
Trajectory Planning for Kinematically Controllable Underactuated Mechanical Systems

Prasun Choudhury and Kevin M. Lynch

Northwestern University, Evanston, IL 60208, USA

Abstract. We develop trajectory planners for a class of second-order underactuated mechanical systems called kinematically controllable systems. For kinematically controllable systems, the problem of planning fast collision-free trajectories can be decoupled into the computationally simpler problems of path planning for a kinematic system followed by time-optimal time scaling. This paper describes efficient path planners using randomized algorithms and dynamic programming to solve the path planning problem for the kinematic system. The resulting kinematic paths are time scaled to produce fast trajectories.

1 Introduction

Underactuated mechanical systems are a subset of mechanical systems that have fewer inputs than degrees of freedom. Example systems include underwater vehicles, spacecraft, three joint robot arms with passive joints, etc. In this paper we focus on the trajectory design problem for underactuated mechanical systems.

The problem of trajectory planning for underactuated systems can be approached in different ways. The trajectory planning problem can be cast as a nonlinear optimization problem, where the control history parameters are coefficients of certain basis functions (e.g., splines, polynomials, truncated Fourier series) and the state trajectory is obtained by integrating the dynamic equations of motion. Such an approach cannot be applied blindly, however; the quality of the solution depends intimately on an initial control guess and the local optima in the design space. The number of local optima, and the size of their basins of attraction, are generally extremely difficult to characterize in advance. Further, constraint and objective function evaluation may involve expensive simulations, resulting in slow convergence times. Gradient-based nonlinear optimization methods are inherently local, most appropriate when an approximate solution is available. On the other hand, dynamic programming methods may provide globally optimal solutions, but they are computationally expensive and not likely amenable to real-time implementations. Heuristic randomized searches [13,9] give up on optimality or completeness in an effort to reduce average computation time.

The key to computationally efficient planning of fast trajectories is to make use of the *structure* of the equations of motion of the underactuated

system. As one example, *differentially flat* (or *dynamic feedback linearizable*) systems are a well known class of systems which simplify this problem [7,8,16]. For such systems, it is possible to identify a set of *flat outputs* y equal in dimension to the number of controls such that the state of the system, and the controls, can be expressed as functions of y and its time derivatives. State-to-state trajectory planning for the system reduces to fitting a curve in the output space satisfying initial and terminal constraints on y and its derivatives. The state trajectory and control history are derived directly from this curve.

There are a few limitations to the differential flatness approach to motion planning, however. First, there is no algorithmic procedure to obtain the flat outputs of a system, nor are there general criteria to establish whether flat outputs exist. Second, differential flatness is a non-generic property. In other words, generic perturbations to a flat model of a system render the system “non-flat.” Third, incorporating obstacle or control constraints greatly complicates the trajectory generation problem [7].

Recently, Bullo and Lynch have discovered a new class of mechanical systems called *kinematically controllable* systems [5]. Like differentially flat systems, the structure of these systems simplifies the trajectory planning problem. In this case, we can use a *kinematic reduction* of the mechanical system to plan feasible trajectories (accounting for underactuation constraints) directly in the system’s n -dimensional configuration space (C-space), rather than in the $2n$ -dimensional state space. This is significant for computationally efficient trajectory planning, as complete and resolution-complete algorithms for collision-free motion planning are known to have running times exponential in the dimension of the search space [6,19,20]. In some cases, path planning for kinematically controllable systems can be reduced to closed-form inverse kinematics.

The kinematic reduction allows kinematic constraints (such as obstacles or joint limits) to be dealt with efficiently, perhaps in real-time, as the search for a feasible motion occurs on the n -dimensional C-space instead of the $2n$ -dimensional state space. Because a kinematic reduction encodes “preferred” motions for the system, a path found by the motion planner can be turned into the time-optimal trajectory along the path using a time-scaling algorithm incorporating control constraints [3,18]. Thus the basic approach deals with obstacle and control constraints efficiently and provides sub-time-optimal solutions.

In a previous paper [15], a planner has been described for a kinematically controllable three-joint robot arm with the third joint passive. The planner uses a grid-based *dynamic programming* approach, which is difficult (in both running time and storage requirements) to extend beyond a few degrees-of-freedom. In this paper we tailor *Rapidly Exploring Random Trees* (RRT) [13] to kinematically controllable systems and perform a number of experimental studies showing the computational improvements that can be achieved over

“generic” RRT methods which do not take advantage of the special structure. Numerical results are presented using the enhanced RRT and modified dynamic programming algorithms and the resulting trajectories have been compared to trajectories found using the original RRT planner.

1.1 Brief overview of decoupled trajectory planning

An underactuated second order system can be represented as

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = T(q) \begin{pmatrix} u \\ 0 \end{pmatrix}, \quad (1)$$

where $q \in \mathfrak{R}^n$ is the configuration, $u \in \mathfrak{R}^m$ is the control, $M(q)$ is an $n \times n$ symmetric positive definite inertia matrix, $C(q, \dot{q})\dot{q} \in \mathfrak{R}^n$ is a vector of centrifugal and coriolis forces, and $G(q) \in \mathfrak{R}^n$ is the vector of gravitational forces. $T(q)$ is an $n \times n$ nonsingular matrix whose first m columns describe the action of the actuators. For a manipulator, where the actuators are collocated with the joints, $T(q)$ is the identity matrix. Premultiplying both sides of the equation by $T^{-1}(q)$, we get

$$M'(q)\ddot{q} + C'(q, \dot{q})\dot{q} + G'(q) = \begin{pmatrix} u \\ 0 \end{pmatrix}, \quad (2)$$

where $M' = T^{-1}M$, $C' = T^{-1}C$, $G' = T^{-1}G$.

