Randomized searches and nonlinear programming in trajectory planning

Timur Karatas Francesco Bullo
General Engineering Department and Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W Main St, Urbana, IL 61801, USA
{tkaratas,bullo}@uiuc.edu, http://motion.csl.uiuc.edu

Abstract

This paper presents a novel trajectory planning algorithm for nonlinear dynamical systems evolving in environments with complex obstacles. The incremental search algorithm entails (i) a global exploration strategy based on randomization and on a rapidly-exploring heuristic, and (ii) a local planner based on collocation and nonlinear programming. To numerically validate the design, we consider a six degree of freedom vehicle model subject to saturation limits on the control inputs and obstacles on the state variables. Experimental results indicate that the proposed scheme outperforms implementations based solely on nonlinear programming or on randomization.

1 Introduction

This paper investigates trajectory design algorithms for high dimensional and highly nonlinear nonlinear systems evolving in complex environments. The objective is to solve trajectory generation and optimization problems for high-fidelity models of air, land and underwater vehicles, as well as robotic manipulators and locomotion systems. Of particular interest are autonomous underwater vehicles (AUVs) and unmanned air vehicles (UAVs) for use in tasks such as search and rescue, surveillance, exploration, and area coverage.

Various numerical techniques deal with trajectory and path planning problems. In numerical optimal control, the optimal open-loop inputs and the resulting trajectories are often obtained through nonlinear programming. Because the optimization problem is infinite dimensional, various forms of transcription are used to cast the variational problem into a nonlinear program; see [2, 3, 16] for various recent surveys. Within the context of robotic path planning, the most successful solution are randomized methods, e.g., see the recent overview article [8]. Specific examples of algorithms include randomized potential field [1], probabilistic roadmaps [5], and rapidly-exploring random trees [10]. Connections of the latter scheme with control theoretical concepts are explored in [4, 15].

In this paper we consider a second order model of a vehicle moving in three dimensional Euclidean space. We assume three control torques and one control force are available. All variables are subject to various bounds including two obstacles in configuration space. The system is a simplified version of a vehicle models, but it describes aircraft motion in a more accurate fashion than a kinematic or driftless model.

We start by comparing the effectiveness of incremental search and nonlinear programming methods. Even though our implementation is suboptimal in various respects, numerical experiments confirm the conjecture that incremental search methods are well suited for environment with obstacles, whereas collocation and nonlinear programming methods perform very nicely on systems with smooth nonlinear dynamics. A second outcome of this phase is a novel way of discretizing the kinematic equations of motion for a rigid body.

Next, we develop a hybrid approach building on the relative strengths of each method. Our algorithms combines a randomized incremental search with a local planner based on collocation and nonlinear programming. The local planner is designed as follows. The number of collocation points is reduced to a minimum, and the nonlinear programming solver is given a maximum number of iterations to ensure that the total running time remain upper bounded. The overall scheme is evaluated in two settings: with and without geometric obstacles in configuration space. In both cases, we present numerical experiments in which the hybrid algorithm outperforms both previous approaches.

The paper is organized as follows. Section 2 describes models of vehicles and of the environment. Section 3 presents numerical methods for optimal control based on collocation and nonlinear programming. Section 4 presents randomized incremental search algorithms for path planning. Section 5 presents our novel algorithm, simulation results and comparisons.

2 Models of vehicle and environment

2.1 Unit quaternions to represent rotations

Quaternions generalize complex numbers and are used to represent rotation matrices in SO(3). A quaternion

is [13] a vector quantity of the form

$$\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}, \quad q_i \in \mathbb{R},\ i = 0, \ldots, 3$$

where q_0 is the scalar component and $\mathbf{q}_v$ is the vector component of $\mathbf{q}$. A convenient notation for the quaternion $\mathbf{q}$, is $\mathbf{q} = (q_0, \mathbf{q}_v)$. The unit quaternions are the subgroup of all quaternions with unit norm $\|\mathbf{q}\|^2 = 1$. Given a rotation matrix in SO(3), we define the associated unit quaternion as

$$\mathbf{q} = (\cos(\theta/2),\ w\sin(\theta/2)),$$

where $w \in \mathbb{R}^3$ represents the unit axis of rotation and $\theta \in \mathbb{R}$ represents the angle of rotation. Vice-versa, given a unit quaternion, $\mathbf{q} = (q_0, \mathbf{q}_v)$, the associated rotation matrix $R \in \mathrm{SO}(3)$ is

$$R(\mathbf{q}) = I_3 + 2q_0\widehat{\mathbf{q}}_v + 2\widehat{\mathbf{q}}_v^2,$$

where the $\widehat{\cdot}$ operator satisfies: $\widehat{x}y = x \times y$, for $x, y \in \mathbb{R}^3$.

