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 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.

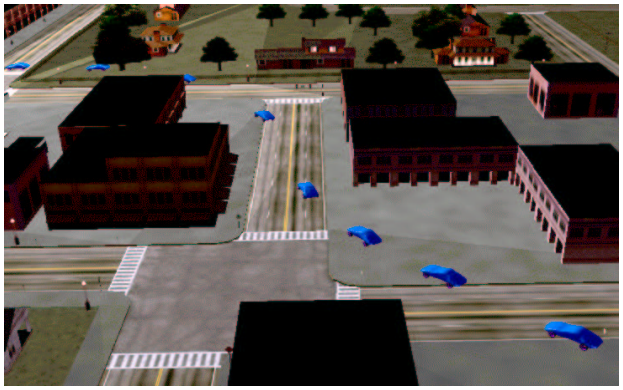

Figure 1: A designed trajectory for a car that drives through a virtual town at 72 kph. The vehicle dynamics are modeled as a nine-dimensional nonlinear systems that accounts for tire loading, skidding, and basic suspension effects.

1 Introduction

Trajectory design for the high-dimensional and nonlinear dynamical systems with differential constraints is the problem considered in this paper. The typical problems are trajectory design for highly-complex robots and autonomous vehicles. It is also applied in the virtual prototyping. For example, evaluation of the design of a vehicle can be achieved by experiments that use a "virtual stunt driver", a trajectory design algorithm, and the "car", a high-quality simulator. Figure 1 shows such an experiment to check if the car can drive safely through a town with a given speed.

Finding a feasible trajectory is hard. Even the simple generalized mover's problem, which has no differential constraints, is PSPACE-hard [27]. The classic approach in robotics research has been to decouple the 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, 19, 29]. 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, 18]. This idea has been proposed in various forms in the motion planning and robotics literature [1, 7, 9, 10, 11, 13, 20, 24, 25, 26, 28]. 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 approaches into trajectory design. Algorithms based on RRTs [21, 22] 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-

wards the random state. Various RRT-based planners has been designed recently for autonomous vehicles motion planning [12] and for nonlinear underactuated vehicles [30]. A trajectory planner also based on randomized incremental search was proposed for time-varying systems in [16].

In spite of the successes of RRTs, one of the key shortcomings is the sensitivity of their performance with respect to a chosen metric. The metric serves as a guide to improve performance; however, for systems that involve substantial momentum, the metric might provide misleading information that dramatically increases the computation time. For some systems, it may be possible to design better metrics (as in the hybrid optimal cost-to-go function in [12]), but in general there is a need to develop randomized trajectory design algorithms that achieve reliable performance in spite of a poor metric. It is this demand that leads to the emergence of the method in this paper.

2 Problem Description

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

1. **State Space:** An n dimensional differentiable manifold, X, with a metric function, $\rho : X \times X \rightarrow [0, \infty)$.
2. **Boundary Values:** $x_{init} \in X$ and $X_{goal} \subset X$.
3. **Constraint Satisfaction:** A function, $D : X \rightarrow \{true, false\}$, that determines whether global constraints are satisfied for state x.
4. **Finite 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.

The objective of the trajectory design problem is to find a piece-wise control function, $u : [t_0, t_f] \rightarrow U$, in which t_0 is the starting time and t_f is the ending time when the robot reaches the goal state. By applying this control function to the robot, 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 $x_{goal} \in X_{goal}$.

3 RRTs and Metric Issues

An introduction to the RRT To understand metric sensitivity of the RRT-based planner, we first describe the RRT. An RRT is constructed as shown in Fig. 2. At first, x_{init} is the root of $\mathcal{T}$. For each iteration, a $x_{rand} \in X$ is chosen, and $x_{near} \in \mathcal{T}$ is selected based on ρ. For x_{near}, u_{best} is chosen to generate x_{new}. If x_{new} satisfies the global constraints, and $\rho(x_{new}, x_{rand})$ is the smallest in all of collision-free states, which are generated by applying every input in U to x_{near}, x_{new} will be added to $\mathcal{T}$.

```
BUILD_RRT(x_init)
 1  T.init(x_init);
 2  for k = 1 to K do
 3       x_rand ← RANDOM_STATE();
 4       x_near ← NEAREST_NEIGHBOR(x_rand, T);
 5       u_best, x_new, success ← CONTROL(x_near, x_rand, T);
 6       if success
 7            T.add_vertex(x_new);
 8            T.add_edge(x_near, x_new, u_best);
 9  Return T
```

