2014 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA

Probabilistically Complete Kinodynamic Planning for Robot Manipulators with Acceleration Limits

Tobias Kunz and Mike Stilman

***Abstract*— We introduce acceleration-limited planning for manipulators as a middle ground between pure geometric planning and planning with full robot dynamics. It is more powerful than geometric planning and can be solved more efficiently than planning with full robot dynamics. We present a probabilistically complete RRT motion planner that considers joint acceleration limits and potentially non-zero start and goal velocities. It uses a fast, non-iterative steering method. We demonstrate both the power and efficiency of our planner using the problem of hitting a nail with a hammer, which requires the robot to reach a given goal velocity while avoiding obstacles. Our planner is able to solve this problem in less than 100 ms. In contrast, a purely geometric planner is unable to hit the nail at the desired velocity, whereas a standard kinodynamic RRT is multiple orders of magnitude slower.**

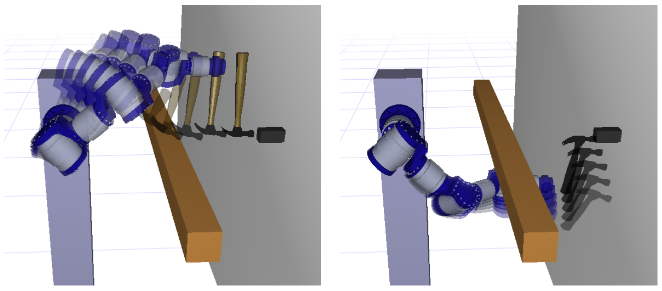

(a) DIMT-RRT (b) Geometric RRT

Fig. 1. The acceleration-limited planner (a) is able to hit the nail at the correct velocity while the geometric planner (b) is not.

I. Introduction

Sampling-based planners like RRTs [1] have been successfully applied to geometric path planning problems for manipulators. In geometric domains they are both probabilistically complete and can quickly find solutions in high-dimensional configuration spaces like that of a 7-DOF manipulator in practice. While the probabilistic completeness of RRTs in geometric domains is based on their ability to densely explore the whole configuration space, in practice, their computational efficiency is based on the ability to find a solution without exploring much. We claim that the efficient performance of RRTs in geometric domains is largely based on the availability of a fast-to-compute steering method. A steering method is able to exactly and optimally connect any two states while ignoring obstacles. In the case of geometric path planning this steering method is trivial and returns a straight line. The distance function used in the RRT is based on the steering method, returning the length of the straight-line path, i.e. the Euclidean distance. The ability of the steering method to exactly connect two states is useful for improving a path through shortcutting, to connect the two trees in a bidirectional RRT planner, for the rewiring step of an RRT* [2] or for exactly reaching a goal configuration.

By ignoring velocities completely, geometric planning makes the assumption that the direction of motion can be changed instantaneously, which is impractical in reality at high speeds. To take the robot's actual capabilities of changing its velocity into account, we need to include the joint velocities in the state space and add differential constraints. RRTs were initially introduced to deal with arbitrary differential constraints. Since efficient steering methods do not exist for arbitrary differential constraints, the standard kinodynamic RRT [3–6] does not make use of a steering method and instead uses a distance function, usually Euclidean, and an incremental simulator to direct the growth of the tree. The Euclidean distance function is uninformed and does not incorporate any knowledge about the system dynamics. This leads to the RRT not being able to explore the state space efficiently. In addition, the most common variant of the kinodynamic RRT is not probabilistically complete in general [7].

We propose acceleration-limited planning as a middle ground between pure geometric planning and planning with full dynamics. Acceleration-limited planning is more powerful than pure geometric planning and at the same time can be solved with greater computational efficiency than full dynamic planning because a steering method is available. To our knowledge we present the first computationally efficient and probabilistically complete sampling-based planner for manipulators that deal with non-zero start or goal velocities while considering actuator limitations. We present a fast, non-iterative steering method that can efficiently solve the boundary value problem.

While being more computationally efficient, acceleration-limited planning is less powerful than considering the full dynamics. But it can solve problems geometric planning cannot. Unlike a geometric planner our planner is able to find a trajectory that satisfies acceleration limits for problems that include non-zero start and/or goal velocities. We evaluate our planner on the task of hitting a nail with a hammer, which requires a non-zero goal velocity. Fig. 1 shows the final parts of the trajectories planned by our acceleration-limited planner (a) and a geometric planner (b). Our planner is able to plan a collision-free trajectory such that the hammer tip reaches the nail with a velocity parallel to the nail. Moreover, our planner finds a feasible solution to this problem in less than 100 ms. In contrast, the geometric planner ignores the goal velocity completely and plans an approach from the side

The authors are with the Institute for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA 30332, USA. Email: tobias@gatech.edu

of the nail, which is not going to drive the nail into the wall.

