TIME-OPTIMAL SAMPLING-BASED MOTION PLANNING FOR MANIPULATORS WITH ACCELERATION LIMITS

A Dissertation
Presented to
The Academic Faculty

by

Tobias Kunz

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Robotics

School of Interactive Computing
Georgia Institute of Technology
May 2015

TIME-OPTIMAL SAMPLING-BASED MOTION PLANNING FOR MANIPULATORS WITH ACCELERATION LIMITS

In memory of my advisor Mike Stilman.

ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisor Mike Stilman. He was the one who convinced me to come back to Georgia Tech for my PhD. He always believed in me and my at first vague ideas on motion planning with differential constraints. Even at times when I was stuck and my ideas did not yield the expected outcome, he encouraged me to stay the course. He taught me how to present my research in papers and presentations. I am deeply saddened he never saw the outcome of all the work and will never read this thesis.

I am thankful for the outpouring of support from everybody in the school after Mike's passing. Especially, I would like to thank Annie Anton, Aaron Bobick and Frank Dellaert for supporting all of us during this difficult time. I would like to thank my second advisor Henrik Christensen for taking me in, supporting me and giving me access to his robots. Thank you to Sasha Lambert, Andrew Price and Thomas Andersen for helping me get my code running on those robots.

I am thankful to have had the opportunity to work with many faculty members across different departments at Georgia Tech, many of whom are also members of my reading committee. I would like to thank Magnus Egerstedt for always encouraging crazy research projects and being such a good teacher; Karen Liu for developing the core functionality of DART and allowing our lab to collaborate; Andrea Thomaz for continuing the constraint learning project without Mike and enabling me to finish my thesis work; Panagiotis Tsiotras for supporting me in my attempts at finding better distance metrics for systems with nonlinear dynamics. While these attempts where unsuccessful, they motivated much of the work presented in this thesis.

I would like to thank Steve LaValle for agreeing to serve on my committee as external member. This reminds me of the beginnings of my work on motion planning. I first learned about sampling-based planning from his book while working on my Diplom thesis six years ago.

I am grateful to the great friends that made my life more balanced and made Atlanta my home. Kate Bennett, Alekhya Narravula, Pierre Le Bodic, Michael Bonner, Isabella Karlsson, Melissa Giaimo, Laurissa Prystaj, Matthew Dutton, June Park, Daniela Steidl and everybody else I forgot, thank you for being you.

Thank you to my girlfriend Jen Laws, who always provided support and encouragement, especially when I was pressed for time before deadlines.

Thank you to my parents, who always supported me throughout my education, even though I decided to move across the ocean from them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Robot actuators have physical limitations in how fast they can change their velocity. The more accurately planning algorithms consider these limitations, the better the robot is able to perform.

Sampling-based algorithms have been successful in geometric domains, which ignore actuator limitations. They are simple, parameter-free, probabilistically complete and fast. Even though some algorithms like RRTs were specifically designed for kinodynamic problems, which take actuator limitations into account, they are less efficient in these domains or are, as we show, not probabilistically complete.

A common approach to this problem is to decompose it, first planning a geometric path and then time-parameterizing it such that actuator constraints are satisfied. We improve the reliability of the latter step. However, the decomposition approach can neither deal with non-zero start or goal velocities nor provides an optimal solution.

We demonstrate that sampling-based algorithms can be extended to consider actuator limitations in the form of acceleration limits while retaining the same advantageous properties as in geometric domains. We present an asymptotically optimal planner by combining a steering method with the RRT* algorithm. In addition, we present hierarchical rejection sampling to improve the efficiency of informed kinodynamic planning in high-dimensional spaces.

CHAPTER I

INTRODUCTION

1.1 Motivation

Robot actuators have physical limitations in how fast they can change their velocity. The more accurately planning algorithms consider these limitations, the better the robot is able to perform or the weaker (and thus cheaper) the actuators can be designed to achieve the same level of performance. The physical limitations can be considered by the planning algorithm at different levels of detail: The planner can ignore them altogether or consider (with increasing level of detail) acceleration limits, torque limits, current limits or temperature limits.

When ignoring physical limitations of the actuators and only planning geometrically, sampling-based planners like rapidly-exploring random trees (RRTs) [30] have been quite successful. They are simple, parameter-free, probabilistically complete and can quickly find solutions in high-dimensional configuration spaces like that of a 7-DOF manipulator in practice. Asymptotically optimal algorithms have been introduced recently as well. Kinodynamic sampling-based planners for problems with arbitrary differential constraints like torque limits, however, do not offer all of the advantageous properties. They are slower and, as we show, not even probabilistically complete in general.

A common approach to deal with the physical limitations of a manipulator's actuators is to decompose the problem and to first plan a geometric path, smooth it and then find a time parameterization of the path that satisfies the actuator limits. While we present some improvements to make the existing algorithm for the last step more reliable, decomposing the problem into multiple steps comes with some drawbacks:

First, a path returned by a geometric planner is only guaranteed to be executable by the robot if the start and goal states are at rest. Second, even if they are at rest and even if we find the shortest path and the optimal time parametrization for it, the resulting trajectory is not time-optimal. To overcome these drawbacks we must combine collision avoidance and satisfying actuator limitations in one planning step.

We propose acceleration-limited planning as a middle ground between geometric planning and general kinodynamic planning. While acceleration-limited planning cannot consider arbitrary differential constraints like torque limits, unlike geometric planning, it takes actuator limitations into account on an abstract level in form of acceleration limits. At the same time, we show that acceleration-limited planning can be solved while retaining the advantageous properties of geometric planners. In particular, we present a probabilistically complete planner based on an RRT that has greater computational efficiency than existing algorithms for general kinodynamic planning. In addition, unlike general kinodynamic planning, acceleration-limited planning is well suited for asymptotically optimal planners like the RRT*. We improve the efficiency of asymptotically optimal kinodynamic planners in high-dimensional spaces by introducing hierarchical rejection sampling and demonstrate that they can efficiently solve acceleration-limited planning problems in high-dimensional spaces, like the 14-dimensional phase space of a 7-DOF robot arm.

Probabilistic completeness and asymptotic optimality properties of the presented algorithms refer to the approximated system dynamics with acceleration limits and not the original system dynamics with torque or other limits. With respect to the original system, the trajectories planned by our algorithms are not optimal. In addition, with respect to the original system, our algorithms are not complete, not correct or both, depending on whether the acceleration limits are chosen pessimistically, optimistically or somewhere in between. This is not surprising and a normal consequence of approximating a system. Even an optimal trajectory for a system with torque limits is not optimal for the original system, because torque limits are only an approximation of temperature limits of the motors. The better the approximation, the closer an optimal trajectory for the approximated system will follow the optimal trajectory for the original system. Every planner or controller tries to find the right compromise between an approximation that closely matches the original system and that can be modeled and solved efficiently. We argue that considering acceleration limits is a good compromise between not considering any actuator constraints by planning geometrically and planning with full robot dynamics considering torque or current limits. Or to phrase it differently, we claim that our approximation is wrong (like every other one) but useful.

"Essentially, all models are wrong, but some are useful."

— George E. P. Box [5]

1.2 Thesis Statement

Acceleration-limited planning models actuator constraints more accurately than geometric planning and, unlike full kinodynamic planning, can be solved efficiently, without parameter tuning and in a probabilistically complete and asymptotically optimal manner.

1.3 Contributions and Overview

We make contributions in two areas. First, we improve an existing algorithm for time-optimally following a given path, which is part of a decomposed planning approach (Section 1.3.1). Second, we present an integrated sampling-based planning method that considers both obstacles and acceleration limits (Sections 1.3.2 - 1.3.5).

1.3.1 Time-Optimal Path Following

There has been a considerable amount of work during the 1980s on time-optimally following a given smooth geometric path considering the robot's torque limits [4, 51, 47, 53, 50]. This approach reduces the problem to one degree of freedom along the given path. For every point along the path a scalar acceleration along the given path needs to be chosen. This acceleration is almost always at one of its two limits, which are induced by the torque limits of the joints. The most critical part of the algorithm is the identification of the switching points at which to switch between maximum and minimum acceleration.

We show in Chapter 3 (published in [34]) that in order to make that approach work reliably some numerical issues need to be addressed. We also demonstrate that the algorithm can be simplified and more easily implemented robustly when only considering joint acceleration limits instead of joint torque limits. We also demonstrate that the acceleration at one type of switching points is given incorrectly in all previous work and empirically determine the correct acceleration in our case. Building on our work, Pham [48] derived the generally applicable acceleration at these switching points. The combination of our contributions leads to a more reliable and useful algorithm.

1.3.2 Incompleteness of Standard Kinodynamic RRTs

The RRT algorithm was originally introduced to deal with arbitrary kinodynamic constraints like acceleration or torque limits. Unlike the geometric RRT, the kinodynamic RRT does not rely on a steering method but instead samples control inputs for extending the tree towards a sampled state. A steering method is able to connect any two given states, i.e. solve the planning problem, while making one simplification: ignoring obstacles. In the geometric case the steering method is a straight-line segment between the two states. Not relying on a steering method makes the kinodynamic

RRT more generally applicable, since steering methods do not exist for all systems. However, not relying on a steering method makes the kinodynamic RRT also less efficient. Instead of making use of the steering method to extend the tree, the kinodynamic RRT samples multiple inputs and simulates the system forward for a small time step. Different variants of the kinodynamic RRT exist. They differ in how they choose the input and the time step. We show in Chapter 4 (published in [35]) that in contrast to geometric RRTs the most common variant of the kinodynamic RRT is not probabilistically complete.

1.3.3 Steering Method: Double-Integrator Minimum Time

In Chapter 5 (published in [36]) we present a steering method for acceleration-limited planning. This steering method is non-iterative and thus can be quickly calculated. Our work improves upon existing work [29, 28, 18] by being both detailed and correct.

1.3.4 Probabilistically Complete Planning

In Chapter 6 (also published in [36]) we combine above steering method with the RRT algorithm. This yields an algorithm that, unlike the kinodynamic RRT, is both probabilistically complete and computationally efficient. This approach is probabilistically complete even if the start or goal states are not at rest, overcoming the first drawback of the decomposed planning approach mentioned above.

1.3.5 Asymptotically Optimal Planning

To overcome the second drawback of the decomposed planning approach, the resulting trajectory not being optimal, we combine our steering method with an asymptotically optimal RRT* planner [25] in Chapter 7. Unlike general kinodynamic planning, acceleration-limited planning is well-suited for being solved by an asymptotically optimal sampling-based algorithm like the RRT*, since the steering method allows for exactly and efficiently connecting two states, which is crucial for the rewiring step of those algorithms.

In order to achieve a near-optimal solution, asymptotically optimal sampling-based planners need to densely fill the state space with samples. However, in high-dimensional spaces this is very inefficient. In order to improve the efficiency and not fill the whole state space with samples, additional information can be used to fill only those parts of the state space that can improve the current solution. Following [12], we call this the *informed subset* of the state space. Assume we can calculate a lower bound on the optimal cost to move from the start through a sample and to the goal. If a steering method is available, this bound can be calculated as the optimal cost ignoring obstacles. If this lower bound is larger than the current solution, the sample cannot improve the current solution, is not part of the informed subset and can be ignored.

For systems without differential constraints and the Euclidean distance as cost function, the informed subset is ellipsoidal [10]. This ellipsoidal subset can be parameterized and sampled directly [12].

For systems with differential constraints, e.g. acceleration limits, however, the informed subset is not ellipsoidal but more complex. No methods to sample directly within that space exist. Instead, samples within the informed subset can be found using rejection sampling, i.e. sampling the full state space and rejecting those samples that cannot improve the current solution. However, especially in high-dimensional spaces, the informed subset might only be a tiny fraction of the whole state space. In this case most samples get rejected and rejection sampling is very slow. In our experiments up to 99.99% of all samples get rejected. This can lead to a sampling-based planner spending most of its time rejection sampling.

To improve the efficiency of informed sampling without the need to explicitly parameterize the informed subset, we introduce *hierarchical rejection sampling (HRS)*. HRS hierarchically combines partial samples into larger samples, making accept/reject decisions at every level. When a partial sample is rejected, only the partial information needs to be resampled. This leads to an efficiency improvement that is exponential in the number of dimensions. In our experiments HRS speeds up the convergence of the RRT* planner for a 7-DOF manipulator with acceleration limits by up to two orders of magnitude.

1.4 Problem Definition

Given a start state $\boldsymbol{x}_{\text{init}}$ and a set of goal states $\mathcal{X}_{\text{goal}}$, we want to find a trajectory that connects the start state with any of the goal states. The trajectory must be 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 $\boldsymbol{a}$

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

$$\dot{\boldsymbol{v}} = \boldsymbol{a} \tag{2}$$

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

$$\boldsymbol{p}_{\text{min}} \leq \boldsymbol{p} \leq \boldsymbol{p}_{\text{max}} \tag{3}$$

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

$$-\boldsymbol{a}_{\text{max}} \leq \boldsymbol{a} \leq \boldsymbol{a}_{\text{max}} \tag{5}$$

$$\boldsymbol{p} \in \mathcal{C}_{\text{free}} \tag{6}$$

The algorithm for time-optimal path following presented in Chapter 3 in combination with a geometric planner can solve this problem as long as the start state and goal states are at rest.

The steering method presented in Chapter 5 solves the relaxed problem that ignores Eq. 3 and 6. 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).*

The probabilistically complete RRT planner presented in Chapter 6 can solve this problem independent of whether the start and goal states are at rest. The asymptotically optimal RRT* planner presented in Chapter 7 solves this problem time-optimally.

CHAPTER II

RELATED WORK

2.1 Adaption of a Geometric Path

One approach of dealing with obstacles and actuator limitations is to decompose the planning problem into two stages, first planning a geometric path avoiding obstacles 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. The methods presented in this section all rely on a geometric path being given and adapt it such that it can be executed on the robot. However, this is only guaranteed to yield a valid trajectory if the start and goal states are at rest. Also, it does not yield an overall optimal trajectory even if the geometric path was optimal and that path is followed optimally.

2.1.1 Parabolic Blends

Some standard textbooks [7, 52] describe a method to generate a smooth trajectory from waypoints such that acceleration and velocity limits are satisfied using linear segments with parabolic blends. However, the approach is not directly applicable to automatically generated paths with potentially dense waypoints. They assume that the timing between the waypoints and thus the velocities for the linear segments are already known. However, a path normally does not include timing for the waypoints. Choosing the timing is not trivial. Generally we prefer to move fast, but if we move too fast, neighboring blend segments might overlap and render the trajectory invalid. Also, the blend segments do not follow the original path exactly and thus might be colliding with obstacles.

2.1.2 Time-Optimal Path Following

There has been much work on time-optimally following a given smooth path such that joint acceleration or force/torque limits are satisfied. When following a given path exactly, collisions do not need to be considered since the path is known to be collision free. However, the path needs to be smooth to be able to follow it exactly without stopping. This usually requires an intermediate step to turn a geometric path consisting of straight-line segments into a smooth path.

The problem consists of finding an optimal time-parameterization $s(t)$ of the path such that the joint velocity and acceleration/force/torque limits are satisfied. The limits on the joints can be converted into constraints on $\dot{s}(t)$ and $\ddot{s}(t)$. Thus, the problem can be reduced to find the optimal scalar time-parameterization $s(t)$ that satisfies the constraints on its derivatives. Sections 2.1.2.1 and 2.1.2.2 present two different methods of finding this optimal time-parameterization given the constraints on its derivatives.

2.1.2.1 Numerical Integration along Extremal-Acceleration Trajectories

The minimum-time trajectory must always be at a limit, either an acceleration limit or a velocity limit. Thus, the optimal trajectory can be found by numerically integrating forward and backward in time at the minimum or maximum acceleration. The critical part is to determine when to switch between minimum and maximum acceleration. The points at which the optimal trajectory switches from minimum to maximum acceleration are always at the velocity limit. The velocity limit defines a limit curve. Along this curve the algorithm searches for switching points. This general approach was introduced by Bobrow et al. [4] and by Shin and McKay [51]. This approach is able to deal with more general actuator constraints than we do. For example, it can limit torques in the presence of robot dynamics. Later work [57] also considered joint velocity limits.

The most critical part of the algorithm is finding the switching points. The original work searches for switching points solely numerically, which is less efficient and makes the algorithm more likely to fail, especially in the presence of discontinuities in the path curvature. The following papers improve upon that by analyzing the conditions for these switching points to be able to calculate them explicitly instead of searching for them numerically. Even with this improvement one case remains that requires numerical search. We will show that that case cannot happen when only considering acceleration limits instead of torque limits and following a piecewise planar path.

[47] notes that points where the limit curve is non-differentiable are potential switching points and gives a necessary condition for identifying them. It also states that there is more than one valid acceleration at these switching points.

