajectory during execution, it may often in fact be necessary to explicitly model the system dynamics. This is exacerbated for systems particularly affected by natural physical laws as well as systems with significant limits on the available controls: helicopters, aircraft, wheeled vehicles, legged robots and underwater robots, among others, exhibit considerable constraints on the allowable velocities at each configuration.

These factors motivate the replacement of traditional configuration space planning techniques by a *kinodynamic* planner operating on a higher-dimensional *state space*. The state space notion as employed by LaValle & Kuffner consists of the usual configuration space augmented with velocity parameters, essentially comprising the tangent bundle of the configuration space. Kinodynamic planning then seeks to find control inputs that drive the robot on a trajectory in state space from an initial to a final configuration and velocity, all the while satisfying both global constraints and local differential constraints.

General kinodynamic planning, given the high-dimensional state space, has been proven to be at least PSPACE-hard [15], with exact, time-optimal trajectory planning in state space being NP-hard [7]. Randomized techniques have enjoyed considerable success in ameliorating computational complexity and enabling simple, efficient yet incomplete planning solutions to be devised for basic holonomic path planning problems.

LaValle & Kuffner's core contribution lies in the application of randomized techniques to the broader problem of kinodynamic planning. Their approach quickly explores state space and scales well as the system's degrees of freedom increase and its dynamics become more complicated. The remainder of this paper is organized as follows: Section 2 gives some pointers to previous work in kinodynamic planning and popular randomized planning techniques. Section 3 defines the notion of state space. Section 4 introduces Rapidly-exploring random trees and shows how they can be used for planning in state space. Results and some performance analysis is given in Section 5. We conclude in Section 6 and give some prospects of current and future work in the area of randomized kinodynamic planning.

2 Related Work

We briefly look at advances in the areas of kinodynamic planning and randomized planning techniques, which the work by LaValle & Kuffner seeks to fuse.

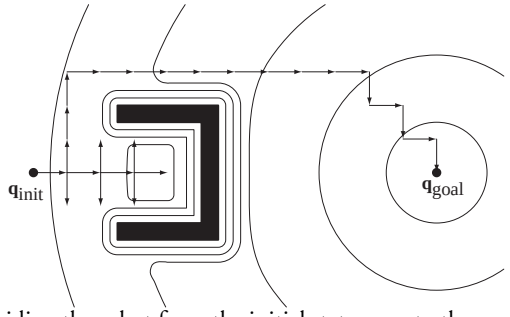

Figure 1: A potential field guiding the robot from the initial state q_{init} to the goal state q_{goal}. Figure from [14].

2.1 Kinodynamic planning

Sahar & Hollerbach [16] presented one of the earliest kinodynamic planning algorithms to find minimum-time trajectories in state space, employing dynamic programming to reduce the search complexity. It has been attempted to solve the trajectory finding problem algebraically, most notably by Canny et. al. [3], although such approaches have so far only been successful for point masses with low dimensional velocity and acceleration bounds. The first provably good approximation algorithm (running in polynomial time) to such point mass problems was presented by [7]. The same authors have suggested further generalizations, as in [8]. All these approaches share the problem of being restricted to rather low DOF and/or simplistic systems.

2.2 Randomized planning techniques

A technique that has been successful for holonomic path planning are randomized potential fields (e.g. [1]). As seen in class, they define a potential field on the configuration space using a heuristic function rewarding distance to the goal and penalizing closeness to obstacles, 'steering' the robot towards the goal. In addition, they use randomization in the form of random walks to avoid local minima in the potential field. Fig. 1 illustrates. Relying heavily on a good heuristic, this approach translates poorly to state space, where the heuristic has to encompass differential constraints in addition to obstacles. For highly dynamical systems, which often possess drift, potential fields can lead to oscillations unless the heuristic function is very carefully designed.

Probabilistic roadmaps (e.g. [9]) construct a graph by generating random configurations q in C-space that do not collide with obstacles and then using a local planner to connect pairs of nearby configurations. This connection step, while efficient in C-space for holonomic systems, is difficult in a state space formulation and for dynamical systems exhibiting drift, potentially requiring thousands of connections before a solution trajectory is found.