Another application that requires a non-zero start velocity is online replanning as the robot is already in motion at the start of the new replanned trajectory. Also, when using a probabilistically optimal planner like an RRT*, planning with acceleration limits is beneficial even if both start and goal velocity are zero, because the planner can take the acceleration limits into account when optimizing. If only the geometric path gets globally optimized, the trajectory resulting from following that path might not be optimal. The latter two applications of acceleration-limited planning are not examined in this paper and are left for future work.

Even though the steering method is fast to compute, it is not as trivial to implement as the Euclidean distance. Some existing work uses a similar steering method for distinct applications (see Sec. III-E). However, none of the papers present the algorithm for the steering method in detail and correctly. Thus, an additional contribution of this paper is the detailed and correct presentation of the steering method, which can solve the acceleration-limited boundary value problem efficiently, in Sec. IV.

In Sec. II we formally define the problem addressed by this paper. In Sec. III we contrast our planner to existing work in greater detail. Our planner, described in Sec. V, combines the RRT algorithm with the steering method. Finally we present the results of our experiments in Sec. VI.

II. Problem Definition

Given a set of start states and a set of goal states, our RRT planner finds a trajectory that connects any of the start states with any of the goal states. The trajectory is a valid trajectory for the following double integrator system with state vector $[\boldsymbol{p}, \boldsymbol{v}]$, which consists of joint positions $\boldsymbol{p}$ and velocities $\boldsymbol{v}$, and input vector $\boldsymbol{a}$

$$\dot{\boldsymbol{p}} = \boldsymbol{v} \qquad \dot{\boldsymbol{v}} = \boldsymbol{a} \tag{1}$$

The solution trajectory must also satisfies the following constraints on position, velocity and acceleration and must be collision-free.

$$\boldsymbol{p}_{\text{min}} \leq \boldsymbol{p} \leq \boldsymbol{p}_{\text{max}} \tag{2}$$

$$-\boldsymbol{v}_{\text{max}} \leq \boldsymbol{v} \leq \boldsymbol{v}_{\text{max}} \tag{3}$$

$$-\boldsymbol{a}_{\text{max}} \leq \boldsymbol{a} \leq \boldsymbol{a}_{\text{max}} \tag{4}$$

$$\boldsymbol{p} \in \mathcal{C}_{\text{free}} \tag{5}$$

The steering method solves the relaxed problem that ignores Eq. 2 and 5. But in addition the steering method minimizes the trajectory duration. We call this relaxed problem that ignores obstacles and minimizes trajectory duration *Double-Integrator Minimum Time (DIMT)*.

III. Related Work

A. Adaption of a Geometric Path

One approach of dealing with actuator limitations is to decompose the planning problem into two steps, first planning a geometric path and then adapting this path and turning it into a time-parameterized trajectory such that actuator limitations are satisfied. As long as the feasible accelerations always contain an ϵ-neighborhood of zero, any geometric path can be followed by the robot. There has been much work on time-optimally following a given path such that either acceleration or force/torque limits are satisfied. [8–15]. Hauser et al. [16] use limited-acceleration shortcuts to turn a path into a smooth trajectory. They use a steering method similar to the one presented in this paper to generate the shortcuts. However, none of the post-processing methods is able to deal with a non-zero start or goal velocity. In addition, if optimality was considered, the post-processing would not result in an optimal trajectory even if the original path was optimal.

B. Kinodynamic RRT with Incremental Simulator

The kinodynamic RRT as presented in [3–6] is able to deal with general differential constraints including problems involving full robot dynamics as well as the acceleration-limited problem considered here. Since efficient steering methods are generally not available, the kinodynamic RRT does not make use of them, but instead only requires an incremental simulator, which can forward simulate the system for a given time step and control input. It also requires a distance function.

The kinodynamic RRT extends the tree toward a sample by selecting the closest node using the provided distance function. The tree is grown from that node by simulating the system forward for a small amount of time. The control input for that is either selected randomly or multiple input trajectories are forward-simulated and the one is picked that gets the tree closest to the sample using the provided distance function. The first option is proven to be probabilistically complete but does not lead to a directed growth of the tree. The second and more common option is time-consuming in high-dimensional spaces and is not probabilistically complete in general [7]. It is unknown whether it is probabilistically complete for the problem considered here. Both options do not allow for exact connection of states and thus do not allow exact goal states, shortcutting or rewiring being used by the algorithm.

The performance of the kinodynamic RRT depends on a good distance function. In the lack of a fast-to-compute and high-quality distance function, most work uses a Euclidean distance, which does not give a good approximation of the real cost and might hinder the efficient growth of the tree. Unlike our steering method and the distance function derived from it, the Euclidean distance is not even aware of the fact that velocity is the derivative of position. A weighted Euclidean distance function [5] might improve performance but requires tuning the weights to the specific problem. Our distance metric, in contrast, is parameter-free.

