Reducing Metric Sensitivity in Randomized Trajectory Design

Peng Cheng   Steven M. LaValle

Dept. of Computer Science
Iowa State University
Ames, IA 50011 USA

Abstract

This paper addresses trajectory design for generic problems that involve: 1) complicated global constraints that include nonconvex obstacles, 2) nonlinear equations of motion that involve substantial drift due to momentum, 3) a high dimensional state space. Our approach to this challenging class of problems is to develop randomized planning algorithms on the basis of Rapidly-exploring Random Trees (RRTs). RRTs use metric-induced heuristics to conduct a greedy exploration of the state space; however, performance substantially degrades when the chosen metric does not adequately reflect the true cost-to-go. In this paper, we present an adaptive version of the RRT that is capable of refining its exploration strategy in the presence of a poor metric. Initial experiments on problems in vehicle dynamics and spacecraft navigation indicate substantial performance improvement over existing techniques.

Figure 1: A designed trajectory for a car that drives through a virtual town at 72 kph. The vehicle dynamics are modeled as a nine-dimensional nonlinear systems that accounts for tire loading, skidding, and basic suspension effects.

1 Introduction

Trajectory design in the presence of both local, differential constraints and global, state-space constraints is an important problem in robotics and many other applications. In addition to these constraints, both the high-dimensionality and the nonlinearity of dynamical systems are of interest in this paper. In robotics, computation of open-loop trajectories for basic manipulation and motor tasks for highly-complex robots, such as the Honda humanoid, is a typical problem. In the control kernel of the autonomous vehicles, a trajectory planner needs to be so fast enough operate almost in real time for a vehicle to adapt to a change in operating regimes or control authority. The trajectory design problem also exists in virtual prototyping. For example, to find a flaw in the design of a new car at an early stage, evaluation of the vehicle can be achieved by experiments that use a "virtual stunt driver", a trajectory design algorithm, and the "car", a high-quality simulator. Fig. 1 shows such an experiment to check if the car can drive safely through a town with a given speed. Similar problems exist in the prototyping of aircraft, spacecraft, hover-crafts, submarines, and a wide variety of mechanical machinery.

Finding a feasible trajectory is hard. Even the simple generalized mover's problem, which has no differential constraints, is PSPACE-hard [30]. The classic approach trajectory design in robotics research has been to decouple the general robotics problem by solving basic path planning that takes into account obstacles while ignoring differential constraints, and then find a trajectory and controller that satisfies the dynamics and tracks the path [6, 22, 32]. However, the result of a purely kinematic planner might be unexecutable by the robot due to limits on the actuator forces and torques. Approaches that avoid decoupling have been proposed recently. For problems that involve low degrees of freedom, classical dynamic programming ideas can be employed to yield numerical optimal control solutions for a broad class of problems

[3, 5, 20, 21]. This idea has been proposed in various forms in the motion planning and robotics literature [1, 7, 10, 11, 12, 13, 15, 23, 27, 28, 29, 31]. Due to the curse of dimensionality, dynamic programming methods are impractical for the generic, high-dimensional problems considered in this paper.

Attempts to fight the curse of dimensionality have led to the introduction of randomized (or Monte-Carlo) approaches into trajectory design. Algorithms based on Rapidly-exploring Random Trees (RRTs) [24, 25] have been proposed recently for trajectory design for problems that involve dynamics and complicated obstacle constraints. An RRT achieves rapid exploration by iteratively sampling a random state in the state space and extending the nearest state in the RRT to get as close as possible to the random state. Various RRT-based planners has been designed recently for autonomous vehicles motion planning [14] and for nonlinear underactuated vehicles [33]. A trajectory planner also based on randomized incremental search was proposed for time-varying systems in [18].

In spite of the successes of RRTs, one of the key shortcomings is the sensitivity of their performance with respect to a chosen metric on the state space. The metric serves as a guide to improve performance; however, for systems that involve substantial momentum, the metric can provide misleading information that dramatically increases the computation time. For some systems it may be possible design better metrics (as in the hybrid optimal cost-to-go function in [14]), but in general there is a need to develop randomized trajectory design algorithms that achieve reliable performance in spite of a bad metric. It is this demand that leads to the emergence of the method in this paper.