Figure 2: The basic RRT construction algorithm, in which NEAREST_NEIGHBOR chooses x_{near} to x_{rand} and CONTROL selects u_{best} to extend x_{near}. The CONTROL algorithm is presented in Fig. 3.

```
CONTROL(x_near, x_rand, T)
 1   d_min ← ρ(x_near, x_rand);
 2   success ← false;
 3   for all u in U
 4        x' ← Integrate(x_near, u);
 5        if D(x')
 6             d ← ρ(x', x_rand);
 7             if d < d_min
 8                  d_min ← d;
 9                  success ← true;
10                  u_best ← u;
11   return u_best, x_new, success
```

Figure 3: The algorithm to choose u_{best}.

To solve a trajectory design problem, the RRT is adapted and incorporated into a planning algorithm [23], 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).

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. Calculating the optimal cost-to-go is at least the same difficulty as the trajectory design problem. Both differential constraints and global constraints have to be considered. The effect of 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 planner 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 problem also occurs in potential field methods [2, 15].

The performance degradation occurs for the following reasons:

1. The RRT chooses x_{near} only depending on ρ. When ρ provides poor information, it is difficult for the RRT to explore the whole state space (Fig. 9). Another problem is that if only ρ is considered, many states might be chosen numerous times, even though they are destined to a result in a collision.

2. The u_{best} for x_{near} is selected by relying only on ρ. It might drive the robot along a poor path. The input that yields good exploration might be discarded because x_{new} derived from this input has "larger" distance to x_{rand} than that of the other states derived from the other inputs.

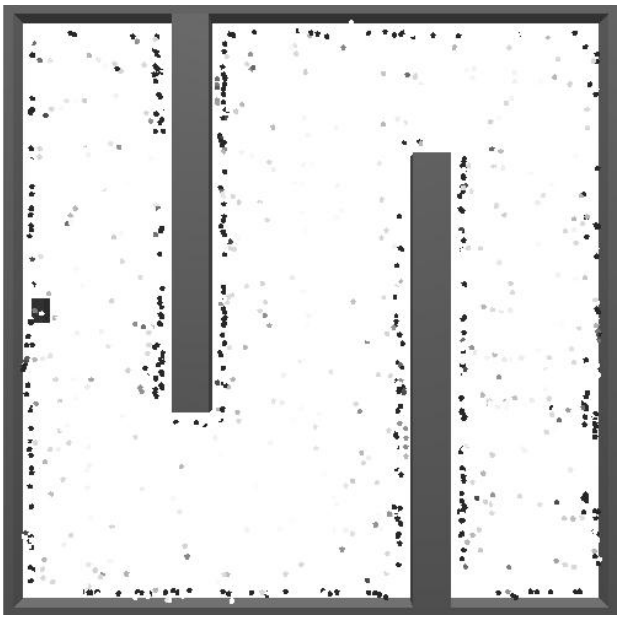

Figure 4: CVF collected in the exploration, in which the points represent the states and the shade of a point represents the value of the constraint violation frequency (darker indicates higher CVF.

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 and expand $\mathcal{T}$ according to both ρ and information collected. It can make the RRT less metric-sensitive and therefore more robust.

The following information is collected during the search:

Exploration information: For each RRT-node, we record 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. Moreover, the planner avoids doing collision checking for the same input and state repeatedly.

Constraint violation frequency (CVF): Given a set, S, of all input sequences of length n time steps, the *constraint violation probability* (CVP) 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 of input sequences in S to x and dividing the number of collision paths by the number of all input sequences. It provides global constraint information such that giving states with smaller CVP more priority to expand is more likely to evade obstacles. However, calculating CVP 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 CVF information is used and collected such that it will approaches but never exceed CVP. Any states with CVP less than 1 always have chance to expand.

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. Initially, each element of the vector is set to be *unexpanded*. If one input leads to a collision or it successfully generates a new node in $\mathcal{T}$, its corresponding element is set to be *expanded*.