C. LQR-RRT

Recently, there have been efforts [17–20] to use a more accurate distance metric by using LQR methods from optimal linear control. These algorithms linearize the system dynamics around the current node or sample. Together with a quadratic cost functional this leads to an LQR problem, which can be solved for the optimal trajectory and cost. Sampling of control inputs is avoided in [18–20]. The solution to the

LQR problem gives an optimal trajectory for the linearized system, which can be applied to the original nonlinear system. For nonlinear systems LQR requires linearization. Thus the distance function is only a good approximation in the neighborhood of the linearization point, which hinders efficient exploration. For linear systems like the acceleration-limited problem considered here LQR is able to consider the differential constraints exactly. However, solving the finite-time LQR problem in [17, 19, 20] requires numerical integration, which is computationally expensive. In contrast, our distance function is non-iterative and fast. In addition, the LQR problem can only deal with quadratic costs on the input and the state, but not limits like our problem involves. Transforming limits into costs requires parameter tuning.

D. *Trajectory Optimization*

Trajectory optimizers like e.g. CHOMP [21] and STOMP [22] can handle very general dynamics and task constraints and can produce smooth trajectories. They can be applied to problems our approach cannot. However, they are prone to getting stuck in local optima. Since obstacles and other hard constraints are also represented as cost, a locally-optimal trajectory might be infeasible. Getting a trajectory optimizer to output feasible and good-quality trajectories often involves tuning parameters to the specific problem. For example the weights of different parts of the cost functional are crucial. In contrast, our algorithm does not have any parameters that need to be tuned. CHOMP and STOMP use randomness to evade local optima, the amount of which is subject to tuning again. Our planner could be used to provide a better initial guess to a trajectory optimizer.

E. *Double-Integrator Minimum Time*

Some previous work also exploits the fact that the double-integrator minimum time problem can be solved efficiently. Kröger et al. [23–25] use it for online trajectory generation. However, their work does not plan or check collisions. They just generate a time-optimal trajectory to the given goal state and follow it in real-time. They assume that a planner or some other program on top of their controller chooses the intermediate goal states wisely. Their early work [23] only deals with a simpler case of the DIMT problem assuming a zero velocity at the goal (Type I in their nomenclature). Their later work [24, 25] deals with a more general and complex problem, which includes jerk limits (Type IV). Since that problem is very complex, they only describe part of its solution in detail and never describe the solution to the problem we are solving, which is a Type II problem in their nomenclature. Based on their high-level description we attempt to note the differences to their algorithm in Sec. IV.

Hauser et al. [16] use the solution to the DIMT problem to smooth a given path using given acceleration limits. They try to solve the same DIMT problem as our approach. While [16] describes the algorithm in detail, parts of their algorithm and equations are incorrect, as will be explained in Sec. IV.

IV. Steering Method: Double-Integrator Minimum Time (DIMT)

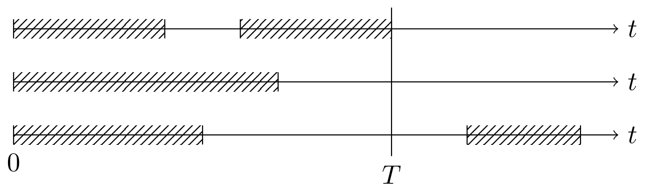


Fig. 2. Determining the overall minimum time t from the individual DOF's minimum times and infeasible time intervals

This section presents the steering method that is used to connect states within the RRT and that provides a distance function for selecting the nearest neighbor. The steering method is able to exactly connect two given states, which consist of positions and velocities of all joints. The steering method returns the time-optimal trajectory satisfying the given velocity and acceleration limits The only constraints the steering method ignores are obstacles in the workspace and position limits. These are handled by the RRT algorithm.

The algorithm for finding the time-optimal trajectory consists of two steps. First, we calculate the minimum time T at which all DOFs can reach their goal state simultaneously. Second, we calculate trajectories for each individual DOF to reach their goal state at that time T.

Note that the overall minimum time is different from the minimum times of the individual joints. Furthermore, unlike claimed in [16], the overall minimum time is not the maximum of all the individual minimum times. In addition to having a minimum time to reach its goal state, each individual DOF also has up to one infeasible time interval, which is greater than the minimum time, but during which the DOF cannot reach its goal state [24]. Section IV-B describes how to calculate the minimum time for an individual DOF and section IV-C how to calculate the infeasible interval. After these have been determined for every DOF, we find the overall minimum time as the smallest time that is at least as great as the greatest individual minimum time and is not within any infeasible time interval (see Fig. 2).

After the overall minimum time T has been determined, section IV-D describes how to calculate a trajectory for each individual DOF to reach its goal state at time T.

All the following subsections are only considering one individual DOF.

A. *Solving Quadratic Equations*

This section describes how to solve a quadratic equation explicitly for either the greater or the lesser one of the two solutions without having to calculate and compare both solutions to figure out whether they satisfy given constraints as it is done in [16].