If the rotation matrix R evolves in time with constant body-fixed angular velocity $\omega = [\omega_x\ \ \omega_y\ \ \omega_z]^T$, the unit quaternion representation of R propagates forward in time via the closed-form equation [12]

$$\mathbf{q}(t + \Delta t) = \left(\cos\left(\|\omega\|\frac{\Delta t}{2}\right) I_4 + \frac{1}{\|\omega\|}\sin\left(\|\omega\|\frac{\Delta t}{2}\right)\mathrm{F}_1(\omega)\right)\mathbf{q}(t), \quad (1)$$

where I_4 is the identity matrix in $\mathbb{R}^4$, $\|\omega\|^2 = \omega^T\omega$, and

$$\mathrm{F}_1(\omega) \triangleq \begin{bmatrix} 0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & -\omega_3 & \omega_2 \\ \omega_2 & \omega_3 & 0 & -\omega_1 \\ \omega_3 & -\omega_2 & \omega_1 & 0 \end{bmatrix}.$$

2.2 Ordinary differential equation model

We consider a simplified model of a typical aircraft flying in three dimensional Euclidean space. The vehicle configuration is described by location of the center of mass and rotation of the body-fixed frame, and the vehicle velocity is composed of angular and translational velocity. More specifically, the system has 11 states. The first 4 states are the quaternion representing the rotation of the body written in inertial coordinate frame, the next three states are the angular velocity written in the body coordinate frame, the next three states are the location of center of mass of the rigid body written in the inertial coordinate frame, and the last state represent the velocity in the direction of translation (fixed in the body coordinate frame). In other words, we have $x = [x_1^T, x_2^T, x_3^T, x_4^T]^T$ and

$$x_1 = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T \qquad x_2 = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 \end{bmatrix}^T$$
$$x_3 = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T \qquad x_4 = v_x$$

According to these definitions, the system is written as

$$\begin{aligned} \dot{x}_1 &= \frac{1}{2}\mathrm{F}_1(x_2)\cdot x_1 & \dot{x}_3 &= \mathrm{F}_2(x_1)\cdot x_4 \\ \dot{x}_2 &= u_1 & \dot{x}_4 &= u_2 \end{aligned} \quad (2)$$

where $u_1 \in \mathbb{R}^3$, $u_2 \in \mathbb{R}$, and

$$\mathrm{F}_2(x_1) = \begin{bmatrix} 1 + 2(-q_2^2 - q_3^2) \\ 2q_1q_2 + 2q_0q_3 \\ -2q_0q_2 + 2q_1q_3 \end{bmatrix}.$$

2.3 Bounds on States and inputs

We assume that the velocity and acceleration variables have upper and lower bounds. Additional bounds exists for configuration variables. the bounds on configuration and velocity variables can be shown as $x \in X$ and the bounds on acceleration variables $u \in U$, where $X \subset \mathbb{R}^{11}$ and $U \subset \mathbb{R}^4$. Some problems will also have obstacles as excluded regions in configuration variables.

2.4 Obstacles in Configuration Space

In order to consider problems with obstacles in the configuration space, we will rely on standard algorithms for collision detection, and representation of obstacles; see [7, 11]. The collision detection algorithm used in this study is borrowed from Proximity Query Package (PQP); see [6]. The PQP library considers objects formed of surfaces, where each surface is defined as unions of triangles in 3D space.