3 State Space and Planning

3.1 Formulation

Consider first the configuration space $\mathcal{C}$ arising from a rigid or articulated body, as it is usually formulated: every configuration $q \in \mathcal{C}$ represents the location of every point in a system of points existing in some ambient space. We augment this definition to arrive at the *state space* $\mathcal{X}$, in which a state $x \in \mathcal{X}$ is simply defined as:

$$x = (q, \dot{q})$$

Recall that, as seen in class, we can express holonomic constraints simply as a set of equations in just the configuration variables and possibly time, i.e. $g_i(q, t) = 0$, requiring no rate variables. Nonholonomic constraints require the use of rate variables and or inequalities, i.e. $g_i(q, \dot{q}, t) = 0$ or $g_i(q, \dot{q}, t) \leq 0$ (Fig. 2 gives an example).

Finally, differential constraints can be written in Lagrangian dynamics as a set of equations of the form $h_i(q, \dot{q}, \ddot{q}, t) = 0$, additionally involving acceleration. The use of the state space formulation allows us to represent these dynamic constraints as a set of m implicit equations of the form $G_i(x, \dot{x}) = 0$, $m < 2n$, $n = |\mathcal{C}|$. We can re-write these equations in the form

$$\dot{x} = f(x, u)$$

where $u \in U$ and U represents the set of allowable control inputs to the system. The equation thus describes the state transitions resulting from a control input.

The planning algorithm presented by LaValle & Kuffner assumes that, given the current state $x(t)$ and a sequence of controls u applied over some time interval Δt, we are able to compute the resulting state $x(t + \Delta t)$. One way to do this is

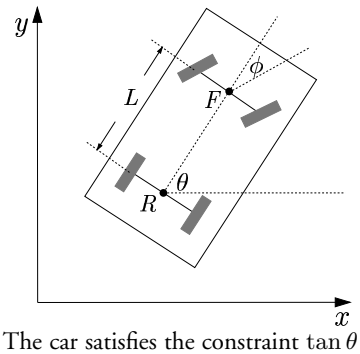

Figure 2: Example of nonholonomic constraint. The car satisfies the constraint $\tan\theta = \frac{\dot{y}}{\dot{x}}$. The equations describing the constraint are: $\dot{x} = v\cos\theta$, $\dot{y} = v\sin\theta$ and $\theta = (\frac{v}{L})\tan\theta$.

by numerical integration of the equation $\dot{x} = f(x, u)$, or by incrementally simulating the system, which yields a discrete-time approximation to the equation, such as $x_{k+1} = f(x_k, u_k)$.

3.2 Obstacles in state space

LaValle & Kuffner note an important distinction between the description of obstacles and free space in C-space and state space. In C-space planning, we assume that we have some way of detecting whether a particular configuration $q \in \mathcal{C}$ is in collision with an obstacle, i.e. $q \in \mathcal{C}_{obst}$. We wish to find a continuous path that maps into $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obst}$.

In state space, rather than defining $x \in \mathcal{X}_{obst} \iff q \in \mathcal{C}_{obst}$ for $x = (q, \dot{q})$, we also need to consider states in which the robot will inevitably collide with an obstacle due to momentum. In such cases, there exist *no* control inputs that can be applied over *any* length of time to avoid collision. This *region of inevitable collision* subsumes all collision states ($\mathcal{X}_{obst} \subseteq \mathcal{X}_{ric}$) and we should define $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{ric}$.

$\mathcal{X}_{ric}$ grows as the speed of the system increases and may ultimately look rather different from $\mathcal{X}_{obst}$. This makes finding a valid kinodynamic trajectory more difficult.

3.3 Finding a solution trajectory

Given this formulation, the goal of kinodynamic planning is to find a trajectory $x(t)$ from a start state $x_{init} \in \mathcal{X}$ to a goal state $x_{goal} \in \mathcal{X}$, that maps into $\mathcal{X}_{free}$. Ultimately, however, we seek to find a time-parameterized function of control inputs $u(t)$ that results in such a trajectory. Given $u(t)$, we can find $x(t)$ by integration of $\dot{x} = f(x, u)$. The high dimensionality makes finding an optimal solution is computationally infeasible, so we usually settle for solutions deemed 'good' by a loose heuristic.