The solution to the quadratic equation

$$ax^2 + bx + c = 0 \tag{6}$$

is given by

$$q = -\frac{1}{2}\left(b + \operatorname{sgn}(b)\sqrt{b^2 - 4ac}\right) \tag{7}$$

$$x_1 = \frac{q}{a} \qquad\qquad x_2 = \frac{c}{q} \tag{8}$$

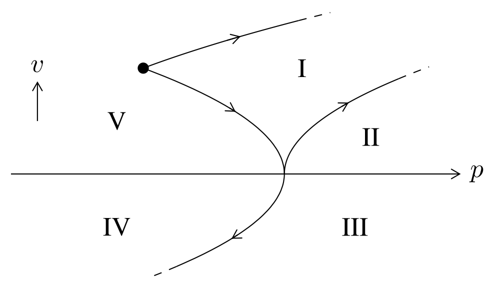


Fig. 3. Regions of the phase plane the goal state can be in relative to the start state. The shown separating trajectories are at an acceleration limit. Without loss of generality this figure assumes a positive start velocity.

The solutions satisfy $|x_1| \geq |x_2|$. If $\mathrm{sgn}(a) = \mathrm{sgn}(b)$, then $x_1 \leq x_2$. Otherwise, $x_1 \geq x_2$.

B. 1-DOF Minimum Time

The minimum-time trajectory for a single DOF consists of either 2 or 3 segments. The first and the last one have a constant acceleration of a_1 and a_2 respectively, whereas the acceleration must be at one of the acceleration limits. The optional middle segment has a constant velocity at one of the velocity limits. We need to determine the sign of a_1 and a_2 and whether the minimum-time trajectory includes a constant-velocity segment. Unlike [16] we do not solve the problem for all 4 possibilities. Instead we first determine the signs of the accelerations a_1 and a_2 and of the velocity limit that we might potentially hit.

We can visualize the dependence of the sign of the accelerations and velocity limit on the start and goal states in the phase plane. Fig. 3 shows minimum- and maximum-acceleration trajectories emanating from a given start state. If the goal state is in regions I/II/III, a_1 has the same sign as the start velocity. In regions IV/V it has the opposite sign.

To determine the sign programmatically, we compare the distance traveled Δp_{acc} while accelerating as fast as possible from the start velocity v_1 to the goal velocity v_2 with the actual distance between the start position p_1 and goal position p_2.

$$\Delta p_{\text{acc}} = \frac{1}{2}(v_1 + v_2)\frac{|v_2 - v_1|}{a_{\max}} \tag{9}$$

$$\sigma = \mathrm{sgn}(p_2 - p_1 - \Delta p_{\text{acc}}) \tag{10}$$

$$a_1 = -a_2 = \sigma a_{\max} \tag{11}$$

$$v_{\text{limit}} = \sigma v_{\max} \tag{12}$$

We first find a solution without a constant-velocity segment. We solve the following quadratic equation for the duration of the first segment t_{a1}.

$$a_1 t_{a1}^2 + 2v_1 t_{a1} + \frac{v_2^2 - v_1^2}{2a_2} - (p_2 - p_1) = 0 \tag{13}$$

This quadratic equation has two solutions, but only one of them is valid. t_{a1} must be positive. So does the duration of the second segment t_{a2} given as

$$t_{a2} = \frac{v_2 - v_1}{a_2} + t_{a1} \tag{14}$$

The requirement for t_{a2} to be positive can be transformed into a lower bound on t_{a1}. Thus, we have two lower bounds on t_{a1} and the valid solution of Eq. 13 is always the greater one of the two (see Sec. IV-A). The total time is $T = t_{a1} + t_{a2}$.

Whereas Eq. 13 is the same as in [16], the constraints on t_{a1} to find the valid one of the two solutions are given incorrectly in [16].

We check whether the solution satisfies the velocity limits by checking the extreme velocity at time t_{a1}. If the solution is valid, then it is the minimum-time one. Otherwise, the minimum-time solution has a constant-velocity segment and is given by

$$t_{a1} = \frac{v_{\text{limit}} - v_1}{a_1} \tag{15}$$

$$t_v = \frac{v_1^2 + v_2^2 - 2v_{\text{limit}}^2}{2v_{\text{limit}}a_1} + \frac{p_2 - p_1}{v_{\text{limit}}} \tag{16}$$

$$t_{a2} = \frac{v_2 - v_{\text{limit}}}{a_2} \tag{17}$$

C. Infeasible Time Interval

A DOF has an infeasible time interval if and only if the goal state is in region I of Fig. 3. Fig. 4 visualizes the reason for the existense of the infeasible interval. It shows the trajectories that define the minimum time as well as the bounds of the infeasible interval. Reaching the goal state can be continuously delayed by choosing a lower-velocity trajectory anywhere between the solid and the dashed trajectory. However, this is only possible to an extent (until the dashed trajectory is reached) while still reaching the goal state directly. To further delay reaching the goal state, the DOF has to come to a complete stop and move backwards before accelerating towards the goal state. This requires additional time and thus gives rise to the infeasible time interval.