[53] lists all possible cases for switching points and gives necessary conditions for some of them that can be used to explicitly calculate switching point candidates. However, they do not give sufficient conditions for switching points. Also, one of the cases still requires a numerical search. We will show that the case that requires a numerical search in [53] cannot happen in our case with acceleration limits instead of torque limits and a piecewise planar path. We only use a numerical search for switching points caused by joint velocity limits.

[50] deals with those switching points along the limit curve where the acceleration is not uniquely determined. However, we show in Chapter 3 that the acceleration they claim to be optimal at such a point is incorrect.

In Chapter 3 we deal with a special case of this approach, i.e. acceleration limits instead of more general torque limits, which leads to some simplifications. We also present some numerical issues that need to be considered for making the algorithm work robustly.

2.1.2.2 Convex Optimization

Hauser [19] chooses a different approach for finding the optimal time parameterization. They discretize the path and use convex optimization. This approach is slower and less accurate, but is motivated by avoiding numerical issues and might be easier to implement robustly.

2.1.3 Acceleration-Limited Shortcuts

Hauser and Ng-Thow-Hing [18] use limited-acceleration shortcuts to turn a path into a smooth trajectory. They use a steering method similar to the one presented in this thesis to generate the shortcuts. Shortcuts are checked for collision.

2.2 Sampling-Based Planning

2.2.1 Probabilistically Complete Planning

2.2.1.1 Geometric RRT

Sampling-based planners like RRTs [30] 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 also useful for improving a path through shortcutting, to connect the two trees in a bidirectional RRT planner, for the rewiring step of an RRT* [25] 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. While the path can be post-processed to satisfy acceleration limits as described in Section 2.1, this cannot solve problems involving non-zero start or goal states and does not return an optimal solution. We have used geometric planning in some of our previous work [32, 33] resulting in slow robot motion.

2.2.1.2 Kinodynamic RRT with Incremental Simulator

The kinodynamic RRT as presented in [38, 39, 40, 41] is able to deal with general differential constraints of the form $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ 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 Δt and control input $\boldsymbol{u}$. It also requires a distance function $\rho : \mathcal{X} \times \mathcal{X} \to [0, \infty)$, which establishes a concept of closeness between states. Most commonly the Euclidean distance is used.

Algorithm 1 shows the construction of a kinodynamic RRT. Lavalle and Kuffner introduced different variants of the kinodynamic RRT algorithm. All variants iteratively grow a tree from $\boldsymbol{x}_{\text{init}}$. In each iteration a state $\boldsymbol{x}_{\text{rand}}$ is sampled (line 4). Then the node x_{near} closest to the sampled state according to the provided distance function (line 5) is selected. This is visualized in Figure 1(a). The NewState function (line 6) extends the tree from the selected node by applying some control input $\boldsymbol{u} \in \mathcal{U}$ for some time step Δt. Variants of the RRT algorithm differ in how Δt and $\boldsymbol{u}$ are chosen.

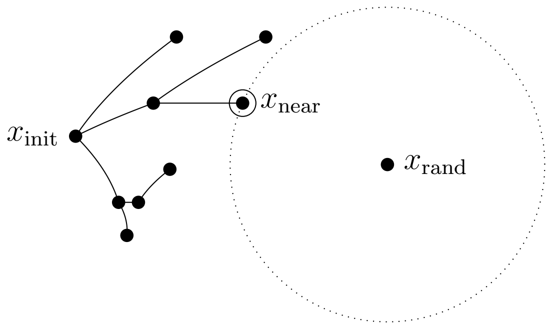
Early work on RRTs [38, 39] used a fixed time step Δt and chose the best input

```
Algorithm 1: BuildRRT($\boldsymbol{x}_{\text{init}}, \mathcal{X}_{\text{goal}}$)
1  $V \leftarrow \{\boldsymbol{x}_{\text{init}}\}$;
2  $E \leftarrow \emptyset$;
3  while $V \cap \mathcal{X}_{\text{goal}} = \emptyset$ do
4      $\boldsymbol{x}_{\text{rand}} \leftarrow$ SampleState();
5      $\boldsymbol{x}_{\text{near}} \leftarrow$ NearestNeighbor($V, \boldsymbol{x}_{\text{rand}}$);
6      $(\boldsymbol{x}_{\text{new}}, \boldsymbol{u}_{\text{new}}, \Delta t) \leftarrow$ NewState($\boldsymbol{x}_{\text{near}}, \boldsymbol{x}_{\text{rand}}$);
7      if CollisionFree($\boldsymbol{x}_{near}, \boldsymbol{x}_{new}, \boldsymbol{u}_{new}, \Delta t$) then
8          $V \leftarrow V \cup \{\boldsymbol{x}_{\text{new}}\}$;
9          $E \leftarrow E \cup \{(\boldsymbol{x}_{\text{near}}, \boldsymbol{x}_{\text{new}}, \boldsymbol{u}_{\text{new}}, \Delta t)\}$;
10 return $(V, E)$;
```

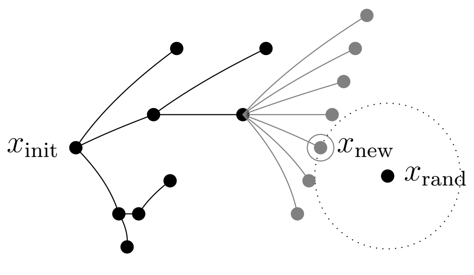
$\boldsymbol{u}$. Each input $\boldsymbol{u}$ is associated with a successor state, in which the system will end up when applying the input for a fixed time Δt from the current node. "Best input" refers to the input whose successor state is closest to the sampled state. This is visualized in Figure 1(b) and formalized in Algorithm 2.

If $\mathcal{U}$ is finite, the best input in line 1 of Algorithm 2 can be chosen by forward simulating all inputs and evaluating all resulting successor states. If $\mathcal{U}$ is continuous,

```
Algorithm 2: NewState($x_{\text{near}}, x_{\text{rand}}$)
(using fixed time step and best-input extension)
1  $\boldsymbol{u}_{\text{new}} \leftarrow \arg\min_{\boldsymbol{u} \in \mathcal{U}} \{\rho(\text{Simulate}(\boldsymbol{x}_{\text{near}}, \boldsymbol{u}, \Delta t), \boldsymbol{x}_{\text{rand}})\}$;
2  $\boldsymbol{x}_{\text{new}} \leftarrow$ Simulate($\boldsymbol{x}_{\text{near}}, \boldsymbol{u}_{\text{new}}, \Delta t$) ;
3  return $(\boldsymbol{x}_{\text{new}}, \boldsymbol{u}_{\text{new}}, \Delta t)$;
```



(a) Select nearest node



(b) Select best input

Figure 1: Visualization of best-input RRT variant. The shown system is a double integrator with $\dot{x}_1 = x_2$, $\dot{x}_2 = u$ and finite input set $\mathcal{U}$.

this is not possible. Instead, an analytical method must be used for an exact solution. However, often the best input is approximated instead by choosing the best one out of a finite number of sampled inputs.

Later, [40, 41] generalized the RRT algorithm and gave choices for the implementation of the NewState function. The time step Δt can either be fixed or variable and either the best or a random input $\boldsymbol{u}$ can be chosen. Algorithm 1 is general enough to allow all these variations. However, when the time step is fixed, the algorithm and data structures can be simplified by leaving out Δt.

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 hinders the efficient growth of the tree. Unlike a 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. Our experiments show that a kinodynamic RRT with a Euclidean distance function is not able to solve the problem efficiently. A weighted Euclidean distance function [40] might improve performance but requires tuning the weights to the specific problem. Our distance metric, in contrast, is parameter-free.

In addition, as we show in Chapter 4, the most common variant of the kinodynamic RRT, which grows the tree by a fixed time step using the best control input, is not probabilistically complete in general.

2.2.2 Asymptotically Optimal Planning

Karaman and Frazzoli [25] introduced asymptotically optimal sampling-based planning algorithms for systems without differential constraints. The algorithms they introduced include PRM*, RRG* and RRT*. Others have proposed RRT$^{\#}$[2], FMT*[20], BIT*[13]. All of these algorithms optimally connect nodes within a given distance of each other. This connection threshold shrinks as the number of nodes increases, avoiding large numbers of connection attempts and improving the efficiency of the algorithms.

Karaman and Frazzoli also extended their work to systems with differential constraints that have steering methods available [23]. Steering methods exist for example for the Dubins car, Reeds-Shepp car and double integrators, which we use.

Section 2.2.2.1 deals with the connection threshold used by all asymptotically optimal planners listed above. It provides an argument why we use an infinite connection threshold for our experiments. The RRT* algorithm, which we use for our experiments, is presented in detail in Section 2.2.2.2.

2.2.2.1 Connection Threshold

According to [25], to guarantee asymptotic optimality for systems without differential constraints, the connection threshold $r(n)$ must be chosen such that

$$r(n) = \left(\frac{\gamma}{\xi_d}\frac{\log(n)}{n}\right)^{\frac{1}{d}} \tag{7}$$

where n is the number of uniform samples in the state space X, d is the number of dimensions, ξ_d is the volume of a unit ball in a d-dimensional space and γ must satisfy

$$\gamma > 2\left(1+\frac{1}{d}\right)\mu(X) \tag{8}$$

where $\mu(X)$ is the volume of the state space.

A connection threshold directly at the lower bound usually gives good results and is therefore common practice. However, this does not mean that the lowest possible connection threshold guaranteeing asymptotic optimality necessarily also leads to the fastest convergence. How to choose the connection threshold for best convergence is an open problem.

The connection threshold above only guarantees asymptotic optimality for systems without differential constraints. Determining the connection threshold and proving asymptotic optimality is more involved for systems with differential constraints. This is because even two arbitrarily close states in the Euclidean space cannot necessarily be connected with arbitrarily small cost. Karaman and Frazzoli [23] present an RRT* variant explicitly for systems with differential constraints and derive a connection threshold for it.

For systems with differential constraints the connection threshold is a cost $l(n)$. Connections are attempted to all states that can be reached with a cost of at most $l(n)$. According to [23], to guarantee asymptotic optimality, $l(n)$ must be chosen such that the set of all states reachable with a cost of at most $l(n)$ includes a ball of radius $r(n)$ as defined in Eq. 7 with γ being "an appropriate constant". It is unclear what "appropriate" means or how to choose γ. In addition, it is not trivial to calculate $l(n)$ such that the space reachable with a cost of at most $l(n)$ contains a ball of a specific size. Karaman and Frazzoli [23] do not describe how they chose $l(n)$ for their example systems.

Work that uses the kinodynamic RRT* [23] often uses a connection threshold different from [23]. However, none of the work mentions the fact that the connection threshold is chosen differently or explains why. [45] and [17] choose $l(n) = r(n)$, which does not guarantee that the space reachable with a cost of at most $l(n)$ contains a ball of radius $r(n)$. They only state that γ is "constant" without any further explanation. So, it is unclear whether their algorithm is asymptotically optimal. [56] uses an infinite or constant finite connection threshold. Unlike [23], their threshold does not approach zero. While this guarantees asymptotic optimality, the paper does not provide a reason why they chose to differ from [23].

Like [56] we choose an infinite connection threshold in our experiments. We deviate from [23] because for systems with differential constraints it is unclear how to choose a finite and shrinking connection threshold such that asymptotic optimality is guaranteed.

2.2.2.2 RRT Algorithm*

The RRT* algorithm is shown in Algorithm 3. It iteratively builds a tree. Like the RRT it samples a state (line 3), finds the closest node (line 4) and grows the tree from it to the sample. Closeness is measured by the cost c of the trajectory returned by the steering method.

The standard RRT ends here and repeats the process. The RRT* continues by trying to find an alternative, lower-cost parent within the backward neighborhood of the sample (lines 6-12). The backward neighborhood of the sample consists of all nodes that can be connected to the sample with a trajectory of cost less than $l(n)$. $l(n)$ is the distance threshold explained in Section 2.2.2.1. In our experiments $l(n) = \infty$. The sample is then added to the tree as a new node with the parent resulting in the lowest cost.

The algorithm then checks if any of the nodes in the forward neighborhood of the new node can reduce their cost by choosing the new node as their new parent and rewires them if the cost can be reduced (lines 15-20).

Note that unlike [23] we are using two different neighborhoods for finding the parent and rewiring. Also note that unlike in the traditional RRT* algorithm we are not making a small step toward the sample, but instead grow the tree all the way to the sample. This is because making a small step results in the algorithm filling the space as the tree is growing outwards, which is inefficient in high-dimensional spaces.

2.2.3 LQR-RRT(*)

Recently, there have been efforts [14, 45, 17, 56] to use a more accurate distance metric by using LQR methods from optimal linear control. These algorithms assume linear system dynamics 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 [45, 17, 56]. The solution to the LQR problem gives an optimal trajectory.

Algorithm 3: RRT*

```
1  $V \leftarrow \{\boldsymbol{x}_{\text{init}}\}$; $E \leftarrow \emptyset$;
2  for $n = 1..N$ do
3      $\boldsymbol{x}_{\text{rand}} \leftarrow$ Sample();
4      $\boldsymbol{x}_{\text{nearest}} \leftarrow$ Nearest($V, \boldsymbol{x}_{\text{rand}}$);
5      if CollisionFree(Steer($\boldsymbol{x}_{\text{nearest}}, \boldsymbol{x}_{\text{rand}}$)) then
6          $X_{\text{near}} \leftarrow \{\boldsymbol{x} \in V : c(\text{Steer}(\boldsymbol{x}, \boldsymbol{x}_{\text{rand}})) \leq l(n)\}$;
7          $\boldsymbol{x}_{\text{min}} \leftarrow \boldsymbol{x}_{\text{nearest}}$;
8          $c_{\text{min}} \leftarrow \text{Cost}(\boldsymbol{x}_{\text{nearest}}) + c(\text{Steer}(\boldsymbol{x}_{\text{nearest}}, \boldsymbol{x}_{\text{rand}}))$;
9          foreach $\boldsymbol{x}_{\text{near}} \in X_{\text{near}}$ do
10             if $\text{Cost}(\boldsymbol{x}_{\text{near}}) + c(\text{Steer}(\boldsymbol{x}_{\text{near}}, \boldsymbol{x}_{\text{rand}})) < c_{\text{min}}$
               $\wedge$ CollisionFree(Steer($\boldsymbol{x}_{\text{near}}, \boldsymbol{x}_{\text{rand}}$)) then
11                 $\boldsymbol{x}_{\text{min}} \leftarrow \boldsymbol{x}_{\text{near}}$;
12                 $c_{\text{min}} \leftarrow \text{Cost}(\boldsymbol{x}_{\text{near}}) + c(\text{Steer}(\boldsymbol{x}_{\text{near}}, \boldsymbol{x}_{\text{rand}}))$;
13         $V \leftarrow V \cup \{\boldsymbol{x}_{\text{rand}}\}$;
14         $E \leftarrow E \cup \{(\boldsymbol{x}_{\text{min}}, \boldsymbol{x}_{\text{rand}})\}$;
15         $X_{\text{near}} \leftarrow \{\boldsymbol{x} \in V : c(\text{Steer}(\boldsymbol{x}_{\text{rand}}, \boldsymbol{x})) \leq l(n)\}$;
16         foreach $\boldsymbol{x}_{\text{near}} \in X_{\text{near}}$ do
17             if $\text{Cost}(\boldsymbol{x}_{\text{rand}}) + c(\text{Steer}(\boldsymbol{x}_{\text{rand}}, \boldsymbol{x}_{\text{near}})) < \text{Cost}(\boldsymbol{x}_{\text{near}})$
               $\wedge$ CollisionFree(Steer($\boldsymbol{x}_{\text{rand}}, \boldsymbol{x}_{\text{near}}$)) then
18                 $\boldsymbol{x}_{\text{parent}} \leftarrow$ Parent($\boldsymbol{x}_{\text{near}}$);
19                 $E \leftarrow E \setminus \{(\boldsymbol{x}_{\text{parent}}, \boldsymbol{x}_{\text{near}})\}$;
20                 $E \leftarrow E \cup \{(\boldsymbol{x}_{\text{rand}}, \boldsymbol{x}_{\text{near}})\}$;
21 return $(V, E)$;
```

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. Linearization also requires solving the LQR problem for each new sample, which requires numerical integration and is computational expensive. For linear systems like the acceleration-limited problem considered here LQR is able to consider the differential constraints exactly. However, the LQR problem can only deal with quadratic costs on the input and state, but not limits like our problem involves. Transforming acceleration limits into a quadratic cost requires parameter tuning. Our approach does not.