We assume $m < n$ — the system has fewer actuators than degrees-of-freedom. This means that the last $n - m$ rows in equation (2) represent state-dependent constraints on the feasible accelerations of the system. We could write these last $n - m$ rows in the form

$$M'_0(q)\ddot{q} + C'_0(q, \dot{q})\dot{q} + G'_0(q) = 0. \quad (3)$$

These constraints are what make trajectory planning for generic underactuated systems difficult. In general, trajectory planning for an underactuated system must occur in the $2n$ -dimensional state space, since the motion constraints described in (3) can only be expressed as constraints on tangent vectors on this space. In contrast, a common way to do sub-optimal trajectory planning for a fully actuated mechanical system is to *decouple* the problem by first planning a path on the configuration space, handling any kinematic constraints such as obstacles and joint limits, and then time scaling the path (subject to actuator limits) to get a trajectory.

Recently Bullo and Lynch [5] have discovered that this decoupled procedure can often be applied to underactuated systems, despite the fact that the constraints described in (3) are second-order. Consider a path on the configuration space $q(s) : \mathfrak{R} \rightarrow \mathfrak{R}^n$ parameterized by $s \in [0, 1]$ and a time scaling $s(t)$ which assigns a point on the path for each $t \in [0, T]$. $s(t)$ is twice-differentiable and $\dot{s}(t) > 0$ for all $t \in (0, T)$. Then the trajectory of the

system can be written $q(s(t))$. Plugging $q = q(s)$, $\dot{q} = \frac{dq}{ds}\dot{s}$, $\ddot{q} = \frac{d^2q}{ds^2}\dot{s}^2 + \frac{dq}{ds}\ddot{s}$ into (3), we can write the constraints in the form

$$a(s)\ddot{s} + b(s)\dot{s}^2 + c(s) = 0, \quad (4)$$

where $a(s), b(s), c(s) \in \mathfrak{R}^{n-m}$ are vectors describing the inertial, centrifugal and Coriolis, and gravity terms, respectively, of the system in the constrained directions when restricted to the path. We call the path $q(s)$ a *kinematic motion* if the constraints described in (4) are satisfied for any time scaling, i.e., arbitrary \dot{s}, \ddot{s} . In other words, such a path can be followed at any speed without violating the constraints (3).

A velocity vector field $V(q)$ is a *decoupling vector field* if all paths $q(s)$ satisfying

$$\frac{dq(s)}{ds} = V(q(s))$$

are kinematic motions. A *kinematic reduction* of the original dynamic system (1) is any set of decoupling vector fields. Any controlled trajectory of the kinematic reduction can be followed by a controlled trajectory of the original dynamic system. The original dynamic system (1) is *locally kinematically controllable* if there exist p decoupling vector fields such that the kinematic system

$$\dot{q} = \sum_{i=1}^p V_i(q)w_i \quad (5)$$

$$(w_1, \dots, w_p) \in \{(\pm 1, 0, \dots, 0), (0, \pm 1, 0, \dots, 0), \dots, (0, \dots, 0, \pm 1)\}$$

is locally controllable [5]. Further, these decoupling vector fields can be followed at high speeds, allowing fast motions. The only constraint is that the system must come to rest when switching between the decoupling vector fields, since instantaneous changes in velocity are not possible with finite forces.

For locally kinematically controllable systems, there exists a feasible path between any two zero velocity states in the same open connected component of free C-space, and one can apply any collision-free path planner for the driftless kinematic system (5); see for example [2,12,13]. Because each segment of the resulting path follows one of the decoupling vector fields, the speed along the segment is limited only by actuator limits, not by the under-actuation constraints. Switches between decoupling vector fields must occur at zero velocity, so it is appropriate for the path planner to minimize the number of switches to obtain fast trajectories.

1.2 Description of the Robotic Systems

In this paper, we have tested our motion planning algorithms for two kinematically controllable systems:

- A planar rigid body with a force and torque (Figure 1a)
- A 3D rigid body with three thrusters (Figure 1b)

Planar Body with a Force and Torque: Consider a planar body endowed with a control torque and a body-fixed control force applied through the center of mass. The configuration is $(x, y, \theta) \in \mathbb{R}^2 \times S^1$, and the kinetic energy is $\frac{1}{2}m(\dot{x}^2 + \dot{y}^2) + \frac{1}{2}I\dot{\theta}^2$. The equations of motion for the planar rigid body are given by

$$m\ddot{x} = u_1 \cos \theta, \quad m\ddot{y} = u_1 \sin \theta, \quad I\ddot{\theta} = u_2.$$

Here u_1 is a body-fixed force through the center of mass, u_2 is the torque acting about the center of mass of the rigid body (Figure 1a), m is the mass and I is the moment of inertia of the rigid body. The two decoupling vector fields associated with the 2D rigid body are (a) translational motion along the direction of applied force u_1 (V_1), and (b) rotational motion about the center of mass of the rigid body (V_2). Since the robot has three degrees of freedom, at most three distinct motions (two switches) are required to reach any goal configuration [5] in an obstacle-free environment. For our simulations, we have used different shapes for the planar body robot (see Figure 3).