4 Rapidly-exploring Random Trees

Motivated by the desire to construct a kinodynamic planning algorithm that requires no environmental preprocessing, is able to answer a single query efficiently and quickly and uniformly explores state space, LaValle & Kuffner introduce the idea of Rapidly-exploring random trees (RRTs).

4.1 Building RRTs

Suppose we are given a discrete time approximation to the differential state transition function: $x_{k+1} = f(x_k, u_k)$. We can easily construct a tree that slowly grows to cover state space as follows: take the root of the tree to be the initial state x_{init}. Repeatedly pick a random existing vertex x_k from the tree and a random control u_k from the set of available controls. Compute $x_{k+1} = f(x_k, u_k)$ and add x_{k+1} and an edge from x_k to x_{k+1} to the tree. While a tree constructed in this way will eventually cover all of state space, it grows very slowly and tends to be biased towards places it has already explored.

To build an RRT, we also start with x_{init} as the tree's root node. Now we pick an **arbitrary** point x_{rand} **at random** from the entirety of state space $\mathcal{X}$ and find the point x_k closest to it that already exists in the tree built thus far. We now choose a control input u_k and apply it over a time interval Δt deliberately such that x_k gets pulled as much as possible to x_{rand}. We finally insert a new vertex $x_{k+1} = f(x_k, u_k)$ and an edge to it from x_k into the tree.

This procedure causes the tree to rapidly explore the state space. Fig 3 compares the two approaches of building trees for a robot operating in discretized planar 2D. By choosing the vertex closest to x_{rand}, the algorithm selects vertices that have

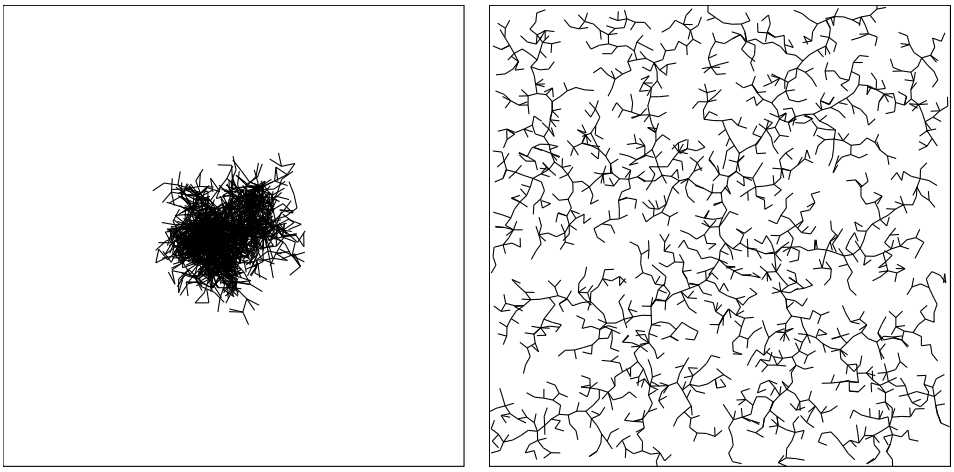

Figure 3: Two trees constructed for a simple discrete 100 × 100 grid in 2D. The purely random tree (left) is biased towards already visited states. The tree on the right rapidly explores the space.

Algorithm 1 BUILD_RRT(x_{init})

Require: $\mathcal{T}$ is an empty tree
  $\mathcal{T}$.init(x_{init});
  for $k = 1$ to K **do**
    $x_{rand} \leftarrow$ RANDOM_STATE();
    EXTEND($\mathcal{T}$, x_{rand})
  end for
  Return $\mathcal{T}$

Algorithm 2 EXTEND($\mathcal{T}$, x)

  $x_{near} \leftarrow$ NEAREST_NEIGHBOR(x, $\mathcal{T}$);
  if NEW_STATE($x, x_{near}, x_{new}, u_{new}$) **then**
    $\mathcal{T}$.add_vertex(x_{new});
    $\mathcal{T}$.add_edge($x_{near}, x_{new}, u_{new}$);
    if $x_{new} = x$ **then**
      Return *Reached;* (new vertex reaches the actual sample)
    else
      Return *Advanced;* (new vertex added to RRT)
    end if
  end if
  Return *Trapped;* (could not produce a state $\in \mathcal{X}_{free}$)

larger Voronoi regions, corresponding to unexplored space, causing the graph to spread uniformly. The two components of the algorithm that construct an RRT are given in Alg. 1 and Alg. 2.