2.2.4 Adaptive Sampling

We present hierarchical rejection sampling to speed up asymptotically optimal planning with differential constraints in high-dimensions. Hierarchical rejection sampling is an efficient implementation of informed sampling described in Section 2.2.4.1. Other methods to adapt the distribution of samples and graph nodes have been proposed as well. Those are described in Sections 2.2.4.2 and 2.2.4.3.

2.2.4.1 Informed Sampling

We want to find the optimal trajectory from the start state $\boldsymbol{x}_{\text{start}}$ to the goal state $\boldsymbol{x}_{\text{goal}}$ according to some given cost function. The trajectory must completely lie in the obstacle-free space X_{free} and potentially satisfy differential constraints of the form $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ with $\boldsymbol{u} \in U$. Note that for a more concise presentation we assume a single goal state in this section. The approach can easily be extended to a finite set of goal states. In our experiments we actually have multiple goal states.

Let c_{best} be the cost of the current solution or infinite if no solution has been found yet. Informed sampling excludes samples that cannot improve c_{best}.

Let $c^*(\boldsymbol{x}_1, \boldsymbol{x}_2)$ be the cost of the optimal trajectory from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$. A sample can only improve the current solution if the cost of the optimal solution trajectory through $\boldsymbol{x}$ is less than c_{best}, i. e.

$$c^*(\boldsymbol{x}_{\text{start}}, \boldsymbol{x}) + c^*(\boldsymbol{x}, \boldsymbol{x}_{\text{goal}}) < c_{\text{best}} \tag{9}$$

$c^*(\boldsymbol{x}_1, \boldsymbol{x}_2)$ is generally unknown. Instead we use a heuristic estimate $c(\boldsymbol{x}_1, \boldsymbol{x}_2)$. c is called admissible if

$$\forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in X : c(\boldsymbol{x}_1, \boldsymbol{x}_2) \leq c^*(\boldsymbol{x}_1, \boldsymbol{x}_2) \tag{10}$$

Informed sampling uses such an admissible heuristic estimate to restrict samples to the subset of the state space that can potentially improve the current solution.

Following [12], we call this the *informed subset* of the state space. The set includes all states $\boldsymbol{x}$ satisfying

$$c(\boldsymbol{x}_{\text{start}}, \boldsymbol{x}) + c(\boldsymbol{x}, \boldsymbol{x}_{\text{goal}}) < c_{\text{best}} \tag{11}$$

Because of Eq. 10, Eq. 9 implies Eq. 11, i.e. the informed subset includes all states that can improve the current solution.

If a steering method is available, the cost of the trajectory returned by the steering method can be used as the heuristic cost estimate, i.e.

$$c(\boldsymbol{x}_1, \boldsymbol{x}_2) = c(\text{Steer}(\boldsymbol{x}_1, \boldsymbol{x}_2)) \tag{12}$$

In our experiments we make use of a steering method. The cost returned by the steering method is a lower bound on the optimal cost, since the steering method returns the optimal trajectory for the relaxed problem without obstacles.

2.2.4.2 Informed Graph Pruning

Informed graph pruning [24] is closely related to informed sampling. But instead of rejecting samples or areas of the state space, it rejects nodes in the graph. Thus, this cannot reject samples until they have been added to the graph. In contrast, informed sampling can reject samples before adding them to the graph. Informed graph pruning uses the graph as additional information. Instead of using a lower bound on the optimal cost-to-come, it uses the cost-to-come along the graph. This is advantageous because the graph is more informed as it considers obstacles. However, the cost along the graph gives an upper bound on the optimal cost-to-come instead of a lower bound. Thus, there is a chance that pruned nodes could have contributed to the optimal solution once the graph becomes closer to optimal.

The samples rejected by informed sampling are a subset of those rejected by informed graph pruning. Therefore, hierarchical rejection sampling can provide an improvement even in combination with graph pruning. We evaluate the performance gain of informed graph pruning in our experiments in Section 7.3.

2.2.4.3 Biased Sampling

There has been a lot of work on improving the efficiency of sampling-based planners through adaption of the sampling distribution. The distribution has been biased toward the current solution trajectory [1, 44] or areas of the state space with a high difficulty, because of low manipulability [43] or narrow passages [37, 55].

Unlike informed sampling, sample biasing still accepts samples that cannot improve the solution. These sample schemes might lead to better or worse performance. They include a parameter to control the strength of the bias, which affects performance and has to be tuned. Informed sampling in contrast is parameter-free. It might not always lead to better performance, but it never causes worse performance. If desired, some of these biased sampling schemes [1, 44, 43] can be combined with the work presented here to improve their performance. Note that special care has to be taken of the connection threshold when using biased sampling, because the standard threshold only guarantees asymptotic optimality for a uniform sampling distribution.

2.3 Trajectory Optimization

Trajectory optimizers like e.g. CHOMP [49] and STOMP [21] 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.

We used an optimization-based approach in one of our previous papers [54], but instead of optimizing a discretized trajectory, we optimized gains for a set of controllers and the switching times between them.

2.4 Double-Integrator Minimum Time

Some previous work also exploits the fact that the double-integrator minimum time (DIMT) problem can be solved efficiently. Kröger et al. [27, 29, 28] 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 [27] 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 [29, 28] 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 Chapter 5.

Hauser et al. [18] 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 [18] describes the algorithm in detail, parts of their algorithm and equations are incorrect, since they do not consider infeasible time intervals as will be explained in Chapter 5.

We have used a double-integrator system to generate motion for a robot arm in one of our previous papers [31]. In that work the arrival time was given and we only determined whether reaching the goal at the given time was possible. Also, there was no collision checking.

CHAPTER III

TIME-OPTIMAL PATH FOLLOWING

3.1 Overview

To deal with the complexity of planning a robot motion, the problem is often subdivided into two or more subproblems. The first subproblem is that of planning a geometric path through the environment, which does not collide with obstacles. In an additional step this path is converted into a time-parametrized trajectory that follows the path within the capabilities of the robot. Preferably we are looking for a near-optimal trajectory according to some optimality criterion. In practice the capabilities of the robot cannot be expressed exactly. Thus, the capabilities of the robot are normally approximated by using some model of the robot and limiting certain quantities. One option is to model the dynamics of the robot and limit the torques that can be applied by the joints. In this thesis we are using limits on joint accelerations and velocities which are often available.

This chapter presents a method to generate the time-optimal trajectory along a given path within given bounds on accelerations and velocities. The path can be given in any arbitrary configuration space. However, we assume that the acceleration and velocity of individual coordinates are limited. Thus, the configuration coordinates need to be chosen such that they match the quantities that need to be limited. For a robot manipulator the coordinates are typically chosen to match joints of the robot. Thus, we will refer to the limits as joint acceleration and joint velocity limits. We assume that the path is differentiable with respect to some path parameter s. This is a weak assumption. If the path was not differentiable at a point, the robot would have to come to a complete stop at that point in order to follow the path. Then the path could be split at that point and the method presented here could be applied to each part of the path. We also assume that the path curvature is piecewise-continuous and that for every piece the path is coplanar.

The output of a typical geometric path planner is a path in configuration space consisting of continuous straight line segments between waypoints. Such a path is not differentiable at the waypoints. Thus, we also present a preprocessing step to make such a path differentiable by adding circular blends.

3.2 Contribution

Our algorithms build on existing work presented in Section 2.1.2.1. The novel contributions of this work are the following:

- Combining path-following with a preprocessing step that converts the output of typical path planners to a differentiable path.
- A more thorough derivation of the constraints in the phase plane than [51]. At the same time our derivation is simpler since we only cover a special case of [51].
- We show that for the case of constraints on only acceleration and piecewise-planar paths, the limit curve is never continuous and differentiable at a switching point. Thus, we can calculate all switching points along the acceleration limit curve explicitly. Previous work relied at least partially on numerical search, but also handled the more general case of torque constraints.
- [50] applies an incorrect optimal acceleration at switching points where the limit curve is non-differentiable. We demonstrate this and provide an alternative solution.
- We provide sufficient conditions for switching points instead of just necessary ones.

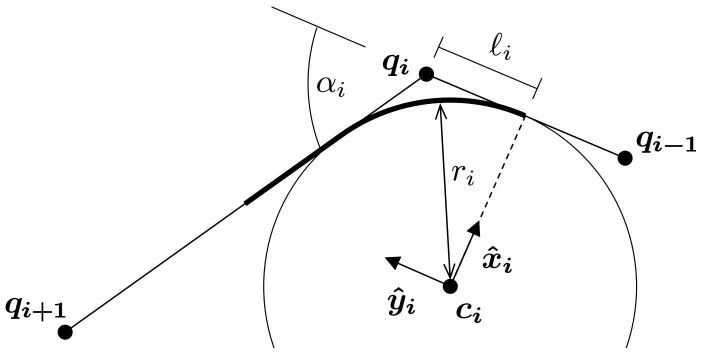


Figure 2: Circular blend around waypoint

- An improvement to the algorithm that makes it more robust in the presence of numerical inaccuracies.
- Our algorithm is demonstrated to be sufficiently robust to follow over 100 randomly generated paths without failing.
- We provide open-source software, available for download at http://www.golems.org/node/1570.

3.3 Path Preprocessing

Common geometric path planners like PRMs or RRTs usually output the resulting path as a list of waypoints, which are connected by straight lines in configuration space. At a waypoint the path is changing its direction instantaneously and thus is not differentiable. In order to follow such a path exactly, the robot would have to come to a complete stop at every waypoint. This would make the robot motion very slow and look unnatural. Therefore, we add circular blends around the waypoints, which make the path differentiable. If the path is already differentiable, the preprocessing described in this section can be omitted.

We are looking for a circular segment that starts tangential to the linear path segment before the waypoint and ends tangential to the linear path segment after the waypoint. We also need to ensure that the circular segment does not replace more than half of each of the neighboring linear segments. Otherwise we might not get a continuous path. In addition, we allow the enforcement of a maximum deviation from the original path.

First, we define some quantities that are helpful to define the circle (see also Figure 2). The unit vector $\mathbf{\hat{y}_i}$ pointing from waypoint $\mathbf{q_{i-1}}$ to $\mathbf{q_i}$ is given by

$$\boldsymbol{\hat{y}_i} = \frac{\boldsymbol{q_i} - \boldsymbol{q_{i-1}}}{|\boldsymbol{q_i} - \boldsymbol{q_{i-1}}|} \tag{13}$$

The angle α_i between the two adjoining path segments of waypoint q_i is given by

$$\alpha_i = \arccos\left(\boldsymbol{\hat{y}_i} \cdot \boldsymbol{\hat{y}_{i+1}}\right) \tag{14}$$

The distance ℓ_i between waypoint $\boldsymbol{q_i}$ and the points where the circle touches the linear segments is given as

$$\ell_i = \min\left\{\frac{||\boldsymbol{q_i} - \boldsymbol{q_{i-1}}||}{2}, \frac{||\boldsymbol{q_{i+1}} - \boldsymbol{q_i}||}{2}, \frac{\delta \sin\frac{\alpha_i}{2}}{1 - \cos\frac{\alpha_i}{2}}\right\} \tag{15}$$

where the first two elements give the maximum possible distances such that the circular segment does not replace more than half of the adjoining linear segments and the last element limits the radius to make sure the circlular segment stays within a distance δ from waypoint $\boldsymbol{q_i}$.

Given the quantities above, we can now define the circular segment. The circle is defined by its center $\boldsymbol{c_i}$, its radius r_i and two vectors $\boldsymbol{\hat{x}_i}$ and $\boldsymbol{\hat{y}_i}$ spanning the plane in which the circle lies. The vectors $\boldsymbol{\hat{x}_i}$ and $\boldsymbol{\hat{y}_i}$ are orthonormal. $\boldsymbol{\hat{x}_i}$ points from the center of the circle to the point where the circle touches the preceding linear path segment. $\boldsymbol{\hat{y}_i}$ is the previously defined direction of the preceding linear segment.

$$r_i = \frac{\ell_i}{\tan\frac{\alpha_i}{2}} \tag{16}$$

$$\boldsymbol{c_i} = \boldsymbol{q_i} + \frac{\boldsymbol{\hat{y}_{i+1}} - \boldsymbol{\hat{y}_i}}{||\boldsymbol{\hat{y}_{i+1}} - \boldsymbol{\hat{y}_i}||} \cdot \frac{r_i}{\cos\frac{\alpha_i}{2}} \tag{17}$$

$$\boldsymbol{\hat{x}_i} = \frac{\boldsymbol{q_i} - \ell_i\boldsymbol{\hat{y}_i} - \boldsymbol{c_i}}{||\boldsymbol{q_i} - \ell_i\boldsymbol{\hat{y}_i} - \boldsymbol{c_i}||} \tag{18}$$

Given these quantities specifying the circle, we can calculate the robot configuration $\mathbf{q}$ for any point on the circular segment as a function $\boldsymbol{f}(s)$ of the arc length s traveled from the start of the path. As we are currently only considering the current circular path segment, we assume $s_i \leq s \leq s_i + \alpha_i r_i$, where s_i specifies the start of the circular segment. Similarly we can calculate the first and second derivatives of the function $\boldsymbol{f}(s)$. Note that these are not time-derivatives but derivatives by s. We will make use of these derivatives later.

$$\boldsymbol{q} = \boldsymbol{f}(s) = \boldsymbol{c_i} + r_i \left(\boldsymbol{\hat{x}_i} \cos\left(\frac{s}{r_i}\right) + \boldsymbol{\hat{y}_i} \sin\left(\frac{s}{r_i}\right)\right) \tag{19}$$

$$\boldsymbol{f}'(s) = -\boldsymbol{\hat{x}_i} \sin\left(\frac{s}{r_i}\right) + \boldsymbol{\hat{y}_i} \cos\left(\frac{s}{r_i}\right) \tag{20}$$

$$\boldsymbol{f}''(s) = -\frac{1}{r_i} \left(\boldsymbol{\hat{x}_i} \sin\left(\frac{s}{r_i}\right) + \boldsymbol{\hat{y}_i} \cos\left(\frac{s}{r_i}\right)\right) \tag{21}$$

3.4 Reduction to One Dimension

The approach we are using was originally proposed in [51], which finds a minimum-time trajectory that satisfies torque limits on the joints. In constrast, we assume acceleration and velocity limits on the joints. Acceleration limits are a special case of torque limits. Using acceleration instead of torque limits results in a simpler problem and a simpler derivation of the following equations than in [51]. Unlike [51] and like [57], we are also considering velocity limits on the joints.

Because the solution is constrained to follow a given path exactly, the problem can be reduced to one dimension: choosing the velocity $\dot{s} = \frac{ds}{dt}$ for every position s along the path.

A path of length s_f is given as a function $f : [0, s_f] \rightarrow \mathbb{R}^n$. The configuration q at a point s along the path is given by

$$\boldsymbol{q} = \boldsymbol{f}(s) \quad 0 \leq s \leq s_f \tag{22}$$

where s can be an arbitrary parameter. We will assume it is the arc length traveled since the start of the path. We can also define the joint velocities and accelerations with respect to the parameter s.

$$\dot{\boldsymbol{q}} = \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{f}(s) = \frac{\mathrm{d}}{\mathrm{d}s}\boldsymbol{f}(s)\,\frac{ds}{dt} = \boldsymbol{f}'(s)\,\dot{s} \tag{23}$$

$$\ddot{\boldsymbol{q}} = \boldsymbol{f}'(s)\,\ddot{s} + \boldsymbol{f}''(s)\,\dot{s}^2 \tag{24}$$

If s is the arc length, $\dot{s}$ and $\ddot{s}$ are the velocity and acceleration along the path, then $\boldsymbol{f}'(s)$ is the unit vector tangent to the path and $\boldsymbol{f}''(s)$ is the curvature vector. [50]

The constraints in the high-dimensional joint space need to be converted into constraints on the scalar path velocity $\dot{s}(s)$ and path acceleration $\ddot{s}(s,\dot{s})$. The constraints on joint accelerations result in constraints on the acceleration and velocity along the path as shown in Section 3.4.1. The constraints on joint velocities result in constraints on the velocity along the path as presented in Section 3.4.2.