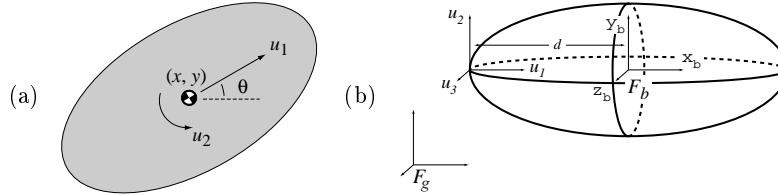


Fig. 1. a) A Planar 2D Rigid Body b) A 3D Spacecraft with Three Thrusters

3D Rigid Body with Three Thrusters: Here a 3D rigid body similar to a spacecraft has been considered with 3 axis-aligned thrusters. The thrusters are located away from the origin at a point along the x_b axis of the 3D rigid body. A frame F_b is attached to the center of mass of the body and aligned with the principal axes of inertia (see Figure 1b). $\omega \in \mathbb{R}^3$ and $v \in \mathbb{R}^3$ are the angular and linear velocity of the spacecraft expressed in inertial frame F_g . The equations of motion for the spacecraft are given by

$$m\dot{v} = F, \quad I\dot{\omega} + \omega \times I\omega = N,$$

where m is the mass of the spacecraft and I is the time-varying inertia matrix expressed in a frame aligned with F_g at the center of mass of the spacecraft. The three external control forces u_1, u_2, u_3 act along lines parallel to the x_b, y_b, z_b axes, respectively, at a point $(-d, 0, 0)$, $d > 0$ in F_b . F and N are the input forces and torques due to u_1, u_2 and u_3 expressed in the inertial frame F_g . The decoupling vector fields for the 3D spacecraft are (a) translation

along the x_b axis (V_1), (b) rotation about the center of percussion in the x_b - y_b plane (V_2) and (c) rotation about the center of percussion in the x_b - z_b plane (V_3) (see [5] for details). The 3D spacecraft, requires at most six motion primitives (five switches) to reach any goal configuration for obstacle-free environments [5]. For our numerical simulations, we have used different shapes for the spacecraft (see Figure 4).

1.3 Problem Statement, Contributions and Organization

The problem can be stated as follows: Given a kinematically controllable underactuated system and a geometric description of the environment, efficiently find a fast collision-free trajectory for the robot.

In this paper, a framework is developed for efficient trajectory planning for kinematically controllable systems. Trajectory planning is reduced to path planning for driftless kinematic systems and time-optimal scaling of the resultant paths to obtain the final trajectory. We use enhanced versions of RRT and dynamic programming based planners for path planning for driftless kinematic systems. The final trajectories have been generated from the above paths in C-space using the optimal time-scaling algorithm described in [3,18]. To judge the efficiency of our path planners, the algorithms have been rigorously tested on various types of underactuated mechanical systems and execution-time and run-time performances have been compared with the standard RRT planner. The new trajectory planning algorithm is computationally efficient (20 to 30 times speedup) compared to the RRT based kinodynamic planner. The execution time of the trajectories generated by the algorithm described in this paper is an order of magnitude faster than the trajectories generated by RRT based kinodynamic planner.

Section 2 describes the RRT and dynamic programming based planning algorithms used in the initial phase of path planning for driftless systems. Section 3 briefly covers the time scaling algorithm for various robotic systems. Experimental results related to the above algorithms as applied to different underactuated systems are described in Section 4. Section 5 compares the execution time of the final robot trajectories for RRT-based kinodynamic planners and the trajectory planning algorithm described in the current paper. Finally a summary of the results of the paper and potential areas of future research are discussed in Section 6.

2 Path Planning Algorithms

As mentioned in Section 1.1, motion planning for second-order dynamic systems can be reduced to planning for a kinematic system with the help of decoupling vector fields. This essentially reduces the dimension of the search space on which planning is performed by a factor of two. In general, the

dimension of the search space for kinodynamic planning is $2n$ and the dimension of the search for kinematic planners is n . Now we will describe in detail the RRT and dynamic programming based motion planning algorithms which we have used for this purpose.

2.1 Rapidly Exploring Random Trees

The RRT algorithm (pseudo-code shown in Table 1 and explained in Figure 2a) has been presented in detail in [13]. Here M is a $2n$ dimensional manifold representing state space (M is of dimension n for C-space). The initial state of the robot, \mathbf{x}_{init} , is chosen as the root of the tree T . For each iteration, a random state $\mathbf{x}_{rand} \in M$ is sampled. The NEAREST-NEIGHBOR function selects the node \mathbf{x}_{near} in the tree T that is closest to \mathbf{x}_{rand} . All the possible inputs U are applied to the selected node \mathbf{x}_{near} . If a newly generated state \mathbf{x}_{new} is collision-free and the distance $\rho(\mathbf{x}_{new}, \mathbf{x}_{rand})$ is the least for all of the collision free states that have been created by applying every input in U to \mathbf{x}_{near} , \mathbf{x}_{new} will be added to the tree T . The CONTROL function returns the input \mathbf{u}_{best} corresponding to the newly added node \mathbf{x}_{new} . The algorithm terminates and returns a successful trajectory when a newly generated collision-free node is within an ε neighborhood of the goal state \mathbf{x}_{goal} , i.e., $\rho(\mathbf{x}_{new}, \mathbf{x}_{goal}) \leq \varepsilon$. The metric function ρ plays a critical role on the performance of RRT planner. The metric used in the RRT algorithm is a weighted Euclidean metric based on linear and angular configuration (angle of rotation for a planar rigid body and quaternion for a 3D rigid body) of the rigid body.