The control input u_{new} that pulls the tree towards x_{rand} can be chosen by trying all inputs and choosing the one that yields the state closest to x_{rand} when applied for some time increment Δt, which can be fixed or randomly selected. The algorithm also assumes that NEW_STATE performs collision detection and only returns states that satisfy the global constraints, which can usually be done very efficiently.

4.2 RRTs and state space

Several additional concerns need to be taken into account when building RRTs in state space, as opposed to the trivial 2D examples given in Fig. 3. Besides the omnipresent drift in dynamical systems, the two main issues are the increased dimensionality of the space and the fact that a good metric for state space (needed to find x_{near} using nearest neighbor) may be considerably difficult to construct and very problem-specific. LaValle & Kuffner suggest the use of a carefully considered metric to some degree based on the optimal metric (which always exists but is at least as hard to compute as the solution to the planning problem itself) and the use of approximate nearest neighbor techniques (see [13] for an overview done by Andrew Moore's group) as possible ways to deal with these concerns. However, their original implementation uses a straightforward Euclidean metric and linear nearest neighbor search.

Algorithm 3 RRT_BIDIRECTIONAL(x_{init}, x_{goal})

$\mathcal{T}_a$.init(x_{init}), $\mathcal{T}_b$.init(x_{goal})
for $k = 1$ to K **do**
  $x_{rand} \leftarrow$ RANDOM_STATE();
  if EXTEND($\mathcal{T}_a$, x_{rand}) $\neq$ *Trapped*) **then**
    if EXTEND($\mathcal{T}_b$, x_{new}) = *Reached*) **then**
      Return PATH($\mathcal{T}_a$, $\mathcal{T}_b$);
    end if
  end if
  SWAP($\mathcal{T}_a$, $\mathcal{T}_b$);
end for
Return *Failure*;

4.3 Path planning with RRTs

A basic bidirectional trajectory RRT planning algorithm proceeds as follows. It grows two RRTs, rooted at x_{init} and x_{goal} until we reach a point where there is a state that is common to both trees (usually "common" is defined flexibly as the two states not being more than ϵ apart in state space under the metric used). The algorithm can optionally continue to run and return further, potentially better, solution trajectories in terms of some cost metric. The algorithm for planning is given in Alg. 3. It alternately extends one of the trees by a new state and then attempts to connect the nearest vertex of the other tree to that node.

The trajectory found this way may not be continuous at the junction of the two RRTs. There are several ways to overcome this, from perturbing the second half of the trajectory to enforce continuity to just growing one RRT, biased to grow towards the goal to some extent, until it reaches a state close to x_{goal}. This, however, incurs significant performance penalty when compared to the bidirectional method.

5 Results

We first briefly describe the dynamic model and experimental setup used in [12], followed by a summary of results LaValle & Kuffner report for their example systems of increasing complexity. Some results from further research on trajectory design using RRTs are also presented. Direct evaluation against other planning approaches is difficult due to the lack of exact bounds on the planner's complexity. Although RRT planning has defied analysis somewhat thus far, it has been shown that an RRT planner is guaranteed to find a solution trajectory (if one exists) as the number of vertices examined tends to infinity. Also, it can be proven that in the limiting case, an RRT will sample state space uniformly [12, 5].

5.1 Dynamic model

All example systems are modeled as rigid bodies with a given mass, inertia tensor and quantities: $\mathbf{p}$ (position of center of mass), $\mathbf{q}$ (unit quaternion representing rotation), $\mathbf{v}$ (linear velocity), ω (angular velocity), resulting in a 12D state vector $\mathbf{x}(t) = [\ \mathbf{p}(t)\, \mathbf{q}(t)\, \mathbf{v}(t)\, \omega(t)\ ]^T$. Each control is modeled as a force-torque pair $(\mathbf{F}, \tau)$, from which the equations of motion (in terms of $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$ can be derived.

The distance metric used to compare two states $\in \mathcal{X}$ is a weighted Euclidean one, normalized to lie in $[0, 1]$ and incorporating all four terms of the state vector, using the quaternion scalar product (cos of the angle between two orientations) to quantify orientation distance.