3.4.1 Joint Acceleration Limits

We have constraints on the joint accelerations given as

$$-\ddot{q}_i^{\max} \leq \ddot{q}_i \leq \ddot{q}_i^{\max} \qquad \forall i \in [0,...,n] \tag{25}$$

where $\ddot{q}_i$ is the ith component of vector $\ddot{\boldsymbol{q}}$. Although the universal quantifier is ommited in the following, all the following inequalities have to hold for all $i \in [0,...,n]$.

$$(25) \Leftrightarrow -\ddot{q}_i^{\max} \leq f_i'(s)\,\ddot{s} + f_i''(s)\,\dot{s}^2 \leq \ddot{q}_i^{\max} \tag{26}$$

If $f_i'(s) > 0$:

$$(26) \Leftrightarrow \frac{-\ddot{q}_i^{\max}}{f_i'(s)} - \frac{f_i''(s)\,\dot{s}^2}{f_i'(s)} \leq \ddot{s} \leq \frac{\ddot{q}_i^{\max}}{f_i'(s)} - \frac{f_i''(s)\,\dot{s}^2}{f_i'(s)} \tag{27}$$

$$\Leftrightarrow \frac{-\ddot{q}_i^{\max}}{|f_i'(s)|} - \frac{f_i''(s)\,\dot{s}^2}{f_i'(s)} \leq \ddot{s} \leq \frac{\ddot{q}_i^{\max}}{|f_i'(s)|} - \frac{f_i''(s)\,\dot{s}^2}{f_i'(s)} \tag{28}$$

If $f_i'(s) < 0$:

$$(26) \Leftrightarrow \frac{-\ddot{q}_i^{\max}}{f_i'(s)} - \frac{f_i''(s)\,\dot{s}^2}{f_i'(s)} \geq \ddot{s} \geq \frac{\ddot{q}_i^{\max}}{f_i'(s)} - \frac{f_i''(s)\,\dot{s}^2}{f_i'(s)} \tag{29}$$

$$\Leftrightarrow \frac{\ddot{q}_i^{\max}}{|f_i'(s)|} - \frac{f_i''(s)\,\dot{s}^2}{f_i'(s)} \geq \ddot{s} \geq \frac{-\ddot{q}_i^{\max}}{|f_i'(s)|} - \frac{f_i''(s)\,\dot{s}^2}{f_i'(s)} \tag{30}$$

If $f'_i(s) = 0$ and $f''_i(s) \neq 0$:

$$(26) \Leftrightarrow \frac{-\ddot{q}_i^{\max}}{|f''_i(s)|} \leq \dot{s}^2 \leq \frac{\ddot{q}_i^{\max}}{|f''_i(s)|} \tag{31}$$

$$\Leftrightarrow \dot{s} \leq \sqrt{\frac{\ddot{q}_i^{\max}}{|f''_i(s)|}} \tag{32}$$

If $f'_i(s) = 0$ and $f''_i(s) = 0$, Eq. 26 is always satisfied. Eq. 28 and Eq. 30 are equivalent. Thus, the limits on the path acceleration $\ddot{s}$ are

$$\ddot{s}^{\min}(s, \dot{s}) \leq \ddot{s} \leq \ddot{s}^{\max}(s, \dot{s}) \tag{33}$$

with

$$\ddot{s}^{\min}(s, \dot{s}) = \max_{\substack{i \in [1,\ldots,n] \\ f'_i(s) \neq 0}} \left(\frac{-\ddot{q}_i^{\max}}{|f'_i(s)|} - \frac{f''_i(s)\dot{s}^2}{f'_i(s)} \right) \tag{34}$$

$$\ddot{s}^{\max}(s, \dot{s}) = \min_{\substack{i \in [1,\ldots,n] \\ f'_i(s) \neq 0}} \left(\frac{\ddot{q}_i^{\max}}{|f'_i(s)|} - \frac{f''_i(s)\dot{s}^2}{f'_i(s)} \right) \tag{35}$$

The limit on the path acceleration also constrains the path velocity, because in order for a path velocity to be feasible we need $\ddot{s}^{\min}(s, \dot{s}) \leq \ddot{s}^{\max}(s, \dot{s})$.

We now derive the path velocity limit $\dot{s}^{\max}_{\text{acc}}(s)$ caused by acceleration constraints. We get rid of the min and max functions in Eq. 34 and 35 by requiring the inequality to hold for all possible combinations of arguments to the min and max functions.

$$\ddot{s}^{\min}(s, \dot{s}) \leq \ddot{s}^{\max}(s, \dot{s}) \tag{36}$$

$$\Leftrightarrow \left(\frac{\ddot{q}_i^{\max}}{|f'_i(s)|} - \frac{f''_i(s)\dot{s}^2}{f'_i(s)} \right) - \left(\frac{-\ddot{q}_j^{\max}}{|f'_j(s)|} - \frac{f''_j(s)\dot{s}^2}{f'_j(s)} \right) \geq 0$$
$$\forall i, j \in [1, ..., n], f'_i(s) \neq 0, f'_j(s) \neq 0 \tag{37}$$

$$\Leftrightarrow - \left(\frac{f''_i(s)}{f'_i(s)} - \frac{f''_j(s)}{f'_j(s)} \right) \dot{s}^2 + \left(\frac{\ddot{q}_i^{\max}}{|f'_i(s)|} + \frac{\ddot{q}_j^{\max}}{|f'_j(s)|} \right) \geq 0$$
$$\forall i, j \in [1, ..., n], f'_i(s) \neq 0, f'_j(s) \neq 0 \tag{38}$$

$$\Leftrightarrow -\left|\frac{f_i''(s)}{f_i'(s)} - \frac{f_j''(s)}{f_j'(s)}\right|\dot{s}^2 + \left(\frac{\ddot{q}_i^{\max}}{|f_i'(s)|} + \frac{\ddot{q}_j^{\max}}{|f_j'(s)|}\right) \geq 0$$

$$\forall i \in [1, ..., n], j \in [i+1, ..., n], f_i'(s) \neq 0, f_j'(s) \neq 0 \tag{39}$$

This gives a set of downward-facing parabolas in $\dot{s}$ horizontally centered around the origin. Each parabola is positive within an interval around 0, which is the interval the feasible velocities may lie in. The positive bound of the interval can be found by setting the parabola equation to zero.

$$-\left|\frac{f_i''(s)}{f_i'(s)} - \frac{f_j''(s)}{f_j'(s)}\right|\dot{s}^2 + \left(\frac{\ddot{q}_i^{\max}}{|f_i'(s)|} + \frac{\ddot{q}_j^{\max}}{|f_j'(s)|}\right) = 0 \tag{40}$$

$$\Leftrightarrow \dot{s} = \sqrt{\frac{\frac{\ddot{q}_i^{\max}}{|f_i'(s)|} + \frac{\ddot{q}_j^{\max}}{|f_j'(s)|}}{\left|\frac{f_i''(s)}{f_i'(s)} - \frac{f_j''(s)}{f_j'(s)}\right|}} \tag{41}$$

The interval of feasible path velocities $\dot{s}$ is defined by the intersection of the feasible intervals of all parabolas. Thus, the upper bound for the path velocity is the minimum of all upper bounds given in Eq. 41. Combining this with the case from Eq. 32, the constraint on the path velocity caused by joint acceleration limits is given by

$$\dot{s} \leq \dot{s}_{\text{acc}}^{\max}(s) \tag{42}$$

with

$$\dot{s}_{\text{acc}}^{\max}(s) = \min \left\{ \min_{\substack{i \in [1,...,n] \\ j \in [i+1,...,n] \\ f_i'(s) \neq 0 \\ f_j'(s) \neq 0 \\ \frac{f_i''(s)}{f_i'(s)} - \frac{f_j''(s)}{f_j'(s)} \neq 0}} \sqrt{\frac{\frac{\ddot{q}_i^{\max}}{|f_i'(s)|} + \frac{\ddot{q}_j^{\max}}{|f_j'(s)|}}{\left|\frac{f_i''(s)}{f_i'(s)} - \frac{f_j''(s)}{f_j'(s)}\right|}}, \min_{\substack{i \in [1,...,n] \\ f_i'(s) = 0 \\ f_i''(s) \neq 0}} \sqrt{\frac{\ddot{q}_i^{\max}}{|f_i''(s)|}} \right\} \tag{43}$$

3.4.2 Joint Velocity Limits

Constraints on the joint velocities are given as

$$-\dot{q}_i^{\max} \leq \dot{q}_i \leq q_i^{\max} \qquad \forall i \in [1, ..., n] \tag{44}$$

Plugging Eq. 23 into Eq. 44 yields

$$-\dot{q}_i^{\max} \leq f_i'(s)\dot{s} \leq q_i^{\max} \qquad \forall i \in [1, ..., n] \tag{45}$$

If $f_i'(s) = 0$, then Eq. 45 is always satisfied. Otherwise, because $\dot{s} > 0$, Eq. 45 is equivalent to

$$\dot{s} \leq \frac{q_i^{\max}}{|f_i'(s)|} \qquad \forall i \in [1, ..., n] \tag{46}$$

Thus, the constraint on the path velocity caused by limits on the joint velocities is given by

$$\dot{s} \leq \dot{s}_{\text{vel}}^{\max}(s) \tag{47}$$

with

$$\dot{s}_{\text{vel}}^{\max}(s) = \min_{\substack{i \in [1,...,n] \\ f_i'(s) \neq 0}} \frac{\dot{q}_i^{\max}}{|f_i'(s)|} \tag{48}$$

We will make use of the slope of this limit curve in the phase plane, which is the derivative by s given by

$$\frac{d}{ds}\dot{s}_{\text{vel}}^{\max}(s) = -\frac{\dot{q}_i^{\max} f_i''(s)}{f_i'(s)\,|f_i'(s)|} \qquad i = \arg\min_{\substack{i \in [1,...,n] \\ f_i'(s) \neq 0}} \frac{\dot{q}_i^{\max}}{|f_i'(s)|} \tag{49}$$

3.5 Algorithm

Figure 3 shows the s-$\dot{s}$ phase-plane. The start point is in the bottom-left corner and the end point in the bottom-right corner. The two limit curves limiting the path velocity, which are caused by joint acceleration and joint velocity constraints respectively, are shown. The trajectory must stay below these two limit curves. We want to find a trajectory that maximizes path velocity $\dot{s}$ at every point along the path. While not on the limit curve, the path acceleration must be at one of its limits, i.e. $\ddot{s}_{\min}$ or $\ddot{s}_{\max}$. Thus, at every point on the phase-plane below the limit curve there

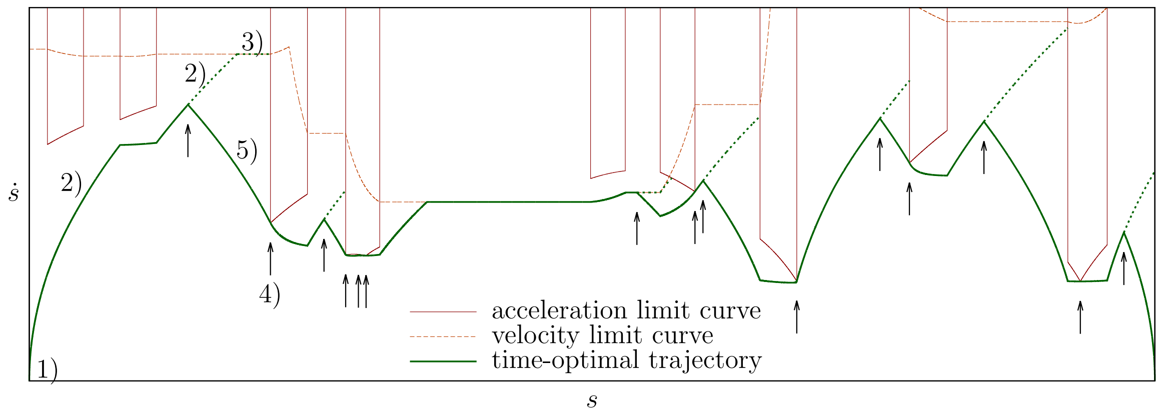


Figure 3: Phase-plane trajectory

are two possible directions to move in: one applying minimum acceleration and the other one applying maximum acceleration. The algorithm needs to find the switching points between minimum and maximum acceleration. In Figure 3 switching points are marked by arrows. Switching points from minimum to maximum acceleration must be on the limit curve, because otherwise we could find a faster trajectory above the solution trajectory [51].

The high-level algorithm is described below. It differs slightly from [51]. We integrate backward from the end point as the very last step. This makes the algorithm slightly simpler to implement, because while integrating forward we can never intersect with another trajectory part and while integrating backward we can never reach the limit curve. Section 3.7 gives some more implementation details. The numbers in Figure 3 correspond to the steps of the algorithm.

1. Start from the start of the path, i. e. $s = 0$ and $\dot{s} = 0$.

2. Integrate forward with maximum acceleration $\ddot{s} = \ddot{s}^{\max}(s, \dot{s})$ until one of the following conditions is met.

    - If $s \geq s_f$, continue from the end of the path, i. e. $s = s_f$ and $\dot{s} = 0$ and go to step 5.

    - If $\dot{s} > \dot{s}^{\max}_{\text{acc}}(s)$, go to step 4.

- If $\dot{s} > \dot{s}_{\text{vel}}^{\max}(s)$, go to step 3.

3. Follow the limit curve $\dot{s}_{\text{vel}}^{\max}(s)$ until one of the following conditions is met.

   - If $\frac{\ddot{s}^{\max}(s,\dot{s}_{\text{vel}}^{\max}(s))}{\dot{s}} < \frac{d}{ds}\dot{s}_{\text{vel}}^{\max}(s)$, go back to step 2.
   - If $\frac{\ddot{s}^{\min}(s,\dot{s}_{\text{vel}}^{\max}(s))}{\dot{s}} > \frac{d}{ds}\dot{s}_{\text{vel}}^{\max}(s)$, go to step 4.

4. Search along the combined limit curve for the next switching point. See Section 3.6.

   - Continue from the switching point and go to step 5.

5. Integrate backward with minimum acceleration until the start trajectory is hit. (If the limit curve is hit instead, the algorithm failed. We used this condition to detect failures shown in Table 3.) The point where the start trajectory is intersected is a switching point. Replace the part of the start trajectory after that switching point with the trajectory just generated.

   - If we transitioned into this step from step 2, halt. The start trajectory reached the end of the path with $s = s_f$ and $\dot{s} = 0$.
   - Otherwise, continue from the end of the start trajectory, which is the switching point found in step 4, and go to step 2.

3.6 Switching Points

With the exception of a finite number of points, at every point on the acceleration limit curve there is only a single feasible acceleration. If this acceleration leads into the feasible region, the point on the limit curve is called a trajectory source [47]. If this acceleration leads out of the feasible region, the point is called a trajectory sink. We define trajectory source and sink similarly for the velocity limit curve if all feasible accelerations lead into or out of the feasible region. If the interval of feasible accelerations allows following the velocity limit curve, we call the point singular.

A switching point along a limit curve is a point where the limit curve switches from being a trajectory sink to being a trajectory source or to being singular. *The* limit curve is the curve given by the minimum of the acceleration limit curve and the velocity limit curve. A switching point of the acceleration or velocity limit curve is a switching point of the limit curve if the limit curve the switching point is on is the lower one. The following two sections deal with finding the switching points on both, the acceleration and velocity limit curves.

3.6.1 Caused by Acceleration Constraints

This section describes how to find switching points along the path velocity limit curve $\dot{s}^{\max}_{\text{acc}}(s)$ caused by constraints on the joint accelerations.

We distinguish three cases of these switching points, depending on whether the curve is continuous and/or differentiable at the switching point.

3.6.1.1 Discontinuous

$\dot{s}^{\max}_{\text{acc}}(s)$ is discontinuous if and only if the path curvature $f''(s)$ is discontinuous [53]. For a path generated as described in Section 3.3 the discontinuities are exactly the points where the path switches between a circular segment and straigt line segment. If the path's curvature $f''(s)$ is continuous everywhere, this case of switching points does not happen.