The algorithm can be extended to two trees (RRT-Dual) instead of one as described above. In that case the two trees will be rooted at the initial state \mathbf{x}_{init} and final state \mathbf{x}_{goal} . Both the trees will grow in the above fashion and the algorithm will terminate and return a successful trajectory when the distance between nearest nodes in the two trees is below a threshold ε .

```

BUILD-RRT( $\mathbf{x}_{init}$ )
1   $T.init(\mathbf{x}_{init});$ 
2  for  $k \leftarrow 1$  to  $K$  do
3     $\mathbf{x}_{rand} \leftarrow RANDOM-STATE();$ 
4     $\mathbf{x}_{near} \leftarrow NEAREST-NEIGHBOR(\mathbf{x}_{rand}, T);$ 
5     $\mathbf{u}_{best}, \mathbf{x}_{new}, success \leftarrow CONTROL(\mathbf{x}_{near}, \mathbf{x}_{rand}, T);$ 
6    if success
7      Add vertex  $\mathbf{x}_{new}$  to tree  $T$ ;
8      Add edge  $(\mathbf{x}_{near}, \mathbf{x}_{new}, \mathbf{u}_{best})$  to tree  $T$ 
9      if  $(\rho(\mathbf{x}_{new}, \mathbf{x}_{goal}) \leq \varepsilon)$ 
10     Return the path from start to goal

```

Table 1. Pseudo-Code for the RRT Planner

In Table 1, CONTROL is the function in the motion planner which selects the best possible input \mathbf{u}_{best} among the set of all possible inputs U . In our revised algorithm (RRT-DualKC), we change the heuristic for selecting the best control input. Typically in the RRT algorithm, control inputs are integrated for a fixed time-step Δt throughout entire planning phase. We modify the control criteria so that one can use adaptive time steps during the planning phase. For our planner, we terminate the integration along a flow field if (a) we hit an obstacle or (b) the flow field returns to a neighborhood of any of its previously reached states or (c) the distance from the new state to the goal (for RRT with a single tree) or the distance from the new state to the nearest node in the other tree (for RRT-Dual) increases after an integration step. The primary motivation for using the above heuristics is to reduce the number of switches, i.e., to reduce the number of changes required between different flow fields in the final trajectory. The computation time is expected to be lower since the dimension of the search space is twice compared to that of kinematic planners.

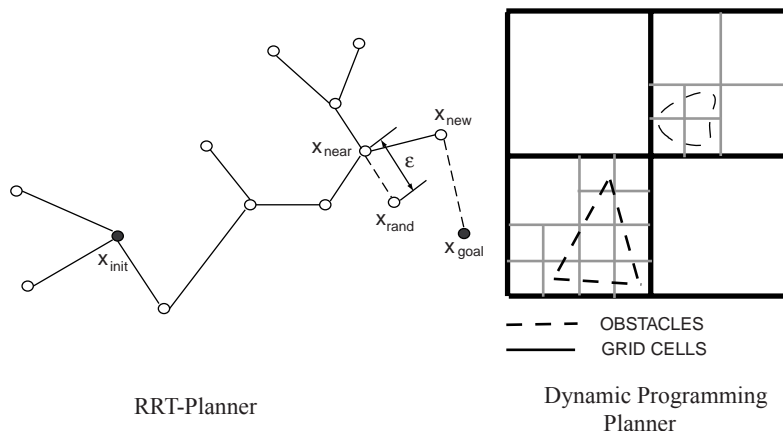


Fig. 2. Rapidly Exploring Random Tree

2.2 Dynamic Programming Based Planning

In this section, we describe in detail a variation of the dynamic programming based motion planner (Multiresolution-DP) which was originally developed by Barraquand and Latombe [2]. The planner is very similar in nature to the A^* search planner using a variation of Dijkstra's algorithm [14]. The overall idea is to discretize the C-space into a grid structure and represent the C-space as a directed graph. The edges between two nodes of the directed graph will represent the cost incurred to reach one state from another. The optimal path in the graph from the start to the goal state will correspond to the least

cost path. Dynamic programming based planners return optimal paths based on the cost function for a particular resolution of the grid.

For our planner, a tree T is used to keep track of all the visited grid cells along with the number of switches it took to reach the grid cell from the initial state (here the COST function is the number of switches between decoupling vector fields). A priority queue Q contains all the inserted grid cells based on the cost metric. In our planning algorithm, we use a multi-resolution strategy where a grid cell can be subdivided into finer cells for better accuracy and collision checking. Similar multi-resolution approaches have also been used by Bohlin [4] for lazy PRM's (Probabilistic Roadmaps) and by Reif and Wang [17] for kinodynamic planning. In our algorithm, each grid cell has been represented by CELL data structure which consists of i, j, \dots indices (each index corresponds to a dimension of the C-space), pointers to cells of finer resolution (if any), the COST to reach the grid cell from the initial state, the distance of the current grid cell to the goal state, a boolean flag that indicates whether the current grid cell has been visited and a boolean flag that indicates whether the grid cell is in collision or not. The pseudo-code of the dynamic programming based planner is described briefly in Table 2.