3 Collocation and nonlinear programming

The optimal control problem of interest can be stated as follows. Given an initial time t_{init} and state x_{init}, determine the final time t_{final}, the control input $u : [t_{init}, t_{final}] \to \mathbb{R}^m$, and the corresponding trajectory $x : [t_{init}, t_{final}] \to \mathbb{R}^n$, that minimize the cost functional

$$\phi(x(t), t) = \varphi(x(t_{final}), t_{final}) + \int_{t_{init}}^{t_{final}} L(x(t), u(t))dt, \quad (3)$$

subject to the differential equation

$$\dot{x} = f(x, u), \quad x(t_{init}) = x_{init}, \quad (4)$$

and the terminal constraints

$$\psi(x(t_{final}), t_{final}) = 0. \quad (5)$$

We present an algorithm based on collocation and nonlinear programming. The idea is to transcribe the optimal control problem into a finite dimensional nonlinear optimization problem to be solved by a nonlinear programming solver. We proceed as follows. The time interval of interest is divided into N segments; the values of input and state variables and of their derivatives over the $(N + 1)$ grid points are called collocation parameters. Cubic polynomials are defined for each variable on each segment using the collocation parameters, so that the cost functional in equation (3) becomes a function of these parameters. Similarly, the differential equation (4) relating state to input variables is discretized using trapezoidal or Simpson's quadrature rules, and becomes a set of nonlinear constraints on the collocation parameters. A successive quadratic program solves numerically the resulting nonlinear program.

3.1 Discretizing the differential equations for the velocity variables

Introduce the uniform partition $\{t_0, t_1, \ldots, t_N\}$, with $t_0 = t_{init}$, $t_N = t_{final}$, and $t_{i-1} + h = t_i$ for all $i \in \{1, \ldots, N\}$. Consider the differential equation relating to angular and translational velocities to the controls:

$$\dot{x}_2(t) = u_1(t), \quad \dot{x}_4(t) = u_2(t). \tag{6}$$

The $x_j(t)$, $j \in \{2, 4\}$ are approximated by the values at the nodes, $x_j^i \approx x_j(t_i)$ for $i \in \{1, \ldots, N\}$. Similarly, $u_k^i \approx u_k(t_i)$ for $k \in \{1, 2\}$, and $i \in \{1, \ldots, N\}$. Next, we introduce the intermediate values

$$y_j^i = x_j(t_{i-1} + h/2), \qquad v_k^i = u_k(t_{i-1} + h/2).$$

We approximate the equations (6) as follows. Compute estimates of y_j^i using cubic interpolation, and v_k^i using linear interpolation, for $j \in \{2, 4\}$, $k \in \{1, 2\}$, i.e.

$$y_j^i = \frac{1}{2}(x_j^i + x_j^{i-1}) + \frac{h}{8}\left[f_j(x^{i-1}, u^{i-1}) - f_j(x^i, u^i)\right]$$
$$v_k^i = \frac{1}{2}(u_k^i + u_k^{i-1}).$$

Next, evaluate the differential equation at the center values (y_j^i, v_k^i), $j \in \{2, 4\}$, $k \in \{1, 2\}$, and integrate across the segment using Simpson's quadrature, i.e.

$$x_j^i = x_j^{i-1} + \frac{h}{6}\left[u_j^{i-1} + 4v_j^i + u_j^i)\right].$$

3.2 Discretizing the differential equations for the configuration variables

Consider the differential equation that describes the rotational and translational velocities

$$\dot{x}_1(t) = \frac{1}{2}\mathrm{F}_1(x_2) \cdot x_1, \quad \dot{x}_3(t) = \mathrm{F}_2(x_1) \cdot x_4. \tag{7}$$

The $x_j(t)$, $j \in \{1, 2, 3, 4\}$ are approximated by the values at the nodes, $x_j^i \approx x_j(t_i)$. To increase the accuracy of the integration, introduce the sub-partition $t_{i,l} = t_i + \frac{h}{K} \cdot (l - \frac{1}{2})$ where $l \in \{1, \ldots, K\}$. As before, we introduce the intermediate values

$$y_j^{i,l} = x_j\left(t_{i-1} + \frac{h}{K}\Big(l - \frac{1}{2}\Big)\right).$$

We approximate the equations (7) as follows. Compute estimates of $y_j^{i,l}$, $j \in \{1, 3\}$ using cubic interpolation

$$y_j^{i,l} = C_{j,3} \cdot \left(\frac{h}{K} \cdot (l - \frac{1}{2})\right)^3 + C_{j,2} \cdot \left(\frac{h}{K} \cdot (l - \frac{1}{2})\right)^2 + C_{j,1} \cdot \left(\frac{h}{K} \cdot (l - \frac{1}{2})\right) + C_{j,0},$$

where

$$\begin{bmatrix} C_{j,0} \\ C_{j,1} \\ C_{j,2} \\ C_{j,3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} x_j^{i-1} \\ f_j(x^{i-1}, u^{i-1}) \cdot h \\ x_j^i \\ f_j(x^i, u^i) \cdot h \end{bmatrix}.$$