2 Problem Description

The problems considered in this paper can be formulated in terms of the following components:

1. **State Space:** A differentiable manifold, X, with an associated real-valued metric function, $\rho : X \times X \to [0, \infty)$, which specifies the distance between pair of points in X.
2. **Boundary Values:** $x_{init} \in X$ and $X_{goal} \subset X$.
3. **Constraint Satisfaction:** A function, $D : X \to \{true, false\}$, that determines whether global constraints are satisfied for state x.
4. **Input set:** A set U that specifies the complete set of controls that can affect the state.
5. **Equation of motion:** $\dot{x} = f(x, u)$, that characterizes the evolution of the state of the robot. An incremental simulator integrates this equation to obtain future states.

```
BUILD_RRT(x_init)
  T.init(x_init);
  for k = 1 to K do
      x_rand ← RANDOM_STATE();
      x_near ← NEAREST_NEIGHBOR(x_rand, T);
      u_best ← CONTROL(x_near, x_rand, x_new, T, success);
      if success
          T.add_vertex(x_new);
          T.add_edge(x_near, x_new, u_best);
  Return T
```

Figure 2: The basic RRT construction algorithm, in which NEAREST_NEIGHBOR chooses the nearest state to the random state and CONTROL selects the best input to extend the nearest state. Their algorithms are presented in Fig. 3.

The global state constraints are encoded in the constraint satisfaction, which can discriminate if a state satisfies the global constraints. The equation of motion expresses the nonlinear differential constraints of the system. The objective of the trajectory design problem is to find a control function, $u : [t_0, t_f] \to U$, in which t_0 is the starting time, t_f is the ending time when the robot reaches the goal state. By applying this control sequence to the robot system, the robot will move from x_{init} and transform from one collision free state to another collision free state according to the equation of the motion until it reaches X_{goal}.

3 RRTs and Metric Issues

An introduction to the RRT To understand metric sensitivity of the RRT-based planner, it is necessary to first describe the RRT, which is employed by the planner to explore the state space. An RRT is constructed as shown in Fig. 2. At first, let the initial state be the root of the tree. For each iteration, choose a random state in the state space, and select the nearest node in the tree based on the metric function, ρ. For the selected nearest state, an input is chosen to applied to generate a new state. If the new state satisfies the global constraints, and the distance between the new state and the random state is the smallest in all collision free states generated by applying every input in the input set to the selected state, this new state will be added to the search tree.

The basic RRT attempts to rapidly explore the state space. To solve a trajectory design problem, the RRT is adapted and incorporated into a planning algorithm [26], such as a goal-biased RRT (sometimes choose the goal state instead of the random state) and bidirectional RRTs (explore the state space using two RRTs).

```
NEAREST_NEIGHBOR(x_rand, T)
  d_m in ← ∞
  for all x in T
      d ← ρ(x, x_rand);
      if d < d_min
          d_min ← d;
          x_best ← x;
  return n_best
```

```
CONTROL(x_near, x_rand, x_new, T, success)
  d_min ← ρ(x_near, x_rand);
  success ← false;
  for all u in U
      x' ← Integrate(x_near, u);
      if D(x')
          d ← ρ(x', x_rand);
          if d < d_min
              d_min ← d;
              success ← true;
              u_best ← u;
  return u_best
```

Figure 3: An algorithm to choose best input and its resulting new state.

Dependency of RRT on the metric The ideal metric is the optimal cost-to-go, which is the optimal cost for the robot to move from one state to another state. The cost might be understood as the distance traveled, the energy consumed, or the time elapsed during the execution of a trajectory. Unfortunately, calculating the optimal cost-to-go is at least the same difficulty as the trajectory design problem. Both the differential constraints and the global constraints have to be considered. The effect of the differential constraints can be seen from the following example. Suppose a car-like robot is driving forward with high speed. The radius of the smallest circle in which it can turn is 100 meters. The robot is driving past the origin of y axis to the positive direction of the y axis. One state is at the 150 meters and another is at -100 meters. A Euclidean metric might cause the algorithm to prefer the -100 state; however it is worse because the car cannot drive backwards and would have to turn around to go to -100 the state; the state at 150, is closer in terms of the true cost. To understand the effect of global constraints on the metric, imagine that a robot is in a labyrinth of nonconvex obstacles. Two states might be close in terms of a Euclidean metric, but the correct metric should use the shortest path within the labyrinth. This is a well-known problem, which also occurs in potential field methods [2, 17].