DYNAMIC PROGRAMMING PLANNER (\mathbf{x}_{init})

```

1   $T \leftarrow \text{NULL}, Q \leftarrow \text{NULL};$ 
2   $\text{GRID-CELLS} \leftarrow \text{DISCRETIZE-C-Space}$ 
3  Mark each CELL as COLLISION or FREE;
4  Mark each CELL as UNVISITED;
5   $T.\text{init}(\mathbf{x}_{init}); Q.\text{init}(\mathbf{x}_{init});$ 
6  while numNodes  $\leq$  MAXNODES and  $Q \neq \text{NULL}$ , do
7       $\mathbf{x} \leftarrow Q.\text{top}$ 
8      for all inputs
9           $\mathbf{x}_{new} \leftarrow \text{CONTROL}(\mathbf{x});$ 
10         if  $\mathbf{x}_{new} \neq \text{COLLISION}$ 
11              $\mathbf{x}_{new} \leftarrow \text{VISITED}$  and Update COST;
12             Add  $\mathbf{x}_{new}$  to tree  $T$ 
13             Add  $\mathbf{x}_{new}$  to priority queue  $Q$ 
14             if  $(\rho(\mathbf{x}_{new}, \text{GOAL}) \leq \varepsilon)$ 
15                 Return the path from start to goal
16         if  $\mathbf{x}_{new} = \text{COLLISION}$ 
17             SUBDIVIDE CELL;
18             Update COLLISION and COST;
```

Table 2. Pseudo-Code for the Multiresolution Dynamic Programming Based Path Planner

The C-space is initially discretized into a relatively coarse resolution grid (dark lines in Figure 2b). All the coarse resolution grid cells are marked UNVISITED and COLLISION or FREE (based on whether a grid cell is in

collision with the obstacles). The initial state \mathbf{x}_{init} is added to the tree T and the priority queue Q . While traversing the tree T , we apply all possible inputs to the current grid cell (this action is carried out by the function CONTROL in Table 2). We mark all the new collision-free grid cells that are reached as a result of applying different inputs. Heuristics described in Section 2.1 are used to perform integration with adaptive step-size during the planning phase. After each CONTROL step, we insert the new VISITED grid cells to the priority queue Q based on its COST. We continue the iteration until we reach a user-prescribed maximum number of nodes or we find a path to reach the goal. During each iteration, the least cost grid cell is popped out of the priority queue Q and the CONTROL function is applied to that cell. Grid cells in collision are SUBDIVIDED into finer cells (light lines in Figure 2b) for better accuracy and efficient collision detection. The planner terminates and returns a successful trajectory when a newly generated collision-free node \mathbf{x}_{new} is within an ε neighborhood of \mathbf{x}_{goal} .

The storage complexity for the dynamic programming algorithm is exponential in the dimension of the search space and hence the algorithm might not be practical for high-dimensional C-space or fine resolution grids. As an example, even a relatively coarse resolution of 20 divisions along each dimension of 6D C-space results in 64 million cells.

3 Time Scaling

Geometric paths can be time scaled to yield a time-optimal trajectory along the path using algorithms described in [3,18]. For the two examples studied in this paper, the time-scaling problem is trivial. Each motion segment (between decoupling vector field switches) is time-scaled individually. For the 2D rigid body, the time-optimal scalings use bang-bang control profiles. Similarly, for the 3D rigid body, time-optimal translations use a bang-bang profile. For rotations, the trajectory is slightly more complicated. In general, the thruster providing non-zero torque follows a bang-off-bang profile. As the angular velocity increases from zero, the thrust through the center of mass (u_1) increases from zero to maintain the stationarity of the center of percussion. When this thruster is saturated, the torque thruster (u_2 or u_3) turns off. For short rotations, the thruster through the center of mass never saturates, and the torque thruster profile is simply bang-bang.

4 Experimental Results

The enhanced RRT and dynamic programming based planners have been tested on different kinematically controllable systems with environments of varying geometric complexity. We compared the running time and execution time of the planners with the original RRT planner. The motion planning algorithms have been developed on the framework provided within the *Motion*

Strategy Library (MSL) [1]. All the algorithms have been implemented on a 400MHz Intel Pentium III PC with 256 MB of memory (we have also implemented the algorithms on a faster 2GHz Pentium IV PC with 512 MB of memory and the nature of the results are similar).

All the motion planning algorithms which have been used for our numerical experiments are enumerated below for clarity.

- RRT-Dual: The original RRT algorithm with two trees rooted at the initial and final states [13]. It performs kinodynamic planning directly on the state space.
- RRT-DualKin: The original RRT algorithm with two trees rooted at the initial and final configurations, used for kinematic planning on C-space.
- RRT-DualKC: Enhanced RRT-DualKin planner (used for kinematic planning) which allows adaptive step size integration (described in Section 2.1).
- Multiresolution-DP: Extension of the dynamic programming based planner [2] (used for kinematic planning) which uses cells of variable size (described in Section 2.2).

The results reported in Section 4.1 compare the efficiency of RRT-DualKC and Multiresolution-DP planner with the RRT-DualKin algorithm for first-order or kinematic systems. The output of these planning algorithms are kinematic paths in C-space. The results described in Section 5 correspond to the performance of the RRT-Dual kinodynamic planner and the trajectory planner consisting of planning in C-space using the RRT-DualKC planner and time scaling the kinematic paths.