Next, evaluate the differential equation at the interpolated center value $y_j^{i,l}$, $j \in \{1, 3\}$, and integrate across the segment using the "Midpoint Rule" and the explicit integration formula for rotation (1), i.e.

$$x_1^i = \prod_{l=1}^{K} \left[\cos\left(\|y_2^{i,l}\|\frac{h}{2K}\right) \cdot I_4 + \frac{1}{\|y_2^{i,l}\|}\sin\left(\|y_2^{i,l}\|\frac{h}{2K}\right)\mathrm{F}_1(y_2^{i,l})\right] \cdot x_1^{i-1}$$
$$x_3^i = \sum_{l=1}^{K}\left[\frac{h}{2K} \cdot \left(\mathrm{F}_2(y_1^{i,l-1}) + \mathrm{F}_2(y_1^{i,l})\right) \cdot y_4^i\right] + x_3^{i-1} \tag{8}$$

where $y_1^{i,0} = x_1^{i-1}$, and for $l \in \{1, \ldots, K\}$ we have

$$y_1^{i,l} = \prod_{m=1}^{l} \left[\cos\left(\|y_2^{i,m}\|\frac{h}{2K}\right) I_4 + \frac{1}{\|y_2^{i,m}\|}\sin\left(\|y_2^{i,m}\|\frac{h}{2K}\right)\mathrm{F}_1(y_2^{i,m})\right] \cdot x_1^{i-1}.$$

Note that the solution to the equation (8) is the exact solution to the equation (2), assuming $x_2(t)$ is constant over the interval of time $[t_{i-1}, t_i]$ and it is equal to $y_2^{i,l}$.

3.3 Resulting nonlinear program

We are now ready to bring it all together and cast the optimal control problem into a nonlinear program. Given initial time t_{init}, and initial state $x(t_{init})$, determine the control sequence $[u^0, \ldots, u^N]$, the corresponding trajectory $[x^0, \ldots, x^N]$, and the final time t_{final} that minimize the cost function

$$\phi(x^0, \ldots, x^N, t_0, \ldots, t_N) = \varphi(x^N, t_N) + \sum_{i=1}^{N} \frac{h}{2}\left[L(x^i, u^i) + L(x^{i-1}, u^{i-1})\right],$$

subject to the collocation conditions

$$\Delta_j^i = 0, \qquad i \in \{1, \ldots, N\}, \;\; j \in \{1, 2, 3, 4\},$$

and the boundary constraints $\psi(x^N, t_N) = 0$, and $x^0 = x_{init}$, where

$$\Delta_1^i = -x_1^i + \prod_{l=1}^{K} \left[\cos\left(\|y_2^{i,l}\|\frac{h}{2K}\right) I_4 + \frac{1}{\|y_2^{i,l}\|}\sin\left(\|y_2^{i,l}\|\frac{h}{2K}\right)\mathrm{F}_1(y_2^{i,l})\right] \cdot x_1^{i-1}$$
$$\Delta_2^i = -x_2^i + x_2^{i-1} + h \cdot v_1^i$$
$$\Delta_3^i = -x_3^i + x_3^{i-1} + \sum_{l=1}^{K}\left[\frac{h}{2K} \cdot \left(\mathrm{F}_2(y_1^{i,l-1}) + \mathrm{F}_2(y_1^{i,l})\right) \cdot y_4^i\right]$$
$$\Delta_4^i = -x_4^i + x_4^{i-1} + h \cdot v_2^i.$$

To solve this nonlinear program, we employ a successive quadratic programming algorithm (SQP) borrowed from the NAG library [14].