Controls u drawn from a system's set of allowable controls U (including drift), when applied, are applied constantly over a fixed time interval Δt. A simple Euler-method numerical integrator is used to recover trajectories. All systems are evaluated in a workspace of $10m$ for each axis with linear velocity bounds of $2\frac{m}{s}$ and angular velocity bounds of $1.5\frac{rad}{s}$.

5.2 Translation only

A rectangular planar translating body with four translational controls only (opposing forces through the c.o.g.), $|\mathcal{X}| = 4$ was used in a 2D maze-setting for path planning. The reported mean time to find a trajectory was 5 seconds, requiring RRTs of up to 2500 nodes. For this trivial task, it is clear that a kinodynamic planner carries more overhead than benefit and is easily outperformed by a purely kinematic planner. Kuffner reports (in [10]) planning times of < 0.2 seconds for a similar task on inferior hardware.

For a similar body translating in 3D with six force-only translational controls $|\mathcal{X}| = 6$ in a generalized maze in 3D (free floating objects and two narrow passages), a candidate trajectory is found after around one minute, requiring RRTs of several

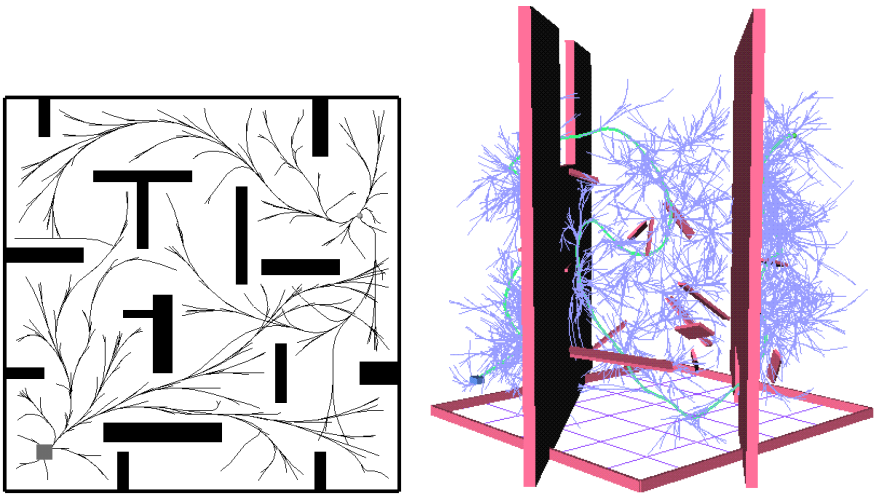

Figure 4: RRTs for the translation-only planning tasks in 2D (left) and 3D (right).

thousand nodes. Again, for this dynamically simplistic system, the kinodynamic planner is likely to be outperformed by purely kinematic planning.

Fig. 4 shows the scenarios and 2D projections of the RRTs for the translation-only case.

5.3 Translation and rotation

Extending the planar case above by allowing the body to rotate in place and translate forward only (c.f. a thruster on a spacecraft) results in a system with 3 controls (rotate cw, rotate ccw, forward, now foce and torque controls) and $|\mathcal{X}| = 6$. A trajectory for the maze planning task was found in around 5 minutes using RRTs of several thousand nodes.

For the most general case of translation and rotation in 3D, albeit for an underactuated system allowing orientation along any axis but translation only along the primary axis of the system (two opposing thrusters), a twelve dimensional state space results ($|\mathcal{X}| = 12$). For a simulated docking task in space amidst a number of obstacles, a solution trajectory was found in around 6 minutes using RRTs of over $20,000$ nodes.

Fig. 5 shows the projected RRTs for these cases. For an aerial vehicle allowing forward, up, down, clockwise and counter-clockwise roll flying through narrow openings into a hangar, the algorithm requires 12 minutes to find a solution. Despite these times, it is in these dynamically more complex scenarios that the benefits of RRTs on kinodynamic planning start to play out. The RRT planner gracefully adapts to the doubling of the dimensionality of state space or the increased environmental complexity of the planning task in the last example. It can clearly be seen from the examples that the rapid exploration of space afforded by the RRTs aids the planning task significantly.