The performance degradation occurs for the following reasons:

1. The RRT chooses the nearest state only depending on the metric function. When the metric function provides bad information, the tree might grow in the wrong direction, making it hard for the RRT to explore the whole state space(Fig. 9). Another problem is that if only the distance information is considered, many states, which are even destined to a result in a collision, might be chosen numerous times for expansion.

2. The best input for the nearest state is selected by relying only on the metric. Similar to the above situation, selecting the best input based on the metric information might drive the robot along a poor path. The input that yields good exploration might be discarded because the new state derived from this input has "larger" distance to the random state than that of the other states derived from the other inputs.

4 Adaptive Reduction of Metric Sensitivity

The idea in this paper is not to design a system-specific metric, but to collect information during the exploration of the state space and expand the search tree according to both the metric information and information collected. These can make the RRT less metric-sensitive and therefore more robust.

The following information is collected during the search:

1. **Exploration information:** For each RRT-node, we keep track of whether an input has already been applied and evaluated. If an input has been applied for a state, it will not be considered for the state again. If the inputs for a state are exhausted, this state will be excluded from the search space. In this way, the RRT will have more chances to explore the state space. Moreover, the planner avoids doing collision checking for the same input and state repeatedly.

2. **Path collision tendency:** Given a set, S, of all input sequences of length n time steps, the *path collision probability* of the state, x, is the collision probability of the path generated by applying a random input sequence in S to x. It can be calculated by applying all input sequences in S to x and dividing the number of collision paths by the number of all input sequences. It provides some characterization of the distribution of global state constraints for the planner. States with collision probability 1 will be prevented from expanding because all paths via them will collide. States with less collision probability will be given more priority to expand because they have better chance to evade the constraints. However, calculating the actual collision probability is impractical for large input sets and long input sequences (one might as well use dynamic programming to solve the

problem in this case). The path collision tendency, a lower bound for the actual probability (explained in the following part), is used in this paper. With more and more exploration, the path collision tendency will approaches but never exceed the actual probability and any states whose actual collision probability is less than 1 will not be prevented from expanding by overestimating its collision probability.

To collect the exploration information, a vector is kept at each state in the search tree. Each element of the vector corresponds to an input in U (it is assumed that U is approximated by a finite set). Initially, each element of the vector is set to be *unexpanded*. If one input leads to a collision or it successfully expands a state and generates a new nodes in the tree, its corresponding element in the vector is set to be *expanded*.

To calculate the path collision tendency, the following method is adopted. Initially, for each new state appended to the search tree, its collision tendency is 0. In the exploration, when the new state, x_s, is selected to expand the tree. Instead of trying all input sequence of length k, all of the possible first inputs are tested on it. For example, suppose there are M inputs in the input set and n inputs lead x_s to collision directly. In this case, the path collision tendency of x_s becomes n/M. Furthermore, the path collision tendency of its parent state, x_p, is increased by n/M^2 because we are sure that n inputs of one of its child states lead to eventual collision. Similarly for the k^{th} parent state of x_s, n collision inputs of x_s will increase its path collision tendency by $n/M^{(k+1)}$. Because the path collision tendency accumulates only when collisions happen and the RRT is not performing an exhaustive search, only part of all possible collision paths are consider; the path collision tendency will always be no larger than the actual probability. Fig. 4 shows the value of the path collision tendency collected in the exploration and provides the global constraints information in the searched state space.