To calculate the CVF, the following method is adopted. Initially, for each new state, x_s, appended to $\mathcal{T}$, its CVF is 0. When x_s is chosen as x_{near}. All of inputs in U are applied on it. Suppose there are m_u inputs in U and n inputs lead x_s to collision. The CVF of x_s becomes $\frac{n}{m_u}$. Furthermore, the CVF of its parent state, x_p, is increased by $\frac{n}{M^2}$ because n inputs of one of its m_u child states lead to collision. Similarly, for the k^{th} parent state of x_s, n collision inputs of x_s will increase its CVF by $\frac{n}{M^{(k+1)}}$. Because the CVF accumulates only when the constraint violation happens and the RRT is not performing an exhaustive search, it will always be no larger than CVP. Fig. 4 shows the value of the CVF, which provides global constraint information in the explored state space.

```
NEAREST_NEIGHBOR(x_rand, T)
 1   d_min ← ∞;
 2   for all x in T
 3       if inputs of x are not exhausted
 4           r ← random number in [0,1];
 5           if r > σ(x);
 6               d ← ρ(x, x_rand);
 7               if d < d_min
 8                   d_min ← d;
 9                   x_best ← x;
10   return x_best;
```

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

The exploration information and CVF help the RRT to choose a better x_{near}. When selecting the x_{near}, the planner will first check if all of the inputs of a state, x, are *expanded*. If they are *expanded*, x is ignored; otherwise, the probability of not choosing x equals to the CVF of x. The modified NEAREST_NEIGHBOR function is given in Fig. 5.

The exploration information is helpful for selecting u_{best}. When x_{near} is selected, whether a input is *expanded* will be checked first. If it is *expanded*, it will not be considered, otherwise it will be applied to check if the new state, x'_{new}, is collision free. The collision free x'_{new} with the lest distance to x_{rand} will be added to $\mathcal{T}$. If x'_{new} is in the collision region, constraint information will be collected. The modified CONTROL algorithm is in Fig. 6.

```
CONTROL(x_near, x_rand, x_new, T, success)
 1   d_min ← ∞;
 2   success ← false;
 3   for all u in U
 4       if u has not been expanded
 5           x' ← Integrate(x_near, u);
 6       if D(x')
 7           d ← ρ(x', x_rand);
 8           if d < d_min
 9               d_min ← d;
10               success ← true;
11               u_best ← u;
12       else
13           mark u as expanded
14           UPDATE_TREEINFO(x_near, T)
15   mark u_best as expanded
16   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.

5 Improved RRT-Based Planners

The improved RRT can be incorporated into any of the RRT-based planners described in [23]. A bidirectional planner expands two trees from both x_{init} and x_{goal}. Its performance is generally better in comparison to a single-tree planner. The original RRT only checks if two x_{new} from both trees are closed enough for a solution. Because the metric problem, it is possible that the original planner might continue to explore the state space even there exists a solution. The exploration frontiers of two search trees passing through each other was also mentioned in [14]. 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.

```
UPDATE_TREEINFO(x_near, T)
1   p ← 1/M;
2   σ(x_near) ← σ(x_near) + p;
3   p ← 1/M^2;
4   x_1 ← x_near;
5   while x_1 is not root
6       x_2 ← parent(x_1);
7       σ(x_2) ← σ(x_2) + p;
8       p ← p/M;
9       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.

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.

6 Experimental Results

Our implementation was built on top of the C++ Motion Strategy Library. Experiments were conducted on a 1000Mhz PC running Red Hat Linux 6.2. In lane changing experiments, comparisons between the original RRT and the improved RRT are done. Several challenging trajectory design experiments that include vehicle dynamics problems and spacecraft problems were used to test the performance of the new method.

Figure 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 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.

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

	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.

Figure 1 shows the virtual driving problem. The nine-dimensional system is described in the appendix. The model considers rolling effects of the car such that the pressure on the individual tires varies. 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 system described in the appendix. The spacecraft can translate and rotate in 3D space. Three thrusters provide the driving forces and torques. 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.

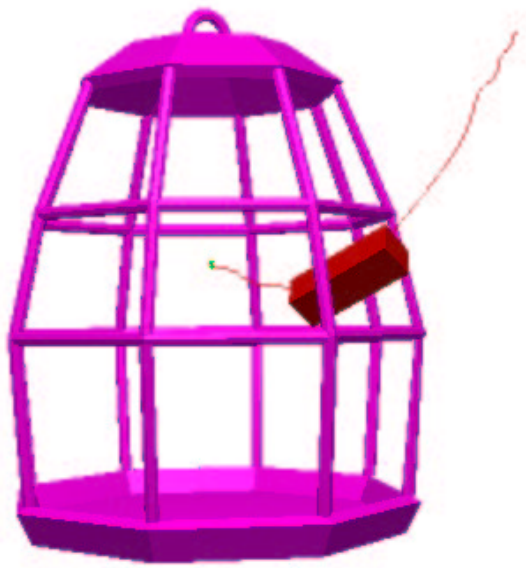

Figure 12: An experiment of thrusting a spacecraft with 3 thrusters to move out of a cage

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 x_{goal}, the RRT reduces to a greedy search without considering repeated states. 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 and enables the planner to be more likely to search the unexplored state space. The constraint violation frequency collected during the exploration provides global state constraint information distributed in the search tree. Combining the collected information and the metric function yields a more reliable improved RRT-based planner.