5.4 Trajectory design

Cheng et. al [5] apply RRT-based planners to even more dynamical systems than those presented in [12], in order to design motion trajectories for fast-moving automobiles through obstacle courses and re-entry motions of a reusable space launch vehicle. For the case of a simple Reeds-Shepp car (as shown in Fig. 2), the resulting RRT for a maze navigation task is shown in Fig. 6. Fig. 7 shows designed trajectories for a car with complex nonlinear dynamics having to move through an obstacle course at $108\frac{km}{h}$ and for reentry of a NASA X33 reusable spacecraft. While Cheng et. al allude to the usefulness of RRTs for these design purposes, they do not supply computation times. It is assumed that these are considerable.

6 Conclusion

LaValle & Kuffner, in [12], presented the first application of randomized techniques to kinodynamic planning in state space. Their use of RRTs to quickly explore state space coupled with a straightforward bidirectional planning mechanism yields promising initial results for systems with significant dynamics.

Since the first mention of the RRT algorithm, it has quickly gained a following. Cortes [6] applies an RRT algorithm augmented with sampling techniques to plan motions of closed kinematic chains of robotic manipulators connected by solid objects. Carpin & Pagello [4] parallelize the RRT algorithm for planning problems involving more than one robot. Bruce & Veloso [2] extend the RRT algorithm using caching to apply it in real time to a team of coordinating robots.

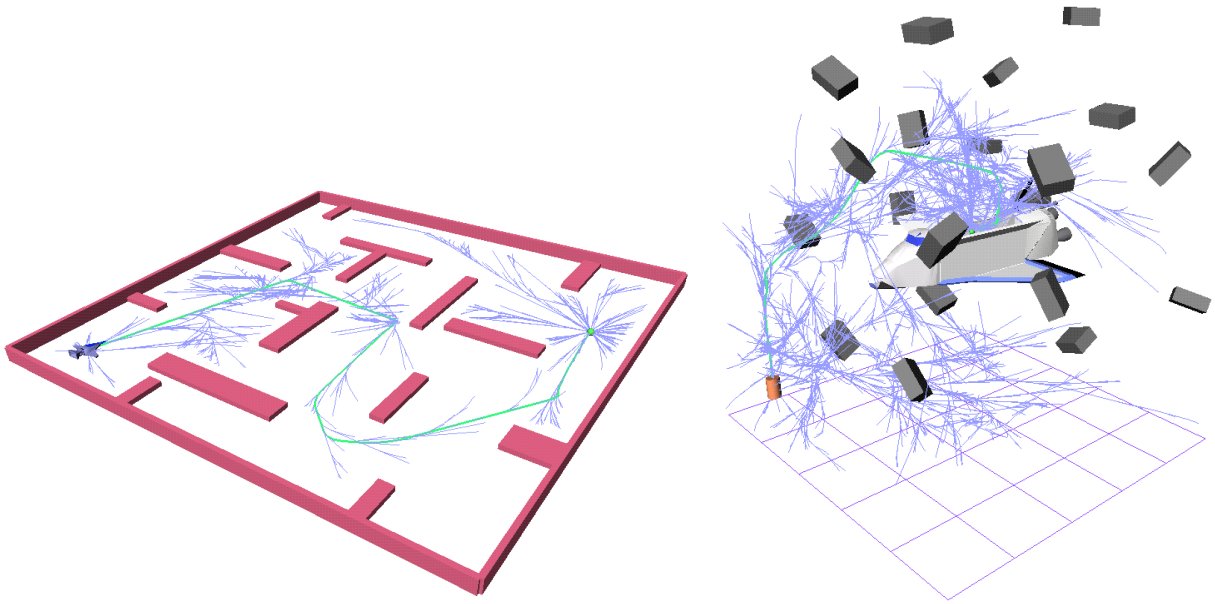

Figure 5: RRTs for the translation & rotation planning tasks in 2D (left) and 3D (right).

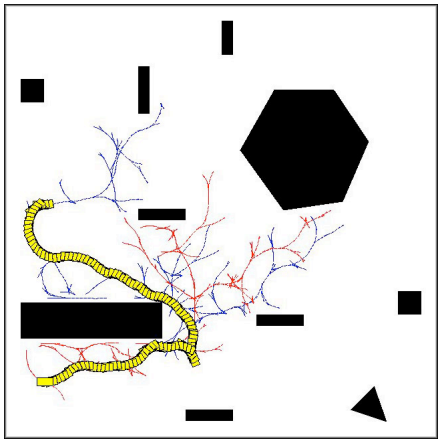

Figure 6: RRT and solution motion trajectory for a 2D Reeds-Shepp car.