The exploration information and path collision tendency help the RRT to choose a better state for expansion. When selecting the nearest neighbor state for a random state, the planner will first check if all the inputs of a state, x, are exhausted. If they are exhausted, x is just ignored; otherwise, the probability of not choosing x equals path collision tendency of x. The modified NEAREST_NEIGHBOR function is given in Fig. 5.

The exploration information is helpful for selecting the best input. When a state is selected, whether a input is an expanded input will be checked first. If it is an expanded input, it will not be considered, otherwise it will be expanded to see if the new state is collision free and is closer to the random state. If this new state is collision free and closer to random state, this input will be considered as a better input. If the new state

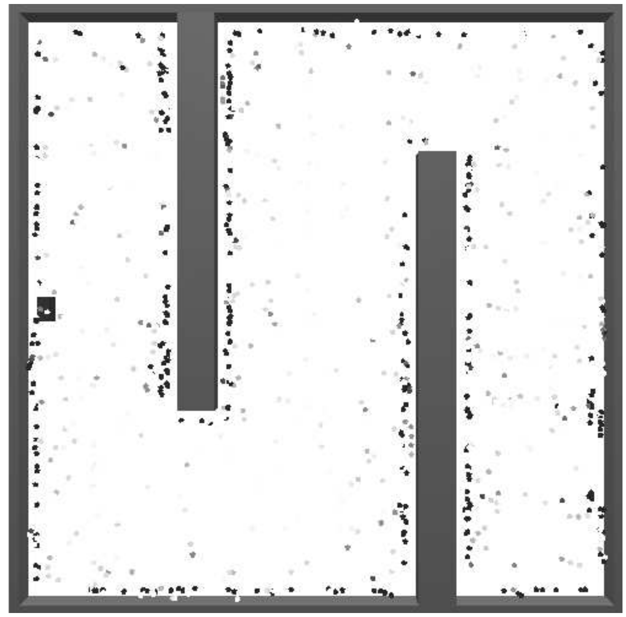

Figure 4: Path collision tendency collected in the exploration, in which the points represent the states and the shade of a point represents the value of the path collision tendency (darker indicates higher collision probability.

is in the collision region, the collected information will be collected. The modified CONTROL algorithm is in Fig. 6.

5 Improved RRT-Based Planners

The improved RRT can be incorporated into any of the RRT-based planners described in [26]. A single-tree planner expands one RRT from the initial state, and a solution is found after a new state added to the RRT that lies in X_{goal}. A goal-based planner is obtained by replacing the random state with a function that tosses a biased coin to determine what should be returned. If the coin toss yields "heads", then x_{goal} is returned; otherwise, a random state is returned. Even with a small probability of returning heads (such as 0.05), the method usually converges to the goal much faster than the basic RRT. If too much bias is introduced; however, the planner begins to behave like a randomized potential field planner that is trapped in a local minimum.

A bidirectional planner expands two trees from both the initial state and the goal state, and a solution if a pair of states, one from each tree, are within a distance threshold. The computation time is divided between two processes: 1) exploring the state space; 2) trying to grow the trees into each other. Two trees, $\mathcal{T}_a$ and $\mathcal{T}_b$ are

```
NEAREST_NEIGHBOR(x_rand, T)
    d_min ← ∞;
    for all x in T
        if inputs of x are exhausted
            r ← random number in [0, 1];
            if r > σ(x);
                d ← ρ(x, x_rand);
                if d < d_min
                    d_min ← d;
                    x_best ← x;
    return x_best;
```

Figure 5: The modified NEAREST_NEIGHBOR algorithm, in which $\sigma(x)$ represents the path collision tendency of the state x.

```
CONTROL(x_near, x_rand, x_new, T, success)
    d_min ← ∞;
    success ← false;
    for all u in U
        if u has not been expanded
            x' ← Integrate(x_near, u);
        if D(x')
            d ← ρ(x', x_rand);
            if d < d_min
                d_min ← d;
                success ← true;
                u_best ← u;
        else
            mark u as expanded
            UPDATE_TREEINFO(x_near, T)
    mark u_best as expanded
    return u_best
```

Figure 6: Modified CONTROL algorithm, in which the UPDATE_TREEINFO function updates the information during the exploration. The detailed algorithm is given in Fig. 7.

```
UPDATE_TREEINFO(x_near, T)
    p ← 1/M;
    σ(x_near) ← σ(x_near) + p;
    p ← 1/M^2;
    x_1 ← x_near;
    while x_1 is not root
        x_2 ← parent(x_1);
        σ(x_2) ← σ(x_2) + p;
        p ← p/M;
        x_1 ← x_2;
```

Figure 7: An algorithm to collect information during the exploration.