There is no such infeasible time interval for regions III - V, since the time-optimal trajectory to those regions has to cross through zero velocity. Thus the arrival time could be arbitrarily delayed by waiting at zero velocity. (Note, however, that this is not what we actually do.) There is no infeasible time interval for region II because the trajectory could be continuously slowed down to reach zero velocity. To calculate the infeasible time interval for region I, we switch the signs of a_1, a_2 and v_{limit}.

$$a_1 = -a_2 = -\sigma a_{\max} \tag{18}$$

$$v_{\text{limit}} = -\sigma v_{\max} \tag{19}$$

We solve Eq. 13 with a_1 and a_2 as defined in Eq. 18. The lower bound of the infeasible interval is given by the lesser solution to the quadratic equation and the upper bound is given by the greater solution (see Sec. IV-A). The trajectory representing the upper bound of the infeasible interval might violate velocity limits. If that is the case, the trajectory satisfying velocity limits is given by Eq. 15 - 17 with a_1, a_2 and v_{limit} as defined in Eq. 18 and 19.

D. Fixed-Time Trajectory

There is an infinite number of possible trajectories to reach a given goal state at a given feasible time. Previous work makes different choices about which trajectory to pick.

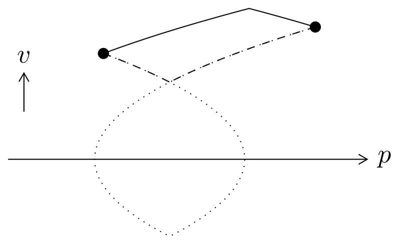
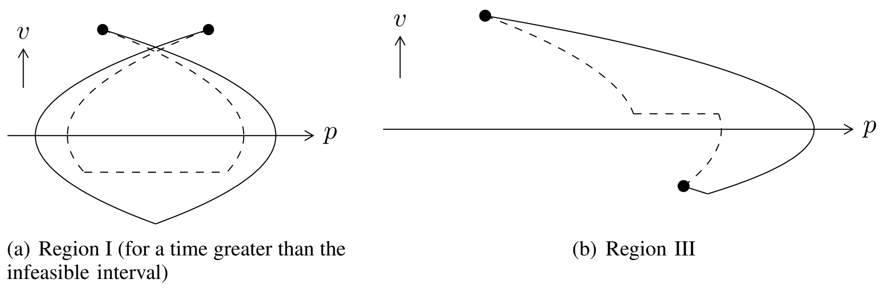


Fig. 4. Extremal-acceleration trajectories defining the minimum time (solid) and the lower (dashed) and upper (dotted) bounds of the infeasible interval.



(a) Region I (for a time greater than the infeasible interval)

(b) Region III

Fig. 5. Comparison of fixed-time trajectories as calculated by this paper and Hauser et al. [16] (solid) with previous work by Kröger [24] (dashed).

Kröger [24] chooses the trajectory to only consist of segments with either extremal or zero acceleration. However, this leads to unnecessarily high accelerations in some cases. Hauser et al. [16] in contrast choose the trajectory such that it minimizes the maximum absolute value of acceleration. However, unlike in [24] this might result in a trajectory that does not satisfy the joint limits even though it would be possible to satisfy them. We choose to follow Hauser et al. [16] and minimize the maximum absolute value of acceleration. Fig. 5 compares the fixed-time trajectories as calculated by us and Hauser et al. with the one calculated by Kröger. The rest of this section presents the algorithm for finding the minimum-acceleration trajectory, which almost exactly follows Hauser et al. [16]. The only small difference is that we are not evaluating both solutions to Eq. 20.

We find the minimum-acceleration trajectory by solving the following quadratic equation for a_1

$$T^2 a_1^2 + (2T(v_1+v_2) - 4(p_2-p_1))\, a_1 - (v_2-v_1)^2 = 0 \quad (20)$$

The solution with the greater absolute value gives the acceleration a_1 with $a_2 = -a_1$. The durations of the two trajectory segments are given by

$$t_{a1} = \frac{1}{2}\left(\frac{v_2 - v_1}{a_1} + T\right) \qquad t_{a2} = T - t_{a1} \quad (21)$$

If the trajectory violates velocity limits, we calculate the alternative solution with a constant-velocity segment at a velocity limit. The limit velocity is given by

$$v_{\text{limit}} = \operatorname{sgn}(a_1)\, v_{\max} \quad (22)$$

with a_1 as given by Eq. 20. The new accelerations satisfying the velocity limits are then given by

$$a_1 = -a_2 = \frac{(v_{\text{limit}} - v_1)^2 + (v_{\text{limit}} - v_2)^2}{2(v_{\text{limit}} T - (p_2 - p_1))} \quad (23)$$

The durations are given by Eq. 15 - 17 while using v_{limit}, a_1 and a_2 as defined in Eq. 22 and 23.