At a discontinuity s there are two path velocity limits $\dot{s}^{\max}_{\text{acc}}(s^-)$ and $\dot{s}^{\max}_{\text{acc}}(s^+)$. The switching point is always at the smaller one of the two. If the discontinuity is a positive step and there is a trajectory sink in the negative direction of the discontinuity, then the discontinuity is a switching point. Equally, if the discontinuity is a negative step and there is a trajectory source in the positive direction of the discontinuity, then the discontinuity is a switching point. Or more formally, a discontinuity of $\dot{s}^{\max}_{\text{acc}}(s)$ is a

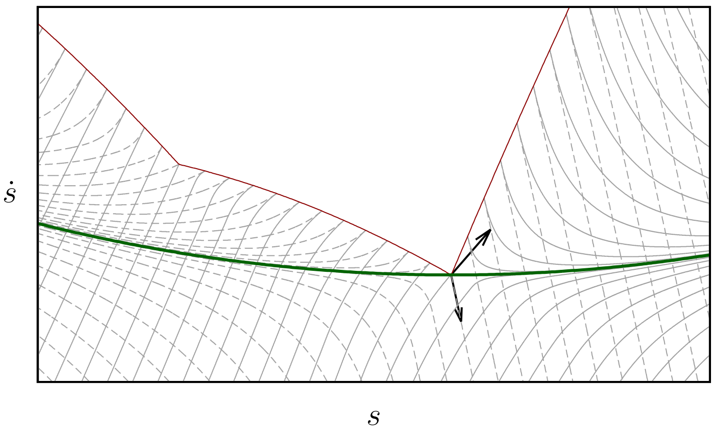


Figure 4: Minimum and maximum acceleration trajectories near a switching point where the limit curve is nondifferentiable

switching point if and only if

$$\Big(\dot{s}^{\max}_{\text{acc}}(s^-) < \dot{s}^{\max}_{\text{acc}}(s^+) \land \ \ddot{s}^{max}(s^-, \dot{s}^{\max}_{\text{acc}}(s^-)) \geq \frac{d}{ds}\dot{s}^{\max}_{\text{acc}}(s^-)\Big) \\ \lor \Big(\dot{s}^{\max}_{\text{acc}}(s^-) > \dot{s}^{\max}_{\text{acc}}(s^+) \land \ \ddot{s}^{max}(s^+, \dot{s}^{\max}_{\text{acc}}(s^+)) \leq \frac{d}{ds}\dot{s}^{\max}_{\text{acc}}(s^+)\Big) \quad (50)$$

3.6.1.2 Continuous and Nondifferentiable

The original paper [51] introducing the approach claims that every point along the limit curve only has a single allowable acceleration with $\ddot{s}^{\min}(s, \dot{s}^{\max}_{\text{acc}}(s)) = \ddot{s}^{\max}(s, \dot{s}^{\max}_{\text{acc}}(s))$. However, this is inaccurate. As [47] notes, there are points along the limit curve where there is an interval of allowable accelerations, i. e. $\ddot{s}^{\min}(s, \dot{s}^{\max}_{\text{acc}}(s)) < \ddot{s}^{\max}(s, \dot{s}^{\max}_{\text{acc}}(s))$. These points are called critical points in [50] and zero-inertia points in [53]. At these points the limit curve is continuous but nondifferentiable [50]. As noted in [47], a neccessary condition for such switching points is

$$\exists i : f_i'(s) = 0 \quad (51)$$

For a path generated from waypoints as described in Section 3.3, points satisfying Eq. 51 can be easily calculated. There are at most n per circular segment.

[47] does not note that choosing the correct acceleration at such a switching point might be an issue. [47] proposes to integrate backward from such a switching point

with minimum acceleration and integrate forward with maximum acceleration as usual. [50] notices that the acceleration at such a switching point needs special consideration. [50] notes that the maximum or minimum acceleration cannot always be followed, because they would lead into the infeasible region of the phase plane. [50] calls such a point a singular point and proposes to follow the limit curve tangentially. However, both [47] and [50] are in error.

Figure 4 shows a switching point at a point where the limit curve is nondifferentiable. At the switching point two arrows indicate the direction of motion in the phase plane when applying minimum or maximum acceleration, respectively. In the feasible region of the phase plane gray curves visualize two vector fields. They show minimum- and maximum-acceleration trajectories. Minimum-acceleration trajectories are dashed, maximum-acceleration trajectories are solid. In the case shown here, if integrating backwards with minimum acceleration, the infeasible region would be entered. Thus, according to [50] we would have to follow the red limit curve tangentially backwards from the switching point. For integrating forward [47] and [50] propose to follow the upward arrow. However, both of these motions are not possible, as they would not move along the vector field sourrounding the switching point. Zero is the only acceleration at the switching point that conforms with the vector field around it, resulting in a trajecory moving horizontally in the phase plane.

We claim that the optimal acceleration is zero at every such switching point. We leave the proof for future work, but all of our experimental data supports this claim.

In order for the zero acceleration to be feasible, the limit curve must switch from a negative to a positive slope at the switching point. We claim that the limit curve switching from a negative to a positve slope is a necessary and sufficient condition for a switching point at a point of nondifferentiability of the limit curve. Together with the sufficient condition stated in Eq. 51, this allows for an explicit enumeration of all such switching points.

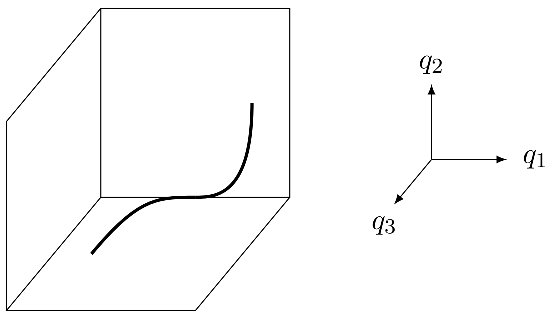


Figure 5: Joint acceleration space for n=3 with box constraints and a trajectory for a switching point that meets the limit curve tangentially.

3.6.1.3 Continuous and Differentiable

According to the following lemma this case does not exist.

Lemma 1. *If only joint accelerations are constrained, the path is piece-wise coplanar, and the curvature $f''(s)$ is discontinuous at the points where the pieces are stitched together, then there are no switching points at which the limit curve is continuous and differentiable.*

Proof. Before the switching point the path acceleration is at the lower limit. This implies that at least one joint is at its acceleration limit. After the switching point the path acceleration is at its upper limit. This implies that at least one joint distinct from the previous one is at its acceleration limit. At the switching point both joints are at their acceleration limit. The limit curve, the phase-plane trajectory and the joint-space trajectory are continuous and differentiable near the switching point. One joint approaches its acceleration limit at the switching point and then stays at the limit. Another distinct joint is at its acceleration limit and leaves the limit at the switching point. Figure 5 shows the joint acceleration space for a 3-joint system with the joint acceleration limits shown as a box. At the switching point two joint are at their acceleration limit. Thus, the trajectory is on an edge of the box at the switching point. Before the switching point the trajectory stays on one surface of the box, approaches the edge tangentially. After the switching point the trajectory leaves the edge tangentially while staying on another surface of the constraint box. This trajectory lives in at least a three-dimensional subspace of the joint acceleration space. There is no two-dimensional subspace that includes the trajectory in the vicinity of the switching point. Because the path is piecewise planar, the accelerations in joint space are also piecewise planar. Because there is no trajectory in joint acceleration space that cannot be included in a two-dimensional subspace, a switching point where the limit curve is differentiable does not exist. The only case where such a switching point can exist is where two planar parts of the path are stitched together. However, these points are those where the curvature $f''(s)$ is discontinuous and thus the limit curve is discontinuous. □

3.6.2 Caused by Velocity Constraints

This section describes how to find switching points along the path velocity limit curve $\dot{s}_{\text{vel}}^{\max}(s)$ caused by constraints on the joint velocities. We distinguish two cases of these switching points, depending on whether $\ddot{s}^{\min}(s, \dot{s}_{\text{vel}}^{\max}(s))$ is continuous.

3.6.2.1 Continuous

s is a possible switching point if, only if

$$\ddot{s}^{min}(s, \dot{s}_{\text{vel}}^{\max}(s)) = \frac{d}{ds}\dot{s}_{\text{vel}}^{\max}(s) \tag{52}$$

We search for these switching points numerically by stepping along the limit curve until we detect a sign change. Then we use bisection to more accurately determine the switching point. See also Section 3.7.2.

3.6.2.2 Discontinuous

$f_i''(s)$ being discontinuos is a necessary condition for $\ddot{s}^{\min}(s, \dot{s}_{\text{vel}}^{\max}(s))$ being discontinuos. s is a possible switching point if and only if

$$(\ddot{s}^{min}(s^-, s_{\text{vel}}^{\max}(s^-)) \geq \frac{d}{ds}\dot{s}_{\text{acc}}^{\max}(s^-)) \;\wedge\; (\ddot{s}^{min}(s^+, s_{\text{vel}}^{\max}(s^+)) \leq \frac{d}{ds}\dot{s}_{\text{acc}}^{\max}(s^+)) \tag{53}$$

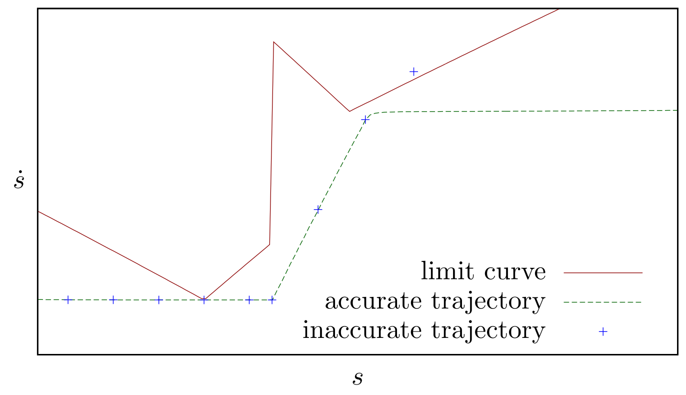
3.7 Numerical Considerations

When doing floating-point calculations we must accept approximate results. However, we cannot accept the algorithm failing completely due to numerical inaccuracies. We now describe how numerical inaccuracies can make the algorithm described in Section 3.5 fail and the measures we have taken to avoid that. None of the previous papers on this approach [51, 47, 53, 50, 57] have dealt with numerical issues.

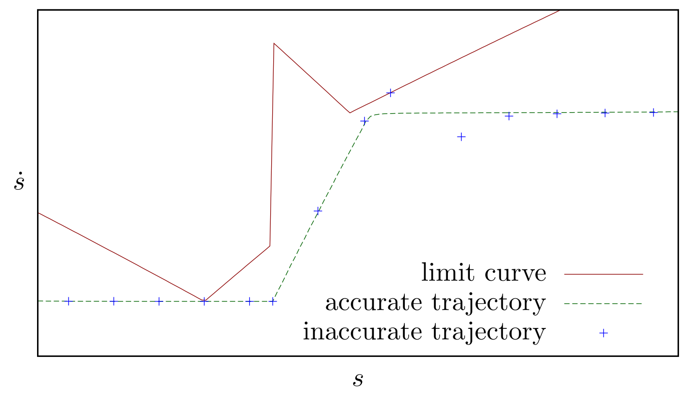
3.7.1 Integration

The algorithm described in Section 3.5 assumes that we can exactly integrate the trajectory. Under this assumption it is impossible to hit the limit curve at a trajectory source when integrating forward. However, as we have to do the integration numerically with some non-zero step size, it is not exact. This can lead to the limit curve being hit at a trajectory source. Figure 6 shows an example of the trajectory hitting the limit curve at a trajectory source. The figure shows the limit curve together with an accurate trajectory generated using a small step size and a not-so-accurate trajectory generated with a 10 times larger step size (1 ms).

The trajectories enter the figure on the left at minimum acceleration until they touch the limit curve. Then they switch to maximum acceleration. In the following the accurate trajectory gets close to the limit curve but does not hit it. The inaccurate trajectory, however, because of the larger step size, misses the bend shortly before the limit curve and hits the limit curve. According to the algorithm described in Section 3.5 and the algorithms described in previous work [51, 47, 50, 57] we would have to stop the forward integration, search for the next switching point along the limit curve and integrate backward from there until we hit the forward trajectory. However, when integrating backward from the next switching point we might hit the limit curve instead of the forward trajectory. This is because the algorithm relies on the fact that there are no trajectory sources between where the forward trajectory



(a) failure without source/sink check



(b) success with source/sink check

Figure 6: Dealing with integration inaccuracies

hits the limit curve and the next switching point, which is not the case if the forward trajectory hits the limit curve at a trajectory source. To deal with this issue, we determine the intersection point with the limit curve by bisection and then check whether this point is a trajectory source by comparing the slope of the limit curve with the slope of the maximum-acceleration trajectory. Figure 6 shows the result of this measure. The inaccurate trajectory does not follow the accurate one exactly, but it does not stop when the limit curve is hit. Thus the algorithm succeeds.

Note that although the smaller step size solved the problem in the example shown in Figure 6, it does not do so in general. In general, a smaller step size or a better integration method only make it less likely that the limit curve is hit at a trajectory source. Thus, the method described above is necessary to ensure completeness of the algorithm.

3.7.2 Switching Point Search

All previous work at least partially relies on numerical search to find switching points, but none notes the potential issues caused by using it. We also use numerical search, but only along the velocity limit curve. If we only had acceleration constraints, numerical search would not be necessary. We try to avoid numerical search as much as possible, because it is slower, less accurate and might make the algorithm fail. Our algorithm relies on the fact that we can find the next switching point along the limit curve. However, by searching numerically and stepping along the limit curve, we could theoretically miss a switching point and find one that is not the next one. The algorithm could potentially be adapted to work with any switching point without requiring it being the next one. However, during our experiments we did not encounter a failure caused by missing a switching point.

3.8 Experimental Results

We evaluated our approach by generating trajectories for a 7-DOF robot arm that is given the task of picking and placing a bottle from or onto a shelf or table. The results are averages over 100 pick-and-place operations. For each pick-and-place operation we randomly selected a pick and a place location. Figure 7 shows the 200 pick and

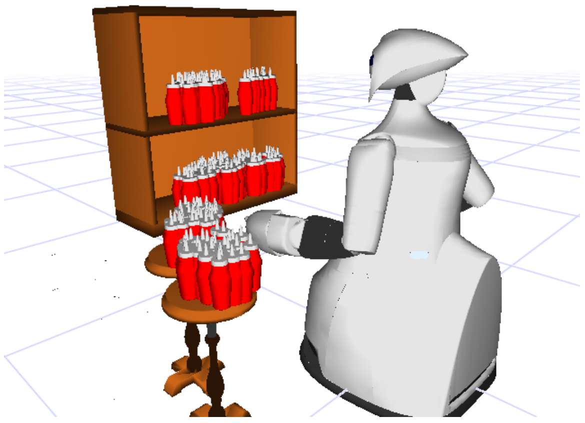


Figure 7: 200 start and goal locations

Table 1: Results for varying time steps

Time Step	Computation Time	Execution Time	Failure rate without source/sink check	Failure rate with source/sink check
10 ms	0.2 s	4.74 s	13 %	0 %
1 ms	0.9 s	4.71 s	3 %	0 %
0.1 ms	31.8 s	4.70 s	0 %	0 %

place locations. We used a bidirectional RRT to plan a configuration space path for the arm. Each pick-and-place operation constists of 3 path segments: from the initial arm configuration to the pick configuration, to the place configuration and back to the intial configuration. In our evaluation we ignore the finger motion necessary to grasp the object. The robot shown in Figure 7 uses only its left arm. The arm configuration shown is the inital arm configuration, which is reached at the beginning and end of every pick-and-place operation. The paths generated by the RRT planner are shortened using random short cutting. The result are paths given by waypoints at most 0.1 apart. Here we evaluate the trajectory generation from these waypoints. This method of generating example paths is very real-world oriented and does not allow us to handcraft waypoints until the trajectory generation accidentally succeeds.

Table 3 shows computation and execution times for different time steps used for the integration of the trajectory. The computation time was achieved on a single core of an Intel Core i5 at 2.67 Ghz. Smaller time steps lead to a longer computation time and a slightly closer to optimal path. What is not shown here is that a smaller step size also leads to the constraints being satisfied more accurately. Related to that is the fact that a larger time step makes the algorithm more likely to fail if the counter measures for numerical inaccuracies described in Section 3.7.1 are omitted. Our algorithm is robust enough to successfully generate trajectories for 100 random pick-and-place operations using different time steps.

We also ran our algorithm on a real-robot arm following a path defined by hand-crafted waypoints. A video of this and a few example pick-and-place operations is available at http://www.golems.org/node/1570.

3.9 Conclusion

We presented an algorithm to follow a path in time-optimal manner while satisfying joint acceleration and velocity constraints. We added blends to a list of waypoint in order to be able to use the output of standart path planners. Our improvements to existing work make the algorithm more robust and less likely to fail. We showed the robustness of our implementation by following paths planned by an RRT planner for 100 random pick-and-place operations.

CHAPTER IV

INCOMPLETENESS OF STANDARD KINODYNAMIC RRTS

4.1 Overview

Rapidly-Exploring Random Trees (RRTs) as introduced by LaValle and Kuffner [38, 39, 40, 30, 41] are a popular method for geometric and kinodynamic planning. Many, e.g. [11, 9, 16], consider RRTs to be a synonym for probabilistic completeness. However, this is not necessarily the case. Kinodynamic RRTs [38, 39, 40, 41] (see Section 2.2.1.2) only have the property of probabilistic completeness under a set of assumptions, which depend on implementation details that are left open by the RRT algorithm. These details govern how the time step and the input are chosen to extend the tree from the selected node. While it has been shown that the RRT algorithm for kinodynamic planning is probabilistically complete with a fixed time step and a random control input [40, 41], we now describe that a more commonly used variant is not probabilistically complete in the general case. This variant uses a fixed time step and chooses the best control input for the extension of the tree from the selected node. This variant is for example used in [6, 15, 46, 3, 22, 9].

Even though we prove this variant to not be probabilistically complete in general, it could potentially be made probabilistically complete by introducing additional requirements on the system dynamics and/or the used distance metric. In fact, one of the goals of this chapter is to spur further research on the exact conditions under which RRTs are probabilistically complete.