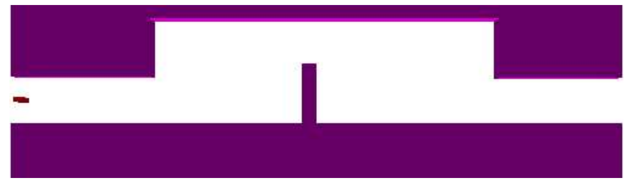

Figure 8: The task is to design a trajectory that causes a fast lane change for a car going 60 mph.

maintained at all times until they become connected and a solution is found. In each iteration, one tree is extended, and an attempt is made to connect the nearest vertex of the other tree to the new vertex.

The performance is generally better in comparison to a single-tree planner; however, detecting a solution becomes an interesting problem. It is possible that during the exploration of the dual RRTs the original planner might continue to explore the state space even there is already a solution existed between the two RRTs. The exploration fronts of two search trees passing through each other is also mentioned in [16]. To overcome the above problem, we currently test whether each new node in one tree is within a specified distance to any node in the other tree. Although costly, it generally leads to reliable performance because all alternatives are considered.

6 Experimental Results

Our implementation was built on top of the C++ Motion Strategy Library[1]. Experiments were conducted on a 1000Mhz PC running Red Hat Linux 6.2. Comparisons between planners were conducted in which for each type of planner, one version uses the original RRT, and the other uses the improved RRT proposed in this paper. Several experiments were performed for challenging trajectory design problems that include vehicle dynamics problems and spacecraft problems.

Fig. 8 shows a problem in which a car drives at 96kph and needs to complete a lane changing maneuver in a 305m stretch of road. This problem is referred to in the automotive industry as the Consumer Union Short Course. The original RRT and the improved RRT are shown in Fig. 9 and Fig. 10, respectively. The vehicle dynamics model is highly nonlinear, considers nonlinear loads on the pavement, and has five state variables. It is a simplified version of the nine-dimensional system given in the appendix.

Figure 11 gives some comparative statistics for solutions to the lane changing problem under the application of the bidirectional planner. Fifty trials were

[1] http://janowiec.cs.iastate.edu/msl/

Figure 9: RRT exploration (after 10000 iterations), using a weighted Euclidean metric. Exploration is limited because some nodes are repeatedly selected for expansion without making progress.

Figure 10: Improved RRT exploration (after 6555 iterations), using the same metric.

performed in which six versions were run: the original RRT with 2000, 4000, and 8000 iterations, and the improved RRT with 2000, 4000, and 8000 iterations. Note that the success rate improves dramatically. Furthermore, the average number of nodes generated by the improved RRT is greatly increased, indicating greater exploration. Also, less collision detection was performed by the improved RRT. Although computation times are comparable, note that most of the original RRT trials result in failure.

Fig. 1 shows an experiment of driving a car at 72kph through a virtual town which is 300m by 300m. The nine-dimensional system is described in the appendix. The model considers the rolling angle and the rolling speed of the car. The pressure on the individual tires varies because of rolling effects. If the pressure on one tire is less than 0, the car is in a dangerous state. This makes it very difficult to control.

Based on 50 trials, in which for each trial, there are 60000 RRT iterations, the improved goal-biased RRT planner finds the solution 38 times with an average of 989.65 seconds and 25712.1 nodes. The original goal-biased RRT planner performed much worse, finding a solution only 10 times out of 50 trials (note that either success rate can be improved by increasing the number of iterations).