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 pair of states. It is very time consuming. Some research in the bidirectional search, such as BS* [17] and wave-shaping algorithms [8] 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 *SAE Technical Paper Series*, number 891991, 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. J. Robot. Res.*, 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] M. Cherif. Kinodynamic motion planning for all-terrain wheeled vehicles. In *IEEE Int. Conf. Robot. & Autom.*, 1999.

[10] 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.

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

[12] 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.

[13] G. Heinzinger, P. Jacobs, J. Canny, and B. Paden. Time-optimal trajectories for a robotic manipulator: A provably good approximation algorithm. In *IEEE Int. Conf. Robot. & Autom.*, pages 150–155, Cincinnati, OH, 1990.

[14] H. Kaindl and G. Kainz. Bidirectional heuristic search reconsidered. *JAIR*, pages 283–317, December 1997.

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

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

[17] J. B.H. Kwa. BS*: An admissible bidirectional staged heuristic search algorithm. *Artif. Intell.*, 38:95–109, 1989.

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

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

[20] 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.

[21] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University., Oct. 1998.

[22] S. M. LaValle and J. Kuffner Jr. Randomized kinodynamic planning. In *IEEE Int. Conf. Robot. & Autom.*, 1999.

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

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

[25] C. ÓDúnlaing. Motion planning with inertial constraints. *ALGO*, 2(4):431–475, 1987.

[26] 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.

[27] 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.

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

[29] Z. Shiller and S. Dubowsky. On computing time-optimal motions of robotic manipulators in the presence of obstacles. *ICRA*, 7(7), Dec 1991.

[30] 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]. Let β be the steering angle, a and b be the distance from the front and rear axles to the car center, respectively, ψ be the yaw angle of the car, s be the forward speed of the car, ν be the sideways speed (arising from slipping), r be the angular velocity, ϕ be the roll (which describes the sideways tilting of the car), q be the roll angle rate, $C_{\alpha f}$ 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, M be the car mass, I be the yaw moment of inertia, 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), K and c are constant, and α_f and α_r be the slipping angle of the front and rear wheels, respectively, the $\alpha_f = \frac{\nu+ar}{s} - \beta$ and $\alpha_r = \frac{\nu-br}{s}$. Considering slipping sideways of the car, 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 $x_r = N_f\mu/2C_{\alpha r}\tan(|\alpha_r|)$.

The state vector is $(x, y, r, \psi, \phi, q, \nu, s, \beta)$.

Let $h = (-(K - MgH_2)\phi - cq - (F_{yf} + F_{yr})H_2)/I$. The following represent the nine equations of motion: $\dot{x} = s\cos\psi - \nu\sin\psi$, $\dot{y} = s\sin\psi + \nu\cos\psi$, $\dot{r} = (F_{yf}a - F_{yr}b)/I$, $\dot{\psi} = r$, $\dot{\phi} = q$, $\dot{q} = h$, $\dot{\nu} = (F_{yf} + F_{yr})/M - sr - H_2h$, $\dot{s} = u_1$, $\dot{\beta} = u_2$.

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: $\dot{x} = s_x$, $\dot{y} = s_y$, $\dot{z} = s_z$, $\dot{\psi} = s_\psi$, $\dot{\phi} = s_\phi$, $\dot{\beta} = s_\beta$, $\dot{s}_x = [1,\ 0,\ 0]\ A_{xyz}$, $\dot{s}_y = [0,\ 1,\ 0]\ A_{xyz}$, $\dot{s}_z = [0,\ 0,\ 1]\ A_{xyz}$, $\dot{s}_\psi = [1,\ 0,\ 0]\ A_{\psi\phi\beta}$, $\dot{s}_\phi = [0,\ 1,\ 0]\ A_{\psi\phi\beta}$, $\dot{s}_\beta = [0,\ 0,\ 1]\ A_{\psi\phi\beta}$, in which $A_{xyz} = R\ [F_x,\ F_y,\ F_z]^T/M$ and $A_{\psi\phi\beta} = R\ I^{-1}\ [F_zL_z,\ F_xL_x,\ F_yL_y]^T$.