V. DIMT-RRT

Our planner combines the existing bidirectional RRT-Connect planner [1] with the steering method presented in Sec. IV. Our DIMT-RRT planner is shown in Algorithm 1.

We sample the state space consisting of joint positions and velocities (line 5). While sampling, we reject samples that cannot be part of a feasible solution, since it is unavoidable to hit a position limit before or after reaching them. This is the case for example if a DOF is moving with a high velocity towards a nearby position limit. This step is more important than it might seem. In our planning scenario 92% of samples get rejected.

We try to grow both trees towards the sample (lines 6 and 7). Different variations of the RRT-Connect algorithm exist. The two trees can either only make one step toward the sample (extend) or try to grow all the way to the sample (connect). We are using the latter method for both trees.

Algorithm 2 makes a connection attempt from a tree to the sample. Unlike in [1] our trajectories are not reversible. Thus, the parameter d keeps track of the direction of the tree growth. We are either growing the start tree forward in time or the goal tree backward in time.

We calculate the closest node in the tree using the steering method presented in Sec. IV (line 1). We use linear search to find the closest node. We cannot use data structures for efficient nearest-neighbor search in metric spaces since the DIMT distance function is not symmetric and thus not a metric. Only Sec. IV-B and IV-C, but not Sec. IV-D are used for the distance calculation.

Once the closest node is found, we calculate the trajectory

Algorithm 1: DIMT-RRT($x_{\text{init}}, X_{\text{goal}}, \Delta t$)

```
1  $V_1 \leftarrow \{x_{\text{init}}\}$; $E_1 \leftarrow \emptyset$;
2  $V_2 \leftarrow X_{\text{goal}}$; $E_2 \leftarrow \emptyset$;
3  $d =$ true;
4  while true do
5      $x_{\text{rand}} \leftarrow$ SampleReachableState();
6      if Connect($V_1, E_1, x_{\text{rand}}, d, \Delta t$) then
7          if Connect($V_2, E_2, x_{\text{rand}}, \neg d, \Delta t$) then
8              return ExtractTrajectory($V_1, E_1, V_2, E_2, d$);
9      Swap($(V_1, E_1), (V_2, E_2)$);
10     $d = \neg d$;
```

Algorithm 2: Connect($V, E, x_{\text{rand}}, d, \Delta t$)

```
1  $x_{\text{near}} \leftarrow$ NearestNeighbor($V, x_{\text{rand}}, d$);
2  $(T, \sigma) \leftarrow$ Steer($x_{\text{near}}, x_{\text{rand}}, d$);
3  if CollisionFree($T, \sigma$) then
4      $X_{\text{int}} \leftarrow$ IntermediateStates($T, \sigma, \Delta t$);
5      $V \leftarrow V \cup X_{\text{int}} \cup \{x_{\text{rand}}\}$;
6      $E \leftarrow E \cup \{x_{\text{near}}\} \times X_{\text{int}} \cup \{(x_{\text{near}}, x_{\text{rand}})\}$;
7      return true;
8  else
9      return false;
```

	RRT only	after 100 shortcuts	after 200 shortcuts
# samples	39.5 ± 41.3	-	-
# nodes	567.1 ± 407.0	-	-
Computation time	56 ms ± 54 ms	113 ms ± 63 ms	157 ms ± 65 ms
Trajectory length	12.4 s ± 4.0 s	6.4 s ± 1.1 s	6.1 s ± 1.1 s

TABLE I

RESULTS OF THE DIMT-RRT PLANNER FOR THE PROBLEM OF HITTING THE NAIL. AVERAGES OVER 100 RUNS ON A SINGLE CORE OF AN INTEL XEON E5-1620 CPU AT 3.6 GHZ WITH STANDARD DEVIATIONS.

σ as described in Sec. IV-D (line 2) and check it for collisions (line 3). If it is collision-free, we add the sample to the tree. The actual trajectory does not need to be stored. It can easily be reproduced, since the steering method is fast and deterministic. For improved performance, we not only add a node for the sample to the tree but also evenly spaced intermediate nodes along the trajectory (line 4).

VI. EXPERIMENTS

To evaluate our planner we apply it to the problem of hitting a nail with a hammer that is attached to a 7-DOF manipulator. We choose joint velocity and acceleration limits of $v_{\max} = \pi/2\ \mathrm{rad/s}$ and $a_{\max} = \pi/4\ \mathrm{rad/s}^2$. The hammer tip must not only reach the nail, it must also do so at a velocity parallel to the nail. We arbitrarily define this velocity to be 0.6 m/s. We automatically generate a set of joint states that satisfy these requirements. The details of this process are beyond the scope of this paper. Here we consider the goal states to be given in joint space. We focus on the problem of planning the trajectory until impact and ignore the problem of what to do when in contact with the nail.