The final experiment involves moving an underactuated spacecraft out of a cage by firing thrusters (Fig. 12). The model is a twelve-dimensional nonlinear system described in the appendix. The spacecraft can translate and rotate in 3D space. Three thrusters provide the driving forces and torques, which applied in directions that avoid the mass center. For 50 trials and 40000 RRT iterations in each run, the improved bidirectional RRT planner solves the problem 41 times with an average of 737.32 seconds and 17129 nodes. The original bidirectional RRT planner only solved the problem 3 times out of 50 trials, even though 100000 RRT iterations were run in each trial.

	RRT Planner			Improved RRT Planner		
It	2000	4000	8000	2000	4000	8000
S	1/50	0/50	4/50	23/50	37/50	49/50
N	336	-	636.5	1359	2542	3283
CD	56.9	-	27.3	2.48	4.49	5.75
T	10.61	-	52.15	11.9	28.8	44.7

Figure 11: Comparison of RRT-based planner and Robust RRT-based planner, in which "It" means how many iterations the search tree extends, "S" means the number of successes out of 50, "N" means the average nodes in the search tree, "CD" means the average collision checking times (in thousands), "T" means the average time needed to find the solution.

7 Discussion

We have presented an improved RRT-based planning method for problems that involve obstacles, high dimensionality, and nonlinear systems with drift. In particular, sensitivity to poor metrics is reduced by applying information gathered during the search. From the perspective of classical AI search, if the random state in the RRT algorithm is replaced by the goal state, then the RRT reduces to a greedy search that does not consider repeated states, and is based on the single heuristic metric function. RRTs are able to overcome typical local minima problems; however, the efficiency of the search still depend on the quality of the heuristic information. The improved RRT uses the exploration information to exclude the repeated states from the state space and enables the planner to have more chances to search the unexplored state space. The path collision tendency collected during the exploration provides the global state constraints information distributed in each state in the search tree. Combining the collected information and the metric function yields an improved RRT-based planner that can find the solutions more efficiently.

To handle the problem of search frontiers of the bidirectional planner passing through each other, a naive method was used in this paper by checking every new

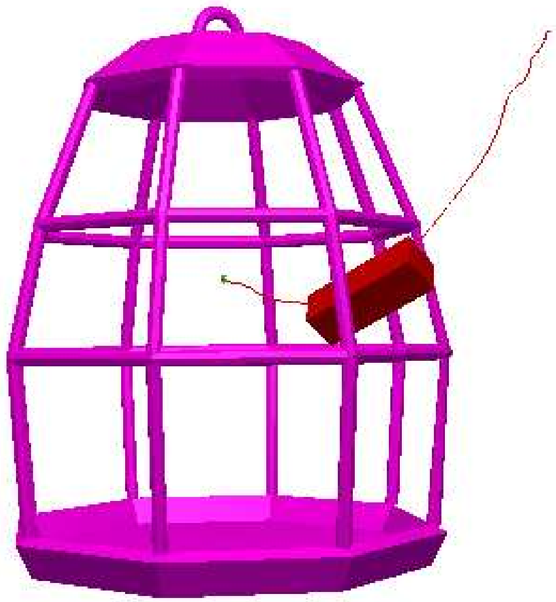

Figure 12: An experiment of thrusting a spacecraft with 3 thrusters to move out of a cage

pair of states. It is very time consuming especially when the problem is difficult and the number of the states in the tree is large. Some research in the bidirectional search, such as BS* [19] and wave-shaping algorithms [8, 9] might help to alleviate this problem.

8 Acknowledgments

We are very grateful to James Bernard for his general advice, and for assisting us in the development of the vehicle dynamics models used in the experiments. We also thank James Kuffner and Libo Yang. This work was funded in part by NSF CAREER Award IRI-9875304 (LaValle).

References

[1] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. In *IEEE Int. Conf. Robot. & Autom.*, pages 2328–2335, 1991.

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

[3] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[4] J. Bernard, J. Shannan, and M. Vanderploeg. Vehicle rollover on smooth surfaces. In *Proc. SAE Passenger Car Meeting and Exposition*, Dearborn, Michigan, 1989.