4.1 Results of Numerical Simulations

Each planner was run 30 times for a specific robot in a particular environment. The computation times of different planners are described in Table 3. The reported times are average computation times.

The paths obtained using different types of planners for the 2D rigid body for environments with different levels of geometric complexity are shown in Figure 3. For the 2D rigid body in a maze like environment with complicated geometry (not shown in figure), the RRT-DualKin planner was unable to find feasible paths for most cases after generating 50,000 nodes. In general, the number of generated nodes (and hence the computation time) is less for the RRT-DualKC algorithm. On the other hand, the dynamic programming planner which is based on grid search technique is generally slower. The number of switches between the decoupling vector fields are lower for RRT-DualKC and Multiresolution-DP planners as compared to the original RRT-DualKin planner which does not try to minimize switches. As seen from Figure 3, the Multiresolution-DP planner tends to produce paths with the fewest switches between decoupling vector fields, especially for environments cluttered with obstacles. This is also an expected result as dynamic programming planners output optimal paths for the given grid resolution. From the

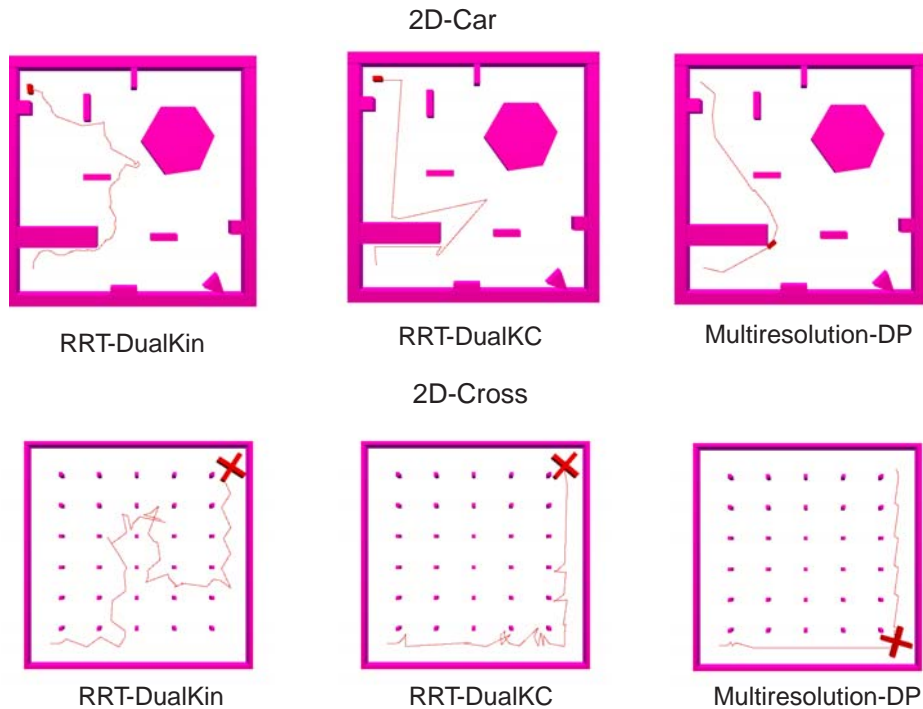


Fig. 3. Motion plans for 2D Rigid Body with various planning algorithms. The number of switches is least for Multiresolution-DP planner.

numerical experiments, we see that the planner took around eight switches for environments with fewer obstacles and the number of switches increased with the complexity of the geometric environment.

Figure 4 shows the output paths for the 3D rigid body in two different environments. The computation times of different planners are described in Table 3. It was not possible to use dynamic programming based planners for 3D spatial environments as the memory required to store the CELL data structure for 6D C-space exceeded the memory available in the computer. Similar to the case for planning for 2D rigid body, RRT-DualKC planner turned out to be more computationally efficient and generated paths with fewer switches between decoupling vector fields. Path shortening techniques based on random sampling algorithms [12] or nonlinear optimization meth-

Robot and Environment Description	Type of Planner			Type of Planner			Type of Planner		
	RRT-DualKin			RRT-DualKC			Multiresolution DP		
	No. of Nodes	No. of Swit-ches	Running Time	No. of Nodes	No. of Swit-ches	Running Time	No. of Nodes	No. of Swit-ches	Running Time
<i>2D Rigid Body with a force and a torque</i>									
2D-Car	1062	77	4.81	152	8	1.96	4144	9	13.4
2D-Cross	10577	152	951.94	1611	31	26.99	12893	15	1041.52
2D-Maze	—	—	—	1863	22	75.02	4031	11	280.02
<i>3D Spacecraft with 3 Thrusters</i>									
3D-Torus	3027	21	16.21	368	9	2.41	N/A	N/A	N/A
3D-Cage	14450	119	492.56	5061	41	84.54	N/A	N/A	N/A
3D Hole	22315	127	1467.89	4168	47	62.82	N/A	N/A	N/A
3D Passage	17111	271	989.31	2766	91	31.83	N/A	N/A	N/A

Table 3. Running Times for Different Planning Algorithms for 2D Rigid Body and 3D Spacecraft

ods [10] can be used as a post-processing step to further improve the quality of the planner.