4.1.1 Problem Formulation

In this analysis, consider a system with differential constraints given as

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{54}$$

with state $\boldsymbol{x} \in \mathcal{X}$ and input $\boldsymbol{u} \in \mathcal{U}$.

The set of all collision-free states is given as $\mathcal{X}_{\text{free}} \subseteq \mathcal{X}$. An initial state $\boldsymbol{x}_{\text{init}} \in \mathcal{X}_{\text{free}}$ and a goal set $\mathcal{X}_{\text{goal}} \subseteq \mathcal{X}_{\text{free}}$ are given. We want to find a duration T and an input trajectory $\boldsymbol{u}(t)$ such that the differential constraints of Eq. 54 are satisfied for all $0 \leq t \leq T$, the trajectory is collision free with $x(t) \in \mathcal{X}_{\text{free}}$ for all $0 < t < T$, $\boldsymbol{x}(0) = \boldsymbol{x}_{\text{init}}$ and $\boldsymbol{x}(T) \in \mathcal{X}_{\text{goal}}$.

4.1.2 Probabilistic Completeness of Kinodynamic RRTs

An algorithm is probabilistically complete if the probability that an existing solution is found converges to 1 as the number of iterations grows to infinity [42].

It has been shown in [40, 41] that if $\mathcal{U}$ is finite, Δt is fixed and $\boldsymbol{u}$ is chosen at random, the RRT algorithm is probabilistically complete. However, choosing $\boldsymbol{u}$ randomly may not result in the RRT exploring the state space rapidly.

In contrast, the preliminary RRT variant introduced in [38, 39] also uses a fixed time step Δt but chooses the best input $\boldsymbol{u}$. The very first paper on RRTs [38] but none of the later papers [39, 40, 41] claimed this variant to be "probabilistically complete

Table 2: Probabilistic completeness of different kinodynamic RRT variants

	Fixed Δt	Variable Δt
Random $\boldsymbol{u}$	Probabilistically complete (if $\mathcal{U}$ finite) [40, 41]	?
Best $\boldsymbol{u}$	Not probabilistically complete [this thesis]	?

under very general conditions". We show that this variant is not probabilistically complete.

Restricting the RRT to a fixed time step renders the algorithm unable to find solutions that do not consist of Δt long segments of constant input. However, even if a solution with Δt long segments of constant input exists, the RRT with best-input extension might never find it.

This section up until here is summarized in Table 4.

As mentioned in Section 2.2.1.2, the best input out of a continuous input set is often approximated in practice by sampling a finite set of inputs and choosing the best input out of the finite set. This approximation may render the RRT algorithm probabilistically complete because of the added randomness. However, an algorithm that is probabilistically complete only thanks to approximation errors is likely to not be very efficient.

4.1.3 RRTs Using Steering Methods

A steering method is able to exactly connect any two states $x_1, x_2 \in \mathcal{X}$ with $\|x_1 - x_2\| < \epsilon$ for some $\epsilon > 0$ while ignoring obstacles. Computationally efficient steering methods are not available for general dynamical systems. They are only available for a few simple systems, e.g. Dubin's car [42, 8] and a set of double integrators [36]. A steering method in combination with a collision checker yields what is called a local planner in the probabilistic roadmap literature [26].

To be generally applicable, kinodynamic RRTs as introduced in [38, 39, 40, 41] do not require a steering method. Instead, they only rely on an incremental simulator that can simulate the system forward for a given input and time step. However, there are RRT algorithms that make use of a steering method. These are not the topic of this chapter. However, we want to briefly mention them in this section to make the differences clear and to emphasize that the negative result on probabilistic completeness presented in this chapter does not apply to those.

A steering method usually returns a trajectory that minimizes some cost, e. g. time. When using a steering method, the distance function used by the RRT is also based on this steering method by defining the distance as the optimal cost to move between two states ignoring obstacles. Karaman and Frazzoli [23] proved that an RRT* using an optimal steering method in combination with a distance function based on that steering method is probabilistically complete. Since an RRT* uses the same vertices as an RRT, the RRT algorithm is also probabilistically complete under these assumptions.

Geometric RRT planners [30] (see Section 2.2.1.1) that use a Euclidean distance function and connect configurations with a straight line in configuration space can also be viewed as using a steering method and fit into the framework assumed in [23]. The straight line is the trajectory that minimizes path length and the distance function returns that path length.

Whereas RRT planners using steering methods have been most successful in practice and come with guarantees on probabilistic completeness, not requiring a steering method was one of the selling points when the RRT was initially introduced.

4.2 Proof

We demonstrate that a kinodynamic RRT with fixed time steps and best-input extension is not generally probabilistically complete. The proof uses a counter example.

The RRT variant we are considering here selects both the node and the input by evaluating closeness to the sampled state according to the provided distance metric ρ. In order for a node to get selected it must be the closest one to the sample. The same goes for the input: In order to be selected, the successor state resulting from the input must be the closest one to the sample among all the successor states resulting from applying inputs from the current node. Even though for every node and for every input there exist states such that the considered node or the considered successor state is closest, in order for a specific input to be selected for extension from a specific node, more is required: (1) The specific node must be the closest to the sample *and* (2) among all the successor states resulting from applying inputs from the specific node, the state resulting from the specific input must be the closest. We provide an example case in which there is no state that could be sampled that satisfies both requirements.

The system used as counter example is described in Section 4.2.1. In Section 4.2.2 we present a possible intermediate tree and in Section 4.2.4 we demonstrate that the considered RRT variant cannot explore the full reachable state space from that intermediate tree because there exists a node and an input such that no sampled state results in selecting both of them. Section 4.2.3 provides some background of Voronoi regions, which are used in the proof in Section 4.2.4.

4.2.1 Counter Example

Consider the following system with a 2-dimensional state vector $[x_1, x_2]$, a scalar input u and no obstacles.

$$\dot{x}_1 = u \tag{55}$$

$$\dot{x}_2 = u^2 - 3 \tag{56}$$

$$\text{with} \quad |u| \leq 1 \tag{57}$$

Note that $-4 \leq \dot{x}_2 \leq -2$ and thus the system is always moving in negative direction along the x_2 axis. The set of possible successor states after a time step of Δt from the current state is a segment of a parabola. Figure 8 shows the set of states reachable within $3\Delta t$ from some initial state $\boldsymbol{x}_{\text{init}}$ assuming constant input during a fixed time step Δt.

Observe that the fact that the system is restricted to always move in the negative direction of the x_2 axis makes it *impossible to revisit an earlier state.* Also, all states at $t = \Delta t$ and $t = 2\Delta t$ are only reachable at one specific point in time.

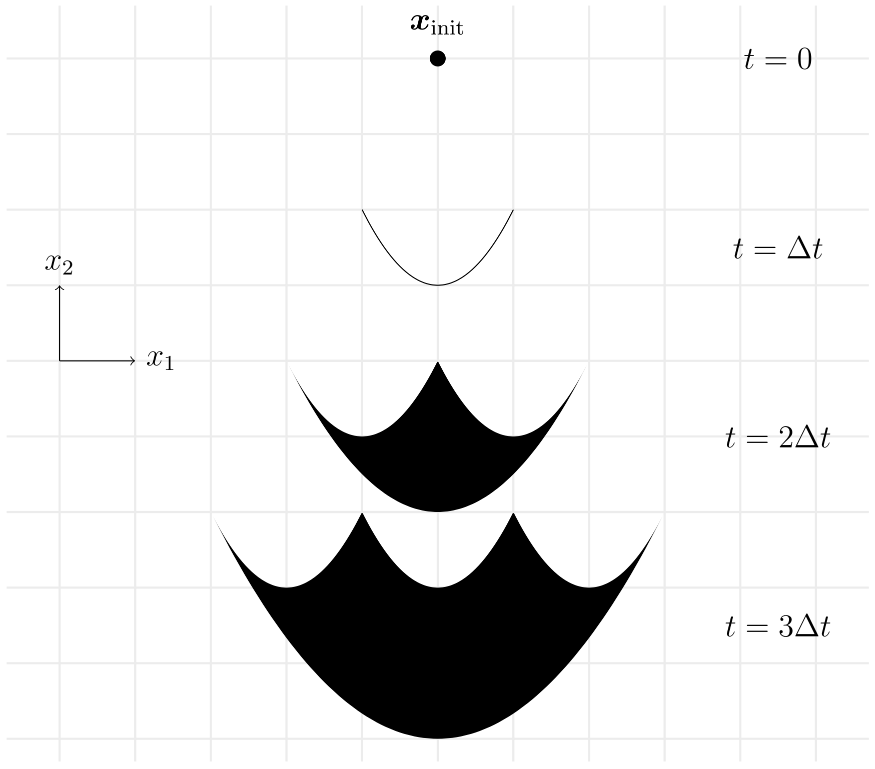


Figure 8: States reachable from $\boldsymbol{x}_{\text{init}}$

Our counter example uses a Euclidean distance for the RRT algorithm.

4.2.2 Intermediate Tree

A probabilistically complete algorithm must be able to explore the whole reachable space from any intermediate tree that the algorithm might produce. Figure 9 shows what the RRT could look like after two extensions from the initial state. The new nodes $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ sit at the ends of the parabola segment that represents the reachable space at time Δt.

If the algorithm was probabilistically complete, it would still be able to explore the whole reachable space. However, we show that given this tree configuration, the RRT is never going to explore the state space areas shown in gray, even though they are reachable by the system. The parabola segment at $t = \Delta t$ is never explored except its endpoints. The unexplored space at $t = 2\Delta t$ and $t = 3\Delta t$ is just the result of the unexplored parabola segment at $t = \Delta t$, since getting there requires moving through

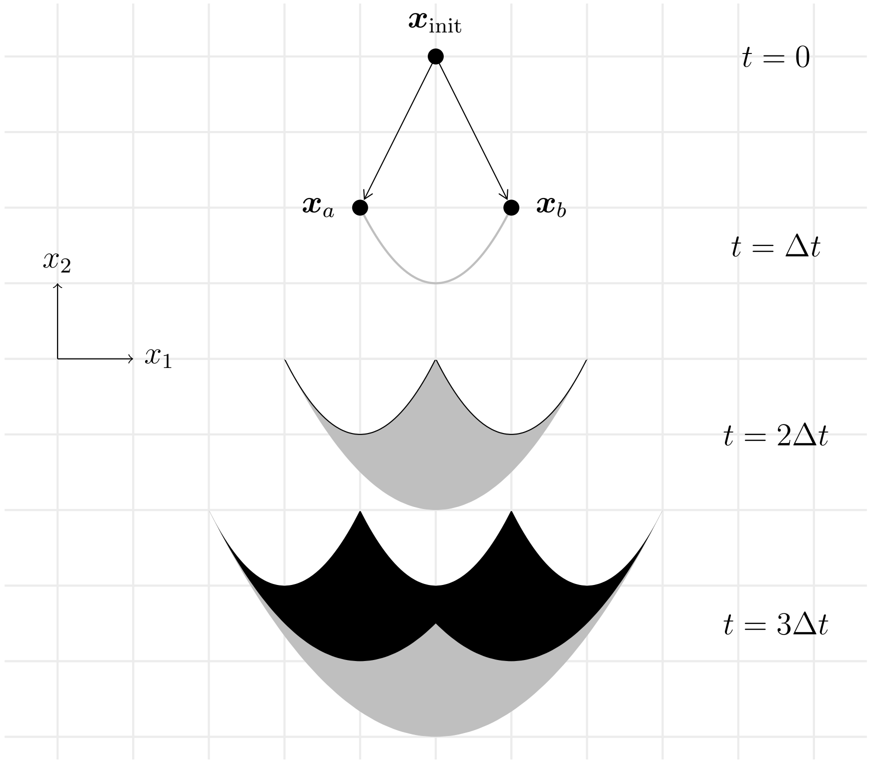


Figure 9: RRT after two extensions. Gray areas of the state space are never explored.

a state in the interior of the parabola segment. Also, note that the unexplored space at $t = 2\Delta t$ and $t = 3\Delta t$ does not play a role for our proof, since the inability of the RRT to explore the interior of the parabola segment is enough for it to not be probabilistically complete. Part of the unexplored space at $t = 3\Delta t$ could potentially still be explored at $t = 4\Delta t$, since it overlaps with the reachable space at $t = 4\Delta t$, which is not shown in the figure.

4.2.3 Background: Voronoi Regions of Sites

Even though Voronoi regions are a well-known concept, we are going to review them in this section since our proof in the next section uses the less common concept of a Voronoi region of an infinite set of points instead of only Voronoi regions of single points.

Consider k subsets $S_i \subset \mathcal{X}$ with $i = 1 \dots k$ such that $\forall i \neq j : S_i \cap S_j = \emptyset$. The sets S_i are called *sites*. The Voronoi region of site S_i is the set of all points that is closer to S_i according to our distance metric ρ than to any other site. Or more formally

$$\mathrm{Vor}(S_i) = \{x \in \mathcal{X} \mid \exists p \in S_i \; \forall j = 1 \ldots k \; \forall q \in S_j \colon \rho(x,p) \leq \rho(x,q)\} \tag{58}$$

Note that in the common case all sites S_i only contain a single point, but we are also going to make use of a site S_i containing infinitely many points. Also note that $\mathrm{Vor}(S_i)$ does not only depend on S_i but on all S_j with $j = 1 \ldots k$. A Voronoi diagram is a tuple $(\mathrm{Vor}(S_i))_{i \in \{1 \ldots k\}}$ of all the k Voronoi regions.

4.2.4 Non-Exploration of Parabola Segment

We now look closer at $t = \Delta t$ to determine why the interior of the parabola segment is not explored by the RRT algorithm. As mentioned in Section 4.2.1, because the example system is constrained to always move in negative x_2 direction, the states on the parabola segment can only be reached at time $t = \Delta t$. Thus, the parabola segment can only be explored by extending the tree from the root node.

To extend the tree to the parabola segment, the random sample of the RRT algorithm must fall in the Voronoi region of the root node. The Voronoi regions of the three tree nodes (which are the Voronoi sites here) are shown in Figure 10. The root node's Voronoi region is shaded with lines.

Now assume the RRT samples somewhere in the root node's Voronoi region and thus selects the root node as the nearest neighbor for extension. The next step is to choose the input to use to extend the tree from the root node. The RRT variant we are considering chooses the input such that the distance of the new child node to the sampled state is minimized. Similar to the way the closest node to the sample gets picked by the RRT algorithm, now the closest successor state of the selected node gets picked. We will now look at Voronoi regions of different successor states of the root node. We consider three sites and their Voronoi regions. Two sites are defined to be the two end points of the parabola segment and the third site is the entire rest, the interior, of the parabola segment. Note that the latter Voronoi site

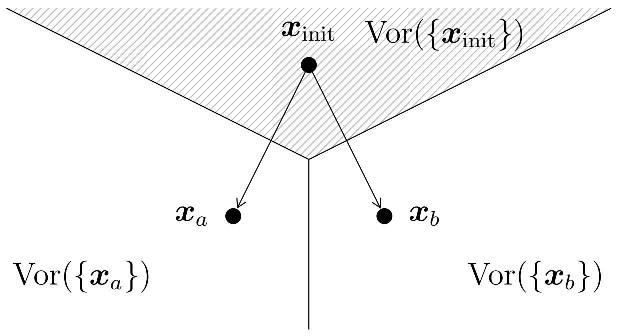

$\boldsymbol{x}_{\text{init}}$
$\text{Vor}(\{\boldsymbol{x}_{\text{init}}\})$
$\boldsymbol{x}_a$
$\boldsymbol{x}_b$
$\text{Vor}(\{\boldsymbol{x}_a\})$
$\text{Vor}(\{\boldsymbol{x}_b\})$

Figure 10: Voronoi diagram of the three tree nodes, i. e. of the three Voronoi sites $S_1 = \{\boldsymbol{x}_{\text{init}}\}$, $S_2 = \{\boldsymbol{x}_a\}$ and $S_3 = \{\boldsymbol{x}_b\}$. The root node's Voronoi region is shaded with lines.