3.4 Simulation Results

We consider a simple setting where no obstacles are present and therefore no collision detection is necessary. The objective is to find a minimum-time trajectory, with boundary conditions

$$x_{init} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T,$$
$$x_{final} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 2 & -4 & 1 \end{bmatrix}^T,$$

and subject to the following component-wise constraints on velocities and inputs:

$$\begin{bmatrix} -2 \\ -2 \\ -2 \\ 1 \end{bmatrix} \leq \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ v_x \end{bmatrix} \leq \begin{bmatrix} 2 \\ 2 \\ 2 \\ 3 \end{bmatrix}, \quad \begin{bmatrix} -2 \\ -2 \end{bmatrix} \leq \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \leq \begin{bmatrix} 2 \\ 2 \end{bmatrix}.$$

For this case we have used three segments (i.e., $N = 2$), and eleven sub-segments within each segment (i.e., $K = 10$). A numerical solution is presented in Figure 1. The computation time was 14 seconds on a dual Intel® Pentium II CPU at 350MHz, and Linux RedHat® 6.2 operating system installed. As the figure illustrates, the discretization error is almost negligible.

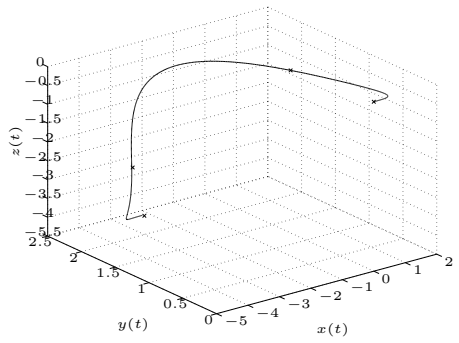

Figure 1: Evolution of the center of mass for the optimal solution. The plots are obtained by numerically integrating the optimal control inputs.

4 Incremental Search Algorithms

Collocation and nonlinear programming are not well suited to planning problems where the environment has complex obstacles. In the robotics literature, the latter problem is referred to as the "piano mover" or the path planning problem.

Most path planning algorithms can be placed into two classes. First, roadmap methods rely on precomputing a finite network, called a roadmap, of collision-free paths that allow future reachability queries over free space, C_{free}, to be answered quickly and efficiently. Instead of precomputing a data structure to assist in future queries, *incremental search methods* perform an iterative search to try to connect the initial configuration to the goal configuration. Both roadmap and incremental methods can be deterministic or randomized [5, 10]. In what follows we shall focus on a recently introduced randomized incremental search method.

4.1 Rapidly-Exploring Random Trees

The RRT algorithm was first introduced in [10] as an efficient incremental search method which quickly and uniformly searches high dimensional spaces that have both algebraic constraints (due to obstacles) and differential constraints. The basic RRT algorithm is described in Figure 2. The algorithm requires a distance

Name: RRT algorithm
Goal: find path from x_{init} to x_{final}

1: initialize a tree T with initial vertex x_{init}
2: **while** distance(x_{final}, T) $> \epsilon$ **do**
3: obtain a random sample p
4: find vertex v $\in$ T with minimum distance to p
5: find path from v to p
6: **if** path is collision free **then**
7: add p to T as new vertex
8: **end if**
9: **end while**

Figure 2: A simplified version of the RRT algorithm [10].

function, a random sampling algorithm, and a collision detection algorithm over the configuration space. The algorithm randomly grows a rapidly-exploring tree until it reaches the final state within a specified tolerance ϵ. In systems with dynamics, the path (roughly) connecting p and v is computed by (i) searching through a finite dimensional sample of available inputs and (ii) selecting the one which best steers the state toward p.

4.2 Simulation Results

In this section we numerically evaluate the RRT algorithm in two settings. The numerical implementation is borrowed from the Motion Strategy Library [9]. The first setting is the same of Section 3. Since the algorithm involves randomization, we present results averaged over 20 trials: we obtained a solution time of 230 ± 200 seconds with the same computer configuration. This is a poor performance compared to collocation and nonlinear programming. Figure 3 displays the center of mass locations of the entire tree and of the resulting connecting path.

The second setting includes two objects and a vehicle, all having the shape of a prism, i.e. each of the surface is formed of triangles as defined in Section 2.4. Thus, the problem becomes that of finding a collision-

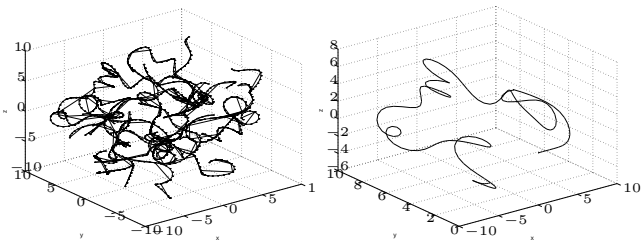

Figure 3: Evolution of the center of mass: the entire tree on the left, the final feasible path on the right.