To make the task harder we place an obstacle in the workspace of the robot. The obstacle is placed such that the robot can touch the nail with the hammer while reaching underneath the obstacle (Fig. 7). However, the obstacle makes it impossible to hit the nail at the required velocity while reaching underneath it. Instead, the planner must find a trajectory that reaches over the obstacle (Fig. 6). We check trajectories for collision discretely every 0.1 seconds.

A. Performance of the Bidirectional DIMT-RRT

The DIMT-RRT planner finds a trajectory that hits the nail in the desired direction and at the desired velocity. We smooth the trajectory using shortcutting as described in [16]. The shortened trajectory is shown in Fig. 6 and Fig. 1(a). The DIMT-RRT planner is not only able to find a trajectory that satisfies the task requirements, it also does so very quickly. Table I shows the computation time of the DIMT-RRT planner and the length of the generated trajectory with and without smoothing using shortcuts. In average the planner finds a feasible trajectory in only 103 ms. The table also shows that this is caused by a low number of samples (only 40 in average) required by the RRT. This shows that our

	DIMT-RRT [this paper]		Kinodynamic RRT [6]	
	extend	connect	extend	connect
# samples	9,019 ± 8,415	14.6 ± 10.8	> 1,000,000	
# nodes	9,633 ± 8,096	434.1 ± 188.4	> 900,000	> 2,500,000
Computation time	19.8 s ± 37.1 s	37 ms ± 22 ms	> 8 hours	> 23 hours

TABLE II

COMPARISON OF A SINGLE-TREE VARIANT OF THE DIMT-RRT AND THE STANDARD KINODYNAMIC RRT WITH A EUCLIDEAN DISTANCE METRIC.

steering method allows the RRT to solve the problem very efficiently without exploring much of the state space.

B. Comparison to Geometric RRT

The geometric planner uses the same set of goal states as the DIMT-RRT planner but ignores the velocity part. The geometric planner is able to reach the nail but does so from the side of the nail, which makes it impossible to drive the nail into the wall. The generated path is shown in Fig. 7 and 1(b). Since the geometric planner generated a path that reaches under the obstacle, the path cannot even be locally adapted to hit the nail at the desired speed.

C. Comparison to Standard Kinodynamic RRT

We compare the performance of our DIMT-RRT algorithm to a standard kinodynamic RRT [6] with a Euclidean distance metric. We use single tree variants of both algorithms that extend the tree by a fixed time step of 0.1 s. We alternate between sampling randomly and among the goal states. We consider two strategies for growing the tree. The *extend* strategy, which corresponds to the original kinodynamic RRT [6], makes exactly one step from the nearest node to the sample. The *connect* strategy repeatedly extends the tree towards the same sample until no more progress is made according to the used distance function.

There are only a few key differences between the two algorithms compared here. For the DIMT-RRT if the sample is less than a time step away, we connect the tree to the sample exactly. The algorithm terminates once the tree exactly connects to a goal state. For the standard kinodynamic RRT the tree is always grown by exactly one time step potentially overshooting the sample. Thus, for termination the algorithm only requires the Euclidean distance between a new node and the goal state the tree was growing toward to be less than 0.5. The DIMT-RRT rejects samples that cannot be reached without violating position limits, the standard kinodynamic RRT does not.

The standard kinodynamic RRT is not able to solve the problem at hand in a reasonable amount of time. We aborted the algorithm after 1,000,000 samples. The algorithm ran for more than 8 hours. Since we use linear search for determining the nearest neighbor, the algorithm could potentially be sped up. But this is not going to reduce the number of samples or get the computation time in the range of seconds. In contrast, as Fig. II shows, the variation of the DIMT-RRT is able to solve the problem efficiently. The connect strategy is significantly faster than the extend strategy.