[5] D. P. Bertsekas. Convergence in discretization procedures in dynamic programming. *IEEE Trans. Autom. Control*, 20(3):415–419, June 1975.

[6] J. Bobrow, S. Dubowsky, and J. Gibson. Time-optimal control of robotic manipulators. *Int. Journal of Robotics Research*, 4(3), 1985.

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

[8] D. D. Champeaux. Bidirectional heuristic search again. *Journal of the ACM*, 30(1):22–32, January 1983.

[9] D. D. Champeaux and L. Sint. An improved bidirectional heuristic search algorithm. *Journal of the ACM*, 24(2):177–191, April 1977.

[10] M. Cherif. Kinodynamic motion planning for all-terrain wheeled vehicles. In *IEEE Int. Conf. Robot. & Autom.*, 1999.

[11] C. Connolly, R. Grupen, and K. Souccar. A Hamiltonian framework for kinodynamic planning. In *Proc. of the IEEE International Conf. on Robotics and Automation (ICRA'95)*, Nagoya, Japan, 1995.

[12] B. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning for cartesian robots and open chain manipulators. *Algorithmica*, 14(6):480–530, 1995.

[13] B. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds. *Algorithmica*, 14(6):443–479, 1995.

[14] E. Frazzoli, M. A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicles motion planning. Technical Report LIDS-P-2468, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1999.

[15] G. Heinzinger, P. Jacobs, J. Canny, and B. Paden. Time-optimal trajectories for a robotic manipulator: A provably good approximation algorithm. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 150–155, Cincinnati, OH, 1990.

[16] H. Kaindl and G. Kainz. Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research*, pages 283–317, December 1997.

[17] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.*, 5(1):90–98, 1986.

[18] R. Kindel, D. Hsu, J.-C. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. In *IEEE Int. Conf. Robot. & Autom.*, 2000.

[19] J. B.H. Kwa. BS*: An admissible bidirectional staged heuristic search algorithm. *Artificial Intelligence*, 38:95–109, 1989.

[20] R. E. Larson. A survey of dynamic programming computational procedures. *IEEE Trans. Autom. Control*, 12(6):767–774, December 1967.

[21] R. E. Larson and J. L. Casti. *Principles of Dynamic Programming, Part II.* Dekker, New York, NY, 1982.

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

[23] S. M. LaValle. Numerical computation of optimal navigation functions on a simplicial complex. In P. Agarwal, L. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective.* A K Peters, Wellesley, MA, 1998.

[24] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University. <http://janowiec.cs.iastate.edu/papers/rrt.ps>, Oct. 1998.

[25] S. M. LaValle and J. Kuffner Jr. Randomized kinodynamic planning. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1999.

[26] S. M. LaValle and J. Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *2000 Workshop on the Algorithmic Foundations of Robotics*, 2000.

[27] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *Int. J. Robot. Res.*, 15(6):533–556, 1996.

[28] C. O'Dunlaing. Motion planning with inertial constraints. *Algorithmica*, 2(4):431–475, 1987.

[29] 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.

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

[31] G. Sahar and J. M. Hollerbach. Planning of minimum time trajectories for robot arms. *Int. Journal of Robotics Research*, 5(3):90–100, 1986.

[32] Z. Shiller and S. Dubowsky. On computing time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Trans. on Robotics and Automation*, 7(7), December 1991.

[33] G. J. Toussaint, T. Başar, and F. Bullo. Motion planning for nonlinear underactuated vehicles using hinfinity techniques. Coordinated Science Lab, University of Illinois, September 2000.