5 Trajectory Planning and Comparison with Kinodynamic Planning

In this section, we evaluate the performance of the RRT-Dual kinodynamic planner and the trajectory planner comprised of planning in C-space using the RRT-DualKC planner and time scaling the paths to obtain the final trajectories of the robot. Table 4 compares the computational efficiency of the two planning algorithms. In addition, details are provided for the execution time of the robot for the two planning algorithms. While implementing the time-scaling algorithm for the kinematically controllable systems, certain thrust limits have been chosen for the systems. For the 2D rigid body the magnitude of the maximum force/torque that can be applied by the linear and rotational thruster has been restricted to 100N and 150N-m respectively. The mass and the inertia of the 2D rigid body are 1kg and 1.5kg-m². For the 3D spacecraft, the magnitude of maximum force for the linear thruster is 300N and the magnitude of maximum torque that can be applied by the rotational thrusters is 450N-m. The mass of the 3D spacecraft is 1kg, the moment of inertia of the spacecraft is 1.5kg-m², and the thrusters are located at a distance of 1.5m away from the origin along x_b axis. We have considered the maximum magnitude of the force/torque as the input for RRT-Dual kinodynamic planner (one actuation input at a time).

For each experiment, the average computation time over 30 trials is much higher for the RRT-Dual kinodynamic planner as compared to the decoupled trajectory planner. For environments with a high degree of geometric

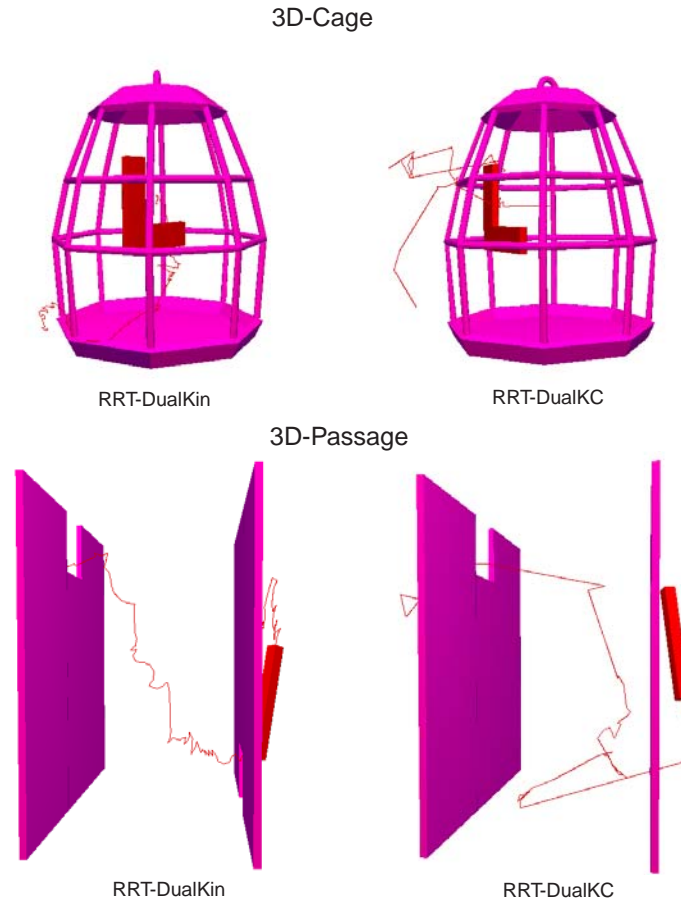


Fig. 4. Motion plans for 3D Spacecraft with three thrusters using various planning algorithms for two complex geometric environments. RRT-DualKC produces paths with lesser number of switches than RRT-DualKin planner.

complexity (maze like environments for 2D rigid body or environments with two narrow passages for 3D rigid body), the RRT-Dual kinodynamic planner was unable to find successful trajectories after generating 50,000 nodes. For the decoupled trajectory planner, computation cost is governed by the time required to plan kinematic paths in C -space (computation time required to time-scale the kinematic paths is negligible).

Execution times i.e., the time the robot will take to execute the trajectory, are much lower for the decoupled trajectory planner than for RRT-Dual kinodynamic planner. Overall, the execution time for the decoupled trajectory planner is roughly 10 times faster and requires much less computation

time (20-30 times speedup in computational efficiency) than RRT-Dual kinodynamic planner.

The computation time reported in this paper for kinodynamic planning is substantially higher than the computation time reported in [13] and the algorithm in [13] seem to work for environments with narrow passages. This might be attributed to the fact that the kinodynamic planning algorithm in MSL is not the most optimized implementation of the RRT-Dual algorithm.

<i>Robot and Environment Description</i>	<i>No. of Triangles</i>		<i>RRT-Dual Kinodynamic Planner [13]</i>		<i>Trajectory Planner: Time Scaling + RRTDualKC</i>		
	<i>Robot</i>	<i>Obstacles</i>	<i>Running Time(s)</i>	<i>Exec. Time(s)</i>	<i>Running Time(s)</i>	<i>No. of Switches</i>	<i>Exec. Time(s)</i>
Planar Body, 2D Grid	4	228	789.6	31.3	27.9	31	3.1
Planar Body, 2D Maze	2	296	N/A	N/A	77.01	22	2.2
3D Spacecraft, Cage	132	6192	2253.1	27.6	85.31	41	2.69
3D Spacecraft Narrow Slot	66	240	1123.3	38.4	63.11	47	2.91
3D Spacecraft, Narrow Passage	36	384	N/A	N/A	32.43	91	6.97

Table 4. Execution Time for RRT-based Kinodynamic Planner and Decoupled Trajectory Planner

6 Conclusion

In this paper, we have presented an algorithm for fast trajectory planning for kinematically controllable systems. Trajectory planning has been decoupled into simpler problems of planning kinematic paths in C-space followed by time-optimal time scaling based on the actuator constraints. The trajectory planning algorithm has been compared with original RRT-based kinodynamic planner [13]. Generally, the enhanced RRT based planner turned out to be computationally efficient and generated much faster trajectories.