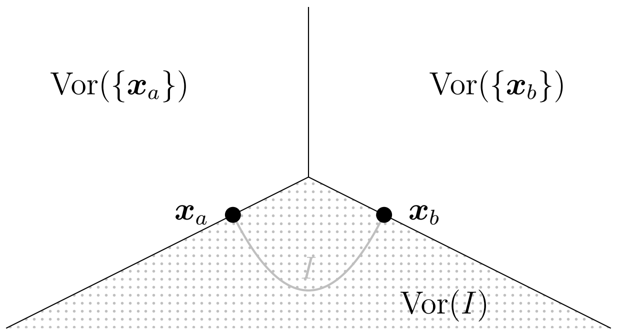

$\text{Vor}(\{\boldsymbol{x}_a\})$
$\text{Vor}(\{\boldsymbol{x}_b\})$
$\boldsymbol{x}_a$
$\boldsymbol{x}_b$
I
$\text{Vor}(I)$

Figure 11: Voronoi diagram of the successor states of the root node. Three Voronoi sites are considered: the two endpoints of the parabola segment, $S_1 = \{\boldsymbol{x}_a\}$ and $S_2 = \{\boldsymbol{x}_b\}$, and its interior $S_3 = I$. The Voronoi region of the interior I of the parabola segment is shaded with dots.

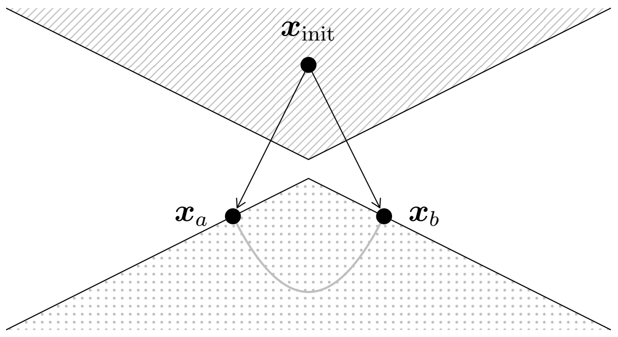


Figure 12: Combining the two Voronoi diagrams from Figures 10 and 11: Voronoi regions of the root node (lines) and the interior of the parabola segment (dots). They do not overlap.

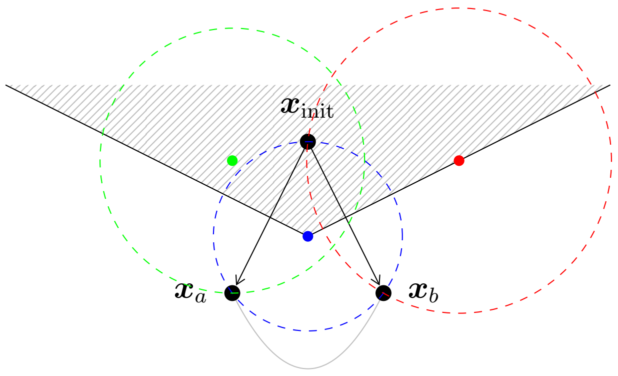


Figure 13: Points within the root node's Voronoi region are closer to either one of the endpoints of the parabola segment than to its interior.

consists of infinitely many states. The three Voronoi regions of those sites are shown in Figure 11. The Voronoi region of the interior of the parabola segment is shaded with dots.

For the RRT to explore the interior of the parabola segment, the sampled state must lie in both, the Voronoi region of the root node and the Voronoi region of the interior of the parabola segment. However, as Figure 12 shows, the two Voronoi regions don't overlap. Thus, the RRT cannot explore the interior of the parabola segment and the algorithm is not probabilistically complete.

Figure 13 provides a slightly different illustration of the same fact that every sample in the root node's Voronoi region is closer to one of the endpoints of the parabola segment than to its interior. The figure shows three exemplary points within the root node's Voronoi region. The dashed circles around them show that the closest point on the parabola segment is always one of the endpoints.

4.2.5 Discrete Inputs

Above proof can easily be extended to the discrete case. For example we can replace the entire interior of the parabola by a single input that leads to a state in the center of the parabola segment. This means Eq. 57 is replaced by $u \in \{-1, 0, 1\}$. The voronoi regions for this discrete counter example are shown in Figure 14. Similarly

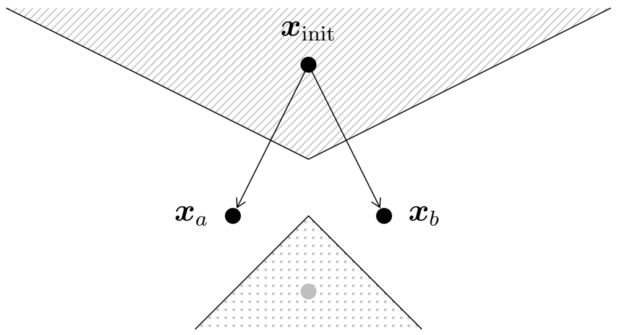


Figure 14: Discrete case: Voronoi regions of the root node (lines) and the zero-input state (dots)

to the continuous case, the zero-input state shown in gray will never be explored.

However, in the discrete-input case the RRT algorithm can be easily adapted to be probabilistically complete by making sure that no input is applied to the same node twice [42]. This forces the RRT to eventually try to expand all inputs of a node. This adaption is not possible in the continuous-input case.

4.3 Conclusion

We showed that a common variant of kinodynamic RRTs is not probabilistically complete. This contradicts general perception that RRTs are inherently probabilistically complete. Instead, probabilistic completeness depends on the implementation details of the RRT, the specific problem and/or the chosen distance metric. Whether the RRT variant considered here can be made probabilistically complete by introducing constraints on the problem or distance metric is left open for further research. The question whether kinodynamic RRTs with a variable time step are probabilistically complete is also left open.

Even though RRTs were initially designed for not requiring a steering method, the finding in this chapter provides an argument for using RRTs with a steering method as we do in Chapter 6.

CHAPTER V

STEERING METHOD: DOUBLE-INTEGRATOR MINIMUM TIME (DIMT)

5.1 Overview

This chapter presents the steering method that we use 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 DOFs. Furthermore, unlike claimed in [18] and [23], 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 [29]. Section 5.3 describes how to calculate the minimum time for an individual DOF and Section 5.4 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

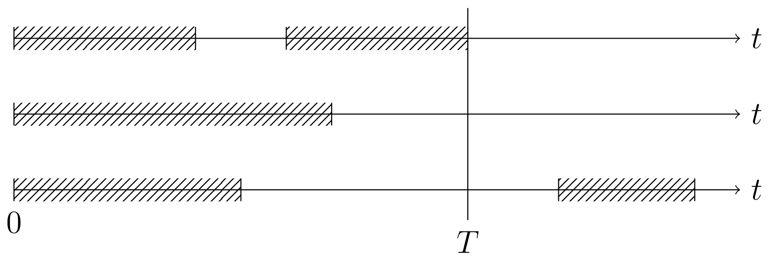


Figure 15: Determining the overall minimum time t from the individual DOF's minimum times and infeasible time intervals

minimum time and is not within any infeasible time interval (see Figure 15).

After the overall minimum time T has been determined, Section 5.5 describes how to calculate a trajectory for each individual DOF to reach its goal state at time T.

All the following sections only consider one individual DOF.

5.2 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 [18].

The solution to the quadratic equation

$$ax^2 + bx + c = 0 \tag{59}$$

is given by

$$q = -\frac{1}{2}\left(b + \operatorname{sgn}(b)\sqrt{b^2 - 4ac}\right) \tag{60}$$

$$x_1 = \frac{q}{a} \qquad\qquad x_2 = \frac{c}{q} \tag{61}$$

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

For calculating the minimum time of an indivisual DOF (Section 5.3), we need the greater solution of a quadratic equation. For calculating the infeasible interval of a DOF (Section 5.4) both the lesser and greater solutions to a quadratic equation are

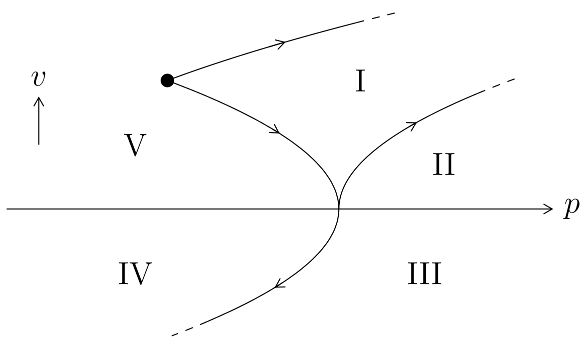


Figure 16: Regions of the phase plane the goal state can be in relative to the start state. The shown separating trajectories are at an acceleration limit. Without loss of generality this figure assumes a positive start velocity.

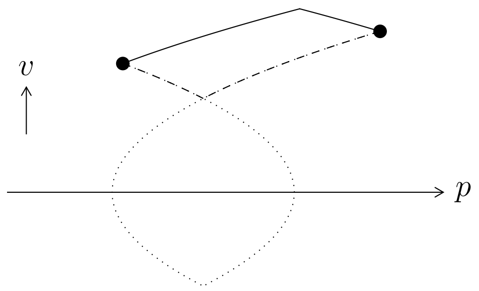


Figure 17: Extremal-acceleration trajectories defining the minimum time (solid) and the lower (dashed) and upper (dotted) bounds of the infeasible interval.

used. For calculating the fixed-time trajectories for the individual joints (Section 5.5), we make use of the solution with the greater absolute value.

5.3 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 [18] 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. Figure 16 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.

$$\Delta p_{\text{acc}} = \frac{1}{2}(v_1 + v_2)\frac{|v_2 - v_1|}{a_{\max}} \tag{62}$$

$$\sigma = \operatorname{sgn}(p_2 - p_1 - \Delta p_{\text{acc}}) \tag{63}$$

$$a_1 = -a_2 = \sigma a_{\max} \tag{64}$$

$$v_{\text{limit}} = \sigma v_{\max} \tag{65}$$

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}.

$$a_1 t_{a1}^2 + 2v_1 t_{a1} + \frac{v_2^2 - v_1^2}{2a_2} - (p_2 - p_1) = 0 \tag{66}$$

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

$$t_{a2} = \frac{v_2 - v_1}{a_2} + t_{a1} \tag{67}$$

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. 66 is always the greater one of the two (see Section 5.2). The total time is $T = t_{a1} + t_{a2}$.

Whereas Eq. 66 is the same as in [18], the constraints on t_{a1} to find the valid one of the two solutions are given incorrectly in [18].

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

$$t_{a1} = \frac{v_{\text{limit}} - v_1}{a_1} \tag{68}$$

$$t_v = \frac{v_1^2 + v_2^2 - 2v_{\text{limit}}^2}{2v_{\text{limit}}a_1} + \frac{p_2 - p_1}{v_{\text{limit}}} \tag{69}$$

$$t_{a2} = \frac{v_2 - v_{\text{limit}}}{a_2} \tag{70}$$

5.4 Infeasible Time Interval

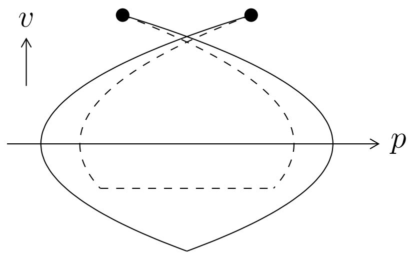
A DOF has an infeasible time interval if and only if the goal state is in region I of Figure 16. Figure 17 visualizes the reason for the existense of the infeasible interval. It shows the trajectories that define the minimum time as well as the bounds of the infeasible interval. Reaching the goal state 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}.

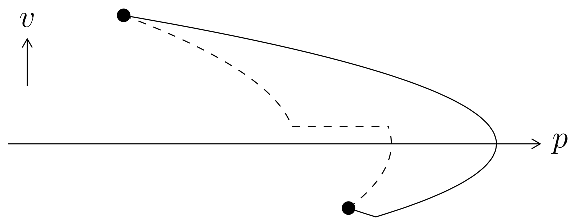
$$a_1 = -a_2 = -\sigma a_{\max} \tag{71}$$

$$v_{\text{limit}} = -\sigma v_{\max} \tag{72}$$

We solve Eq. 66 with a_1 and a_2 as defined in Eq. 71. The lower bound of the infeasible interval is given by the lesser solution to the quadratic equation and



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



(b) Region III

Figure 18: Comparison of fixed-time trajectories as calculated by this thesis and Hauser et al. [18] (solid) with previous work by Kröger [29] (dashed).

the upper bound is given by the greater solution (see Section 5.2). 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. 68 - 70 with a_1, a_2 and v_{limit} as defined in Eq. 71 and 72.

5.5 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. Kröger [29] 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. [18] in contrast choose the trajectory such that it minimizes the maximum absolute value of acceleration. However, unlike in [29] 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. [18] and minimize the maximum absolute value of acceleration. Figure 18 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. [18]. The only small difference is that we are not evaluating both solutions to Eq. 73.

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

$$T^2 a_1^2 + (2T(v_1+v_2) - 4(p_2-p_1))\, a_1 - (v_2-v_1)^2 = 0 \tag{73}$$

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

$$t_{a1} = \frac{1}{2}\left(\frac{v_2 - v_1}{a_1} + T\right) \qquad\qquad t_{a2} = T - t_{a1} \tag{74}$$

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

$$v_{\text{limit}} = \text{sgn}(a_1)\, v_{\text{max}} \tag{75}$$

with a_1 as given by Eq. 73. The new accelerations satisfying the velocity limits are then given by

$$a_1 = -a_2 = \frac{(v_{\text{limit}} - v_1)^2 + (v_{\text{limit}} - v_2)^2}{2(v_{\text{limit}}T - (p_2 - p_1))} \tag{76}$$

The durations are given by Eq. 68 - 70 while using v_{limit}, a_1 and a_2 as defined in Eq. 75 and 76.

CHAPTER VI

PROBABILISTICALLY COMPLETE PLANNING

6.1 Overview

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 [38, 39, 40, 41] 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, as described in Chapter 4, the most common variant of the kinodynamic RRT is not probabilistically complete in general.

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 use a fast, non-iterative steering method that can efficiently solve the boundary value

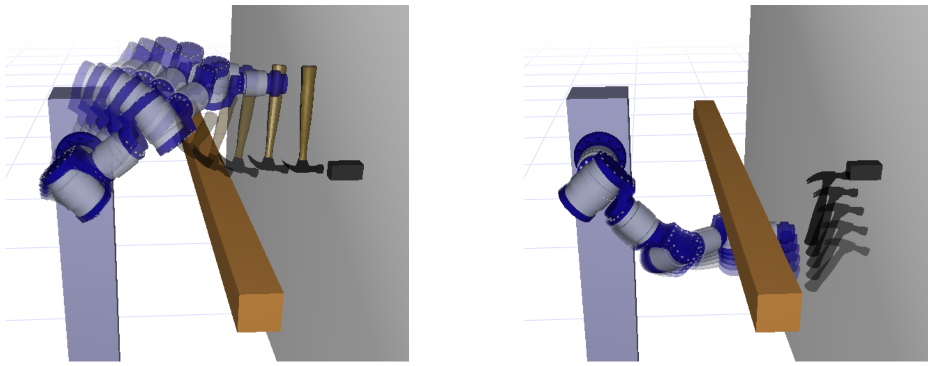

(a) DIMT-RRT (b) Geometric RRT

Figure 19: The acceleration-limited planner (a) is able to hit the nail at the correct velocity while the geometric planner (b) is not.

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. Figure 19 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 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. This application is not examined in this thesis and is left for future work.

Our planner, described in Section 6.2, combines the RRT algorithm with the steering method presented in Chapter 5. We compare our planner against a geometric and kinodynamic RRT in Section 6.3.

6.2 DIMT-RRT

Our planner combines the existing bidirectional RRT-Connect planner [30] with the steering method presented in Chapter 5. Our DIMT-RRT planner is shown in Algorithm 4.

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 5 makes a connection attempt from a tree to the sample. Unlike in [30] 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 Chapter 5 (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 Sections 5.3 and 5.4, but not Section 5.5 are used for the distance calculation.