Appendix

The nine-dimensional car model The following model is adapted from [4]. The state vector is $(x, y, r, \psi, \phi, q, \nu, s, \beta)$. The 3D coordinate frame is designed with the x coordinate increasing from left to right, the y coordinate increasing from top to bottom, and the z coordinate inward (to form a right-handed coordinate system). Let β be the steering angle. Let a and b be the distance from the front and rear axles to the car center, respectively. Let ψ be the yaw angle of the car. Let s be the forward speed of the car, and let ν be the sideways speed (arising from slipping). Let r be the angular velocity. Let ϕ be the roll (which describes the sideways tilting of the car). Let q be the roll angle rate. Let α_f and α_r be the slipping angle of the front and rear wheels, respectively. These are expressed as

$$\alpha_f = \frac{\nu + ar}{s} - \beta$$

and

$$\alpha_r = \frac{\nu - br}{s}.$$

Let $C_{\alpha f}$ and $C_{\alpha r}$ be the cornering stiffness between the forces along the y axis, F_{yf} and F_{yr}, respectively, on the front and rear wheels. Under some conditions, it is possible for the car to slip sideways. If $N_f \mu/2 > C_{\alpha f} \tan(|\alpha_f|)$, the calculated friction force is less than the maximum possible friction, then $F_{yf} = -C_{\alpha f}\alpha_f$; otherwise, $F_{yf} = \mu N_f Sgn(\alpha_f)(1 - x_f/2)$, in which Sgn denotes the sign function, μ is a constant, and $x_f = N_f\mu/2C_{\alpha f}\tan(|\alpha_f|)$. Similarly, if $N_r\mu/2 > C_{\alpha r}\tan(|\alpha_r|)$, then $F_{yr} = -C_{\alpha r}\alpha_r$; otherwise, $F_{yr} = \mu N_r Sgn(\alpha_r)(1 - x_r/2)$, in which $x_r = N_f\mu/2C_{\alpha r}\tan(|\alpha_r|)$. Let M be the car mass, and let I be the yaw moment of inertia. Let H_2 be the distance from the joint connecting the chassis with the car frame (the chassis and frame are flexibly attached to model a simple suspension system). The constants K, c, and other details are described an in [4].

Let $h = (-(K - MgH_2)\phi - cq - (F_{yf} + F_{yr})H_2)/I$. The following represent the nine equations of motion: $\dot{x} = s\cos\psi - \nu\sin\psi$, $\dot{y} = s\sin\psi + \nu\cos\psi$, $\dot{r} = (F_{yf}a - F_{yr}b)/I$, $\dot{\psi} = r$, $\dot{\phi} = q$, $\dot{q} = h$, $\dot{\nu} = (F_{yf} + F_{yr})/M - sr - H_2 h$, $\dot{s} = u_1$, $\dot{\beta} = u_2$.

The inputs are u_1, which is linear acceleration, and u_2, which is the change in the steering angle.

The twelve-dimensional spacecraft model The state is $(x, y, z, \psi, \phi, \beta, s_x, s_y, s_z, s_\psi, s_\phi, s_\beta)$, in which x, y, z are the position, ψ, ϕ, β are the Euler orientation angle, s_x, s_y, s_z are the speeds of translation in x, y, z axis directions and s_ψ, s_ϕ, s_β are the speeds of the Euler angles.

Let M is the mass of the spacecraft, I is its inertial matrix. Three thrusters are respectively installed on the x, y, z axis direction. To provide both the driving forces and torques, these thrusters do not apply the force through the mass center. Let the forces from the thrusters are F_x, F_y, F_z, the vertical distance from the mass center to the force is L_x, L_y, L_z and the orientation transformation matrix is R, which is the function of ψ, ϕ, β. The twelve motion equations are:

$$\dot{x} = s_x,\ \dot{y} = s_y,$$
$$\dot{z} = s_z,\ \dot{\psi} = s_\psi,$$
$$\dot{\phi} = s_\phi,\ \dot{\beta} = s_(\beta),$$
$$\dot{s}_x = F_x/M,\ \dot{s}_y = F_y/M,$$
$$\dot{s}_z = F_z/M,\ \dot{s}_\psi = [1,\ 0,\ 0]\ R\ A_{\psi\phi\beta},$$
$$\dot{s}_\phi = [0,\ 1,\ 0]\ R\ A_{\psi\phi\beta},$$
$$\dot{s}_\beta = [0,\ 0,\ 1]\ R\ A_{\psi\phi\beta},$$

in which $A_{\psi\phi\beta} = I^{-1}\ [F_x L_x,\ F_y L_y,\ F_z L_z]^T$.