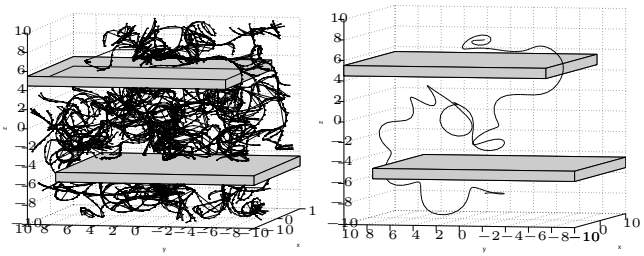

Figure 4: Evolution of the center of mass: the entire tree on the left, the final feasible path on the right.

free path with boundary condition

$$x_{init} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -7.5 & 1 \end{bmatrix}^T$$
$$x_{final} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7.5 & 1 \end{bmatrix}^T,$$

and subject to the same constraints on velocities and inputs as in previous section. For this setting, a solution time of 630 ± 250 seconds is obtained after 20 trials. Dealing with the same setting, the collocation and nonlinear programming algorithm was unable (in our implementation) to provide a feasible solution. Figure 4 displays the center of mass locations of the entire tree and of the resulting connecting path.

5 A hybrid method

In the last two sections we have seen how collocation and nonlinear programming is a very efficient method for planning for systems with differential constraints, which is unfortunately not applicable when some kind of geometrical constraint is introduced. On the other hand, randomized algorithms for path planning are well suited to overcome the difficulty posed by geometrical constraints, although their performance degrades significantly when differential constraints are present. These observation lead us to the design of a new algorithm that merges randomization and nonlinear programming features. The objective is for the new algorithm to inherit the strength of both methods.

5.1 Merging randomization and collocation

As described in Section 4.1, the RRT algorithm depends on a metric to drive the search process. This also means that performance relies significantly on the selection of an appropriate metric. As argued in [10], the optimal metric is the cost-to-go or value function from the minimum-time optimal control problem. Indeed, the value function provides a meaningful measure of the distance between points. Unfortunately, it is impossible for most systems to compute the value function or other metric functions which take into account differential and geometric constraints. The misleading information from the mismatched metric leads to a significant reduction in performance and final accuracy. To reduce the RRT sensitivity to the metric selection, we replace the local planner (which depends on the metric) with a planner utilizing collocation and NLP equation from Section 3.

Table 1: Number of nodes in the tree for second setting after solution, hybrid algorithm vs. RRT

Approach	*With Obstacles*
Hybrid	94 ± 41 (sec)
RRT	4050 ± 1150 (sec)

As local planner, we use a collocation approach based on a single segment (i.e., no partition of the time interval). We assume that the control inputs are second and states variables are third order polynomial function of time. The collision constraints and the bounds on the state variables are checked only after calculating candidate inputs. Furthermore, the nonlinear programming solver is invoked with a maximum number of iterations. All these settings concur to ensure that the total running time of the local planner is low and upper bounded. If the local planner is successful, the input parameters and final state are added to the search tree, similarly to the RRT algorithm. As in Table 1, the proposed hybrid algorithm obtains a connecting path with many fewer nodes than to the original RRT algorithm.

5.2 Simulation Results

We evaluate the algorithm in the same two settings of last section. In both cases the novel algorithm showed better performance than the previous two methods. In the first setting the solution time is 8 ± 9 seconds averaged over 20 trials. In the second setting, the solution time is 170 ± 70 seconds averaged over 20 trials. Figures 5 and 6 display the center of mass of the entire tree and of the resulting connecting path for both settings.

Table 2: Computation time for each approach, hybrid algorithm, RRT, and NonLinear Programming.