In future, we would like to study the performance of our trajectory planner for other kinematically controllable systems such as a 3-R robot manipulator with a passive joint [15,5]. In addition, we will explore motion planners based on exact inverse kinematics for point-to-point motions for kinematically controllable systems [5]. These motion planners will use probabilistic roadmaps [11] with a local planner implementing point-to-point motions from inverse kinematics. Along with that, we want to pursue our research in areas related to practical multi-resolution planners for motion planning of kinodynamic systems [17,19,20]. Furthermore, we will focus on smoothing the

paths generated by the RRT planner [12,10] to alleviate jerky motions and reduce the time required by the robot to execute motion along collision-free trajectories.

Acknowledgment

We thank Steve LaValle and Peng Cheng for many stimulating discussions on RRT and motion planning and for their help with the Motion Strategy Library. This work was funded in part by NSF grants IIS-9875469 and IIS-9811571.

References

1. <http://msl.cs.uiuc.edu/msl/>, "Motion Strategy Library," Dept. of Computer Science, Univ. of Illinois, Urbana-Champaign.
2. J. Barraquand and J. C-Latombe, "Nonholonomic multibody mobile robots: Controllability and motion planning in presence of obstacles," *Algorithmica*, vol. 10, no. 2-3-4, pp. 121-155, 1993.
3. J. E. Bobrow, S. Dubowsky and J. S. Gibson, "Time-optimal control of robotic manipulators along specified paths," *International Journal of Robotics Research*, vol. 4, no. 3, pp. 3-17, 1985.
4. R. Bohlin, "Path planning in practice; lazy evaluation on a multiresolution grid," in *Proc. IEEE Conference on Intelligent Robots and Systems*, October 2001.
5. F. Bullo and K. M. Lynch, "Kinematic controllability for decoupled trajectory planning in underactuated mechanical systems," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 4, pp. 402-412, 2001.
6. B. Donald, P. Xavier, J. Canny and J. Reif, "Kinodynamic motion planning," *Journal of Association for Computing Machinery (ACM)*, vol. 40, no. 5, pp. 1048-1066, 1993.
7. N. Faiz, S. Agrawal, and R. M. Murray, "Differentially flat systems with inequality constraints: An approach to real-time feasible trajectory generation," *AIAA Journal of Guidance, Control and Dynamics*, vol. 24, no. 2, pp. 219-227, 2001.
8. M. Fliess, J. Levine, P. Martin and P. Rouchon, "Flatness and defect of nonlinear systems: Introduction, theory and examples," *International Journal of Control*, vol. 61, no. 6, pp. 1327-1361, 1995.
9. D. Hsu, R. Kindel, J.-C. Latombe and S. Rock, "Randomized motion planning with moving obstacles," *International Journal of Robotics Research*, vol. 21, no. 3, pp. 233-255, 2002.
10. T. Karatas and F. Bullo, "Randomized searches and nonlinear programming in trajectory planning," in *Proc. IEEE Conference on Decision and Control*, pp. 5032-5037, 2001.
11. L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566-580, 1996.
12. J.-P. Laumond, P. Jacobs, M. Taix and R. M. Murray, "A motion planner for nonholonomic mobile robots," *IEEE Transactions of Robotics and Automation*, vol. 10, no. 5, pp. 121-155, 1993.

13. S. M. LaValle and J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378-400, 2001.
14. K. M. Lynch and M. Mason, "Stable pushing: mechanics, controllability and planning," *International Journal of Robotics Research*, vol. 15, no. 6, pp. 533-556, 1996.
15. K. M. Lynch, N. Shiroma, H. Arai and K. Tanie, "Collision free trajectory planning for a 3-dof robot with a passive joint," *International Journal of Robotics Research*, vol. 19, no. 12, pp. 1171-1184, Dec. 2000.
16. P. Martin and P. Rouchon, "Any controllable (driftless) system with 3 inputs and 5 states is flat," *System and Control Letters*, vol. 25, no. 3, pp. 167-173, 1995.
17. J. Reif and H. Wang, "Non-uniform discretization approximations for kinodynamic motion planning and its applications," in *Workshop on the Algorithmic Foundation of Robotics (WAFR,96)*, (J. P. Laumond and M. Overmars eds.), pp. 97-112, 1996.
18. K. G. Shin and N. D. McKay, "Minimum time-control of robotic manipulators with geometric path constraints," *IEEE Transactions on Automatic Control*, vol. 30, no. 6, pp. 531-541, 1985.
19. P. Xavier and B. Donald, "Provably good approximation algorithms for optimal kinodynamic planning: robots with decoupled dynamics bounds," *Algorithmica*, vol. 14, no. 6, pp. 443-479, 1995.
20. P. Xavier and B. Donald, "Provably good approximation algorithms for optimal kinodynamic planning for cartesian robots and open chain manipulators," *Algorithmica*, vol. 14, no. 6, pp. 480-530, 1995.