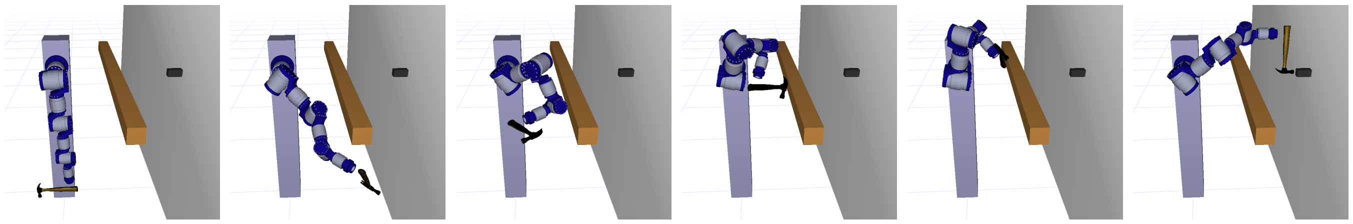


Fig. 6. The DIMT-RRT planner hits the nail at the desired velocity

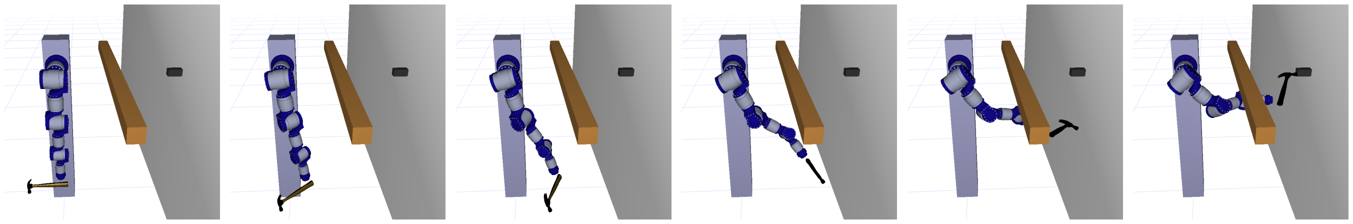


Fig. 7. The geometric RRT planner reaches the nail but not at the desired velocity

VII. Conclusion

We proposed the problem domain of acceleration-limited planning for manipulators. The presented probabilistically complete planner for this problem domain is highly efficient thanks to the use of a non-iterative steering method, which can solve the boundary value problem. We showed that this planner can solve problems a geometric planner cannot. Thus, for the presented problem our planner has advantages over both geometric and dynamic planners.

Some improvements to the DIMT-RRT planner are left for future work. The steering method could be changed to consider joint position limits. The nearest neighbor procedure could be improved to find the closest state within a trajectory. This would make adding intermediate nodes unnecessary.

We plan to apply the DIMT steering method to asymptotically optimal planners in the future, which require the ability to exactly connect states for rewiring.

Acknowledgments

This paper is dedicated to the memory of Mike Stilman. This work was supported in part by ONR grant N00014-14-1-0120.

References

[1] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE Int. Conf. on Robotics and Automation*, 2000.

[2] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The Int. Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

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

[4] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *IEEE Int. Conf. on Robotics and Automation*, 1999.

[5] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions 2000 WAFR*, 2000, pp. 293–308.

[6] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The Int. Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[7] T. Kunz and M. Stilman, "Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete," in *Int. Workshop on the Algorithmic Foundations of Robotics*, 2014.

[8] J. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The Int. Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985.

[9] K. Shin and N. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Trans. on Automatic Control*, vol. 30, no. 6, pp. 531–541, 1985.

[10] F. Pfeiffer and R. Johanni, "A concept for manipulator trajectory planning," *IEEE Journal of Robotics and Automation*, vol. 3, no. 2, pp. 115–123, 1987.

[11] J.-J. Slotine and H. Yang, "Improving the efficiency of time-optimal path-following algorithms," *IEEE Trans. on Robotics and Automation*, vol. 5, no. 1, pp. 118–124, 1989.

[12] Z. Shiller and H. Lu, "Computation of path constrained time optimal motions with dynamic singularities," *Journal of dynamic systems, measurement, and control*, vol. 114, p. 34, 1992.

[13] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," in *Robotics: Science and Systems*, 2012.

[14] Q.-C. Pham, "Characterizing and addressing dynamic singularities in the time-optimal path parameterization algorithm," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2013.

[15] K. Hauser, "Fast interpolation and time-optimization on implicit contact submanifolds," in *Robotics: Science and Systems*, 2013.

[16] K. Hauser and V. Ng-Thow-Hing, "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts," in *IEEE Int. Conf. on Robotics and Automation*, 2010.

[17] E. Glassman and R. Tedrake, "A quadratic regulator-based heuristic for rapidly exploring state space," in *IEEE Int. Conf. on Robotics and Automation*, 2010.

[18] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *IEEE Int. Conf. on Robotics and Automation*, 2012.

[19] G. Goretkin, A. Perez, R. Platt, and G. Konidaris, "Optimal sampling-based planning for linear-quadratic kinodynamic systems," in *IEEE Int. Conf. on Robotics and Automation*, 2013.

[20] D. J. Webb and J. van den Berg, "Kinomdynamic rrt*: Optimal motion planning for systems with linear differential constraints," in *IEEE Int. Conf. on Robotics and Automation*, 2013.

[21] N. Ratliff, M. Zucker, J. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE Int. Conf. on Robotics and Automation*, 2009.

[22] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *IEEE Int. Conf. on Robotics and Automation*, 2011.

[23] T. Kroger, A. Tomiczek, and F. Wahl, "Towards on-line trajectory computation," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006.

[24] T. Kröger, *On-line trajectory generation in robotic systems*. Springer, 2010.

[25] T. Kroger and F. Wahl, "Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94–111, 2010.