Approach	*Without Obstacles*	*With Obstacles*
Hybrid	8 ± 9 (sec)	170 ± 70 (sec)
RRT	232 ± 204 (sec)	652 ± 243 (sec)
NLP	14 (sec)	no solution

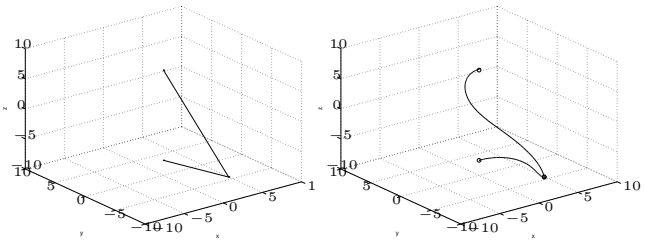

Figure 5: Evolution of the center of mass: the entire tree on the left, the final feasible path on the right.

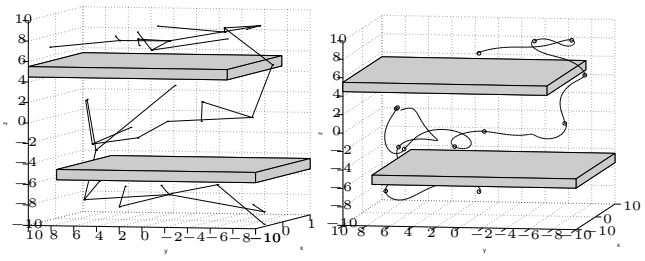

Figure 6: Evolution of the center of mass: the entire tree on the left, the final feasible path on the right.

6 Summary

We have presented a novel hybrid approach to trajectory design for nonlinear dynamical systems evolving in environments with state and input constraints. The proposed algorithm combines the best features of randomized incremental searches and of collocation methods. In two typical settings, the hybrid approach provides better performance than approaches based solely on nonlinear programming or on randomization.

As a final remark, we note numerous aspects of our implementation are suboptimal. A more accurate setup should include the closed-form computation of partial derivatives in the collocation method, and an algorithm for incremental distance computations in the incremental search method. These and other lines of investigation are open for further research.

Acknowledgments: This research was partially supported by NSF Grant IIS-0118146 and ARO Grant DAAD 190110716. It is a pleasure to thank Steve LaValle and Bruce Conway for numerous helpful discussions, and the authors [6, 9] of the freely available MSL and PQP software libraries.

References

[1] J. Barraquand and J-C. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, 10(6):628–649, 1991.

[2] J. T. Betts. Survey of numerical methods for trajectory optimization. *AIAA Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998.

[3] P. J. Enright and B. A. Conway. Discrete approximations to optimal trajectories using direct transcription and nonlinear programming. *AIAA Journal of Guidance, Control, and Dynamics*, 15(4):994–1002, 1992.

[4] E. Frazzoli, M. A. Dahleh, and E. Feron. A hybrid control architecture for aggressive maneuvering of autonomous helicopters. In *IEEE Conf. on Decision and Control*, pages 2471–6, Phoenix, AZ, December 1999.

[5] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional space. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[6] E. Larsen and S. Gottschalk. A proximity query package. `http://www.cs.unc.edu/~geom/SSV`, June 1999. Version 1.2.

[7] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical Report 99-018, Department of Computer Science, University of N. Carolina, Chapel Hill, NC, 1999.

[8] J.-C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, 18(11):1119–1128, 1999.

[9] S. LaValle et al. The motion strategy library. `http://msl.cs.uiuc.edu/msl`, January 2001. Version 1.2.

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

[11] M. C. Lin, D. Manocha, and J. Canny. Fast contact determination in dynamic environments. In *IEEE Int. Conf. on Robotics and Automation*, pages 602–608, San Diego, CA, May 1994.

[12] R. A. Mayo. Relative quaternion state transition relation. *Journal of Guidance and Control*, 2:44–48, 1979.

[13] R. M. Murray, Z. X. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton, FL, 1994.

[14] The Numerical Algorithms Group, Ltd. The NAG C library. `http://www.nag.co.uk`, 2001. Version 6.

[15] G. J. Toussaint, T. Başar, and F. Bullo. Motion planning for nonlinear underactuated vehicles using H^∞ techniques. In *American Control Conference*, pages 4907–4102, Arlington, VA, June 2001.

[16] O. von Stryk and R. Bulirsch. Direct and indirect methods for trajectory optimization. *Annals of Operations Research*, 37(1-4):357–73, 1992.