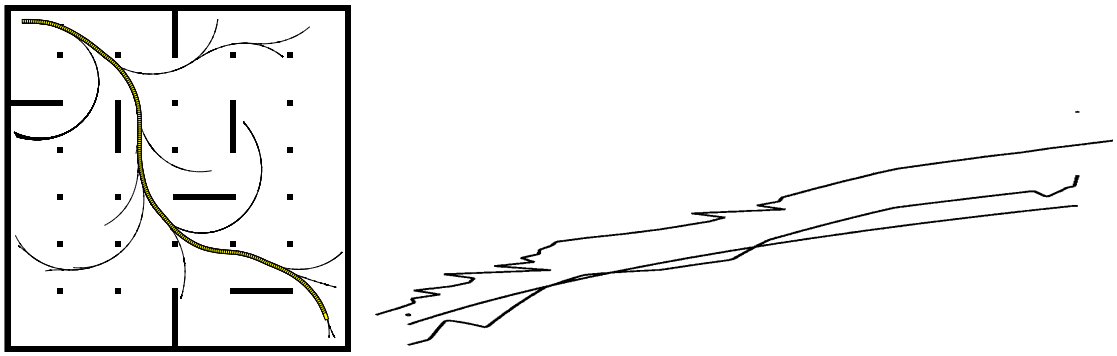

Figure 7: Designed trajectory for a fast car moving through obstacles (left). An altitude vs. speed plot of the re-entry corridor for the X-33 launch vehicle (right).

Despite recent popularity, several key challenges remain. A thorough computational analysis of RRT-based planning algorithms is still outstanding. The design of good metrics for state space is crucial for RRT planners and is likely to remain a difficult undertaking. There exists potential for adapting RRT algorithms specifically to temporally varying environments.

References

[1] J. Barraquand and J.-C. Latombe. Robot motion planning: a distributed representation approach. *Int. J. Rob. Res.*, 10(6):628–649, 1991.

[2] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. *Proceedings of IROS-2002*, October 2002.

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

[4] S. Carpin and E. Pagello. On parallel rrts for multi-robot systems. *Proc. 8th Conf. of the Italian Assoc. for Artificial Intelligence*, pages 834–841, 2002.

[5] P. Cheng, Z. Shen, and S. M. LaValle. Rrt-based trajectory design for autonomous automobiles and spacecraft. *Archives of Control Sciences*, 11(3–4):51–78, 2001.

[6] J. Cortés. *Motion Planning Algorithms for General Closed-Chain Mechanisms*. PhD thesis, Institut National Polytechnique de Toulouse, 2003.

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

[8] B. R. Donald and P. G. Xavier. Provably good approximation algorithms for optimal kinodynamic planning for cartesian robots and open chain manipulators. In *Symposium on Computational Geometry*, pages 290–300, 1990.

[9] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. Technical report, Stanford University, Stanford, CA, USA, 1994.

[10] J. J. Kuffner. Goal-directed navigation for animated characters using real-time path planning and control. In *CAPTECH '98: Proceedings of the International Workshop on Modelling and Motion Capture Techniques for Virtual Environments*, pages 171–186, London, UK, 1998. Springer-Verlag.

[11] J.-C. Latombe. *Robot Motion Planning*. International Series in Engineering and Computer Science; Robotics: Vision, Manipulation and Sensors. Kluwer Academic Publishers, Boston, MA, U.S.A., 1991.

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

[13] T. Liu, A. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *Proceedings Conference on Neural Information Processing Systems*, 12 2004.

[14] M. Mason. *Mechanics of Robotic Manipulation*. MIT Press, August 2001. Intelligent Robotics and Autonomous Agents Series, ISBN 0-262-13396-2.

[15] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings 20th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 421–427, 1979.

[16] G Sahar and J M Hollerbach. Planning of minimum-time trajectories for robot arms. *Int. J. Rob. Res.*, 5(3):90–100, 1986.