Once the closest node is found, we calculate the trajectory σ as described in Section 5.5 (line 2) and check it for collisions (line 3). If it is collision-free, we add the

Algorithm 4: 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 5: 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;
```

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).

6.3 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 $v_{\max} = \pi/2$ rad/s and $a_{\max} = \pi/4$ 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

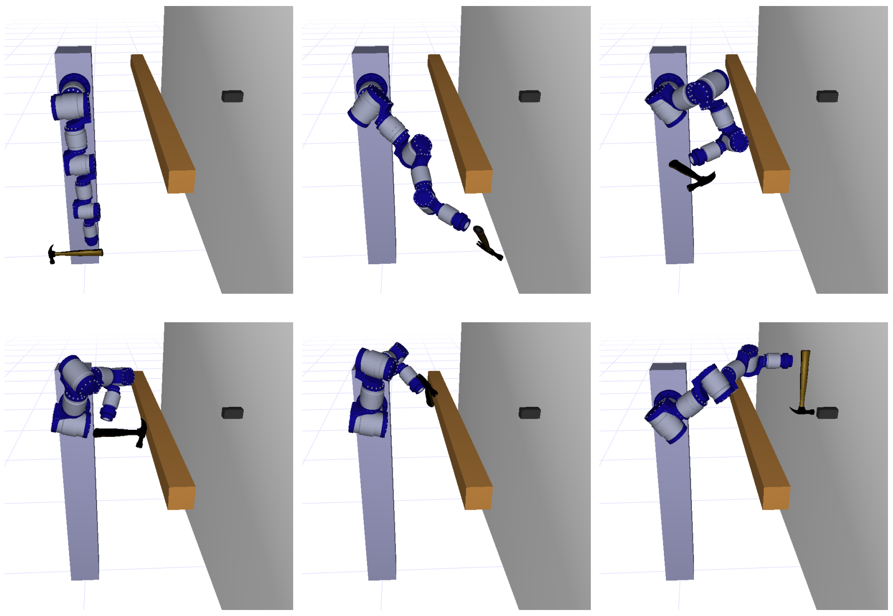


Figure 20: The DIMT-RRT planner hits the nail at the desired velocity

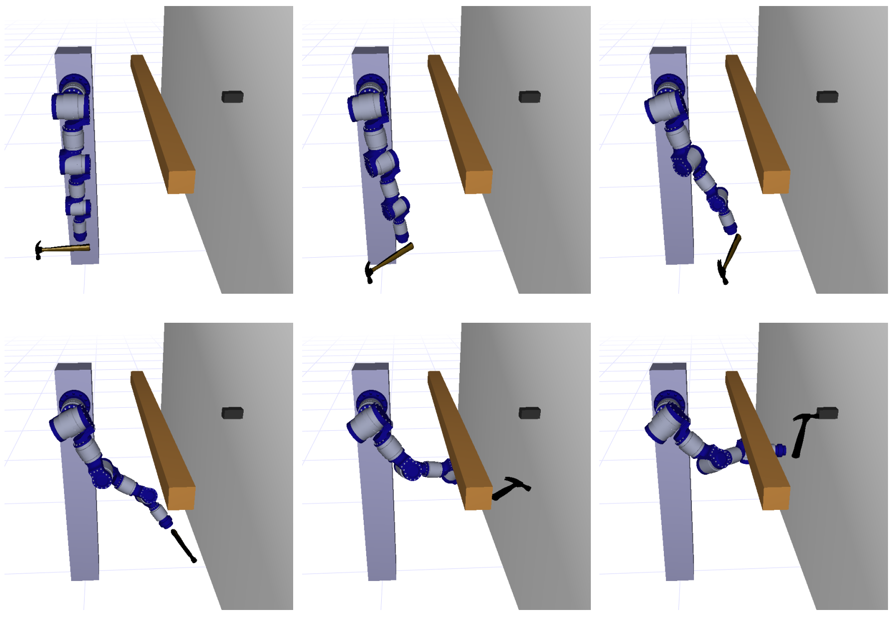


Figure 21: The geometric RRT planner reaches the nail but not at the desired velocity

this thesis. 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 (Figure 21). 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 (Figure 20). We check trajectories for collision discretely every 0.1 seconds.

6.3.1 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 [18]. The shortened trajectory is shown in Figure 20 and Figure 19(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 3 shows the computation time of the DIMT-RRT

Table 3: 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.

	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

planner and the length of the generated trajectory with and without smoothing using shortcuts. In average the planner finds a feasible trajectory in only 56 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 steering method allows the RRT to solve the problem very efficiently without exploring much of the state space.

6.3.2 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 Figures 21 and 19(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.

6.3.3 Comparison to Standard Kinodynamic RRT

We compare the performance of our DIMT-RRT algorithm to a standard kinodynamic RRT [41] 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

Table 4: Comparison of a single-tree variant of the DIMT-RRT and the standard kinodynamic RRT with a Euclidean distance metric.

	DIMT-RRT [this work]		Kinodynamic RRT [41]	
	extend	connect	extend	connect
# samples	9,019 $\pm$ 8,415	14.6 $\pm$ 10.8	$>$ 1,000,000	
# nodes	9,633 $\pm$ 8,096	434.1 $\pm$ 188.4	$>$ 900,000	$>$ 2,500,000
Computation time	19.8 s $\pm$ 37.1 s	37 ms $\pm$ 22 ms	$>$ 8 hours	$>$ 23 hours

the tree. The *extend* strategy, which corresponds to the original kinodynamic RRT [41], 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 Table 4 shows, the variation of the DIMT-RRT is able to solve the problem efficiently. The connect strategy is significantly faster than the extend strategy.

6.4 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.

CHAPTER VII

ASYMPTOTICALLY OPTIMAL PLANNING

7.1 Overview

While the method in the previous chapter is able to quickly find a feasible solution and tries to improve that solution by shortcutting, it does not yield an optimal solution. To achieve an optimal solution, in this chapter we combine the steering method with an asymptotically optimal sampling-based planner. We use an RRT* planner, but any other asymptotically optimal sampling-based planner could have been used in the same way, e.g. PRM*, RRG*, RRT$^{\#}$[2], FMT*[20] or BIT*[13]. Unlike general kinodynamic planning, acceleration-limited planning is well-suited for being solved by an asymptotically optimal sampling-based algorithm, since the steering method allows for exactly and efficiently connecting two states, which is crucial for the rewiring step of those algorithms.

In order to achieve a near-optimal solution, all these planners need to densely fill the state space with samples. However, in high-dimensional spaces this is very inefficient. In order to improve the efficiency and not fill the whole state space with samples, additional information can be used to fill only those parts of the state space that can improve the current solution. Following [12], we call this the *informed subset* of the state space. Assume we can calculate a lower bound on the optimal cost to move from the start through a sample and to the goal. If a steering method is available, this bound can be calculated as the optimal cost ignoring obstacles. If this lower bound is larger than the current solution, the sample cannot improve the current solution, is not part of the informed subset and can be ignored.

For systems without differential constraints and the Euclidean distance as cost function, the informed subset is ellipsoidal [10]. This ellipsoidal subset can be parameterized and sampled directly [12].

For systems with differential constraints, however, the subset of the space that can improve the solution is more complex than an ellipsoidal subset. No methods to sample directly within that space exist. Instead, samples within the informed subset can be found using rejection sampling, i.e. sampling the full state space and rejecting those samples that cannot improve the current solution. However, especially in high-dimensional spaces, the informed subset might only be a tiny fraction of the whole state space. In this case most samples get rejected and rejection sampling is very slow. In our experiments up to 99.99% of all samples get rejected. This can lead to a sampling-based planner spending most of its time rejection sampling.

To improve the efficiency of informed sampling without the need to explicitly parameterize the informed subset, we introduce *hierarchical rejection sampling (HRS)* in Section 7.2. HRS hierarchically combines partial samples into larger samples, making accept/reject decisions at every level. When a partial sample is rejected, only the partial information needs to be resampled. This leads to an efficiency improvement that is exponential in the number of dimensions.

7.2 Hierarchical Rejection Sampling

For systems with differential constraints no method to explicitly sample within the informed subset exists. Instead, we have to sample the full state space and then test the sample for validity. This can be very inefficient as the informed subset might only be a tiny fraction of the whole state space. This is especially true in high-dimensional spaces. Therefore, we propose hierarchical rejection sampling to find valid samples faster without having to explicitly parameterize the informed subset.

Instead of sampling a whole state and then deciding whether to accept or reject it, we build up a sample hierarchically making accept or reject decisions at every

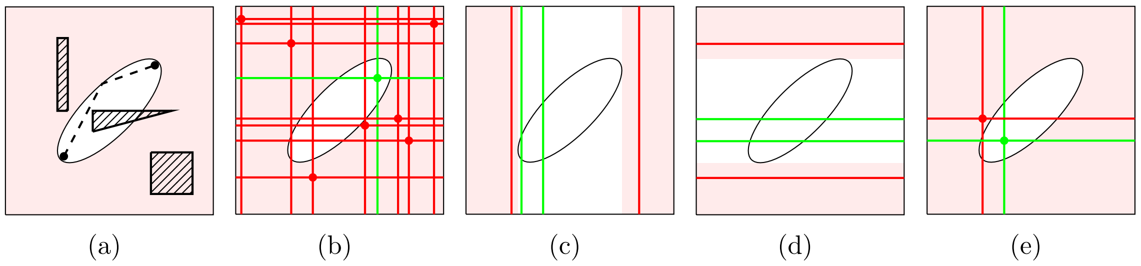


Figure 22: Illustration of hierarchical rejection sampling using a very simple example (2 dimensions, no differential constraints): (a) current solution trajectory and informed subset, (b) standard rejection sampling, (c) - (e) hierarchical rejection sampling.

level based on partial state samples. Only the highest level in the hierarchy makes a decision based on the whole state sample. But instead of sampling the whole state, it combines two partial samples retrieved from the next lower level. The samples passed to the highest level from the next lower level have already been tested by the next lower level based on partial information. If the partial information is enough to reject the partial sample, only the partial sample gets resampled. In certain cases this reduces the amount of samples necessary and speeds up the sampling dramatically.

7.2.1 A Simple Example

Figure 22 uses a very simple example to demonstrate hierarchical rejection sampling and why it leads to more efficient sampling. The example uses a two-dimensional problem without differential constraints. Note, that we are using this simple example only to visualize hierarchical rejection sampling. Hierarchical rejection sampling is not needed for this problem, because the informed subset can be represented explicitly.

Figure 22(a) shows a planning problem with start and goal states and obstacles. It also shows current, suboptimal solution trajectory. The white ellipse marks the informed subset, in which a sample has to lie in order to be able to improve the solution trajectory. The informed subset covers 1/8 of the state space. The axis-aligned bounding box around it shown in Figure 23 is a square and covers 1/4 of the state space.

Figure 22(b) shows samples produced by standard rejection sampling. Since the informed subset covers 1/8 of the whole state space, we need in average 8 samples to find one valid sample (shown in green). Since the state space is 2-dimensional, these 8 samples consist of 16 scalar samples (shown as lines).

Figure 22(c)-(e) visualize hierarchical rejection sampling. In Figure 22(c) only the horizontal dimension is considered. There are 4 partial, 1-dimensional samples shown. Even though only partial samples are conidered, 2 out of 4 samples can be rejected. This is because mo matter what the value of the vertical dimension is, those partial samples can never result in a full sample that lies within the informed subset. Figure 22(c) shows the same for the vertical dimension with 2 out of 4 samples being rejected. Figure 22(d) combines the 4 partial, 1-dimensional samples into 2 full samples. Only one of the 2 full samples lies within the informed subset. In Figures 22(c) and 22(d) a total of 8 scalar samples are generated.

In this simple example in average hierarchical rejection sampling generates half as many scalar samples as standard rejection sampling. Admittedly, this is not a big improvement. However, the improvement grows bigger when increasing the number of dimensions. In fact, the efficiency improvement caused by hierarchical rejection sampling grows exponentially with the number of dimensions. Consider the axis-aligned bounding box around the informed subset. The edge length of the box is half the length of the state space in every dimension. While Figure 23 shows this box in 2 dimensions, we can straight-forwardly extent this scenario to d dimensions. The volume of the box is $\left(\frac{1}{2}\right)^d$ of the volume of the whole state space. Thus, for standard rejection sampling the average number of scalar samples required to find one sample within the box is

$$d2^d \tag{77}$$

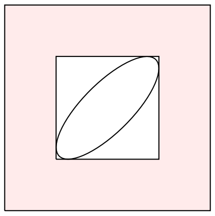


Figure 23: Cube with half the edge length of the state space in every dimension

If we use hierarchical rejection sampling and first sample each dimension individually, we require in average 2 scalar samples for each dimension. Since we are considering the whole bounding box instead of just the informed subset, a full sample retrieved by combining valid partial samples is always valid. Thus, for hierarchical rejection sampling the average number of scalar samples required to find one sample within the box is

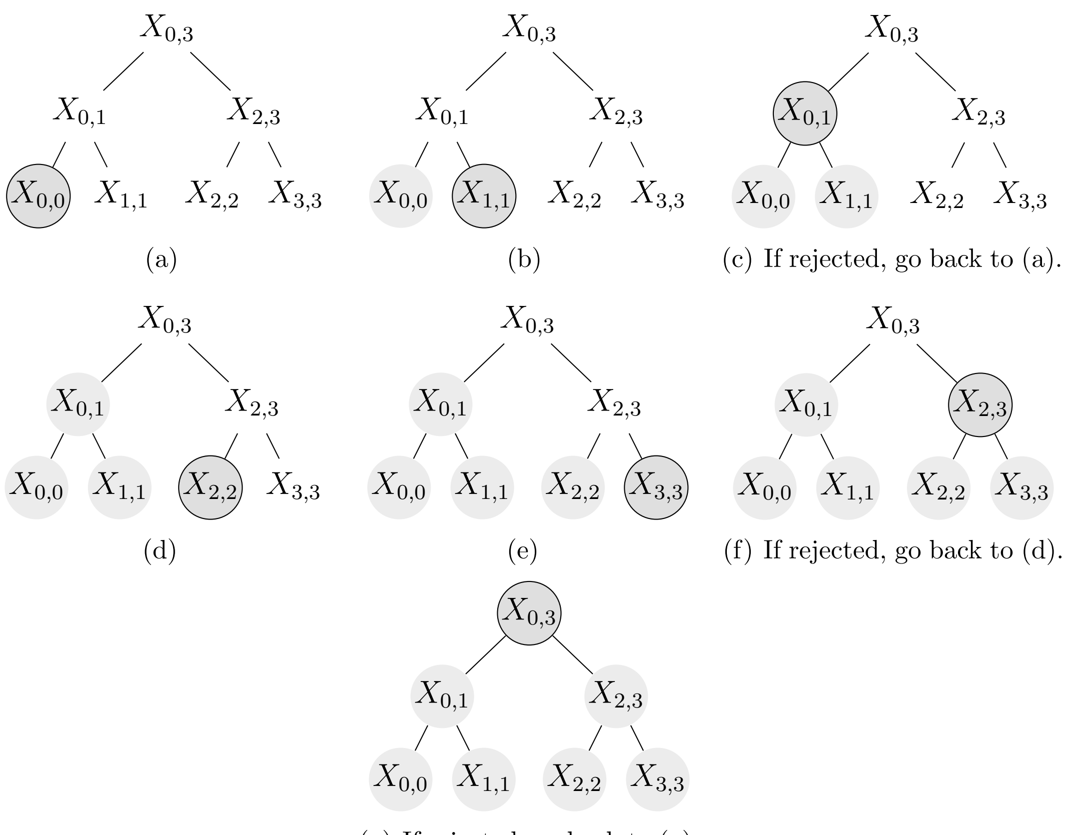
$$2d \tag{78}$$

The fact that the number of scalar samples required by hierarchical rejection sampling in this example is only linear in the number of dimensions instead of exponential explains the advantage of hierarchical rejection sampling in high-dimensional spaces.

7.2.2 Algorithm

Figure 24 visualizes the hierarchical rejection sampling algorithm using a 4-dimensional problem. At every node reject/accept decisions are made based on partial state information. $X_{i,j}$ represents subspace of the state space X that only consists of the dimensions i to j. Leaf nodes sample a single dimension until a scalar sample is accepted. Interior nodes combine two partial samples. If the combined sample is rejected, its whole subtree gets resampled.

Algorithm 6 shows a recursive implementation of hierarchical rejection sampling. The parameters i and j specify the currently considered subspace $X_{i,j}$. Dimensions i - j are considered for calculating a lower bound on the cost of a trajectory through a sample and for making an accept/reject decision. In order to make an accept/reject



(a) (b) (c) If rejected, go back to (a).

(d) (e) (f) If rejected, go back to (d).

(g) If rejected, go back to (a).

Figure 24: Hierarchical rejection sampling: The sample gets build bottom-up along a tree structure. Whenever a node rejects part of a state, only its subtree gets resampled. Parts that have already been sampled are shown in gray. The active node is circled.

decision, the start state $\boldsymbol{x}_{\text{start}}$, goal state $\boldsymbol{x}_{\text{goal}}$ and the cost c_{best} of the best solution trajectory found so far are provided by the planning algorithm. $\boldsymbol{x}$, c_{start} and c_{goal} are outputs of the algorithm. $\boldsymbol{x}$ is the state sample that is being generated. c_{start} and c_{goal} are lower bounds on the cost to move from the start state to the sample and from the sample to the goal state respectively. The algorithm also updates a vector $\boldsymbol{n}$ with a size equaling the number of nodes in the tree representing the sampling hierarchy: $2d-1$. Each element of $\boldsymbol{n}$ is associated with one node in the sampling hierarchy and stores the number of samples drawn by that node from its children. We call these the numbers of *explicit samples*. The numbers of explicit samples are updated in lines 6 and 15. For more details on the use of $\boldsymbol{n}$ see Section 7.2.4.

Algorithm 6: HRS$(i, j, \boldsymbol{x}, c_{\text{start}}, c_{\text{goal}})$

```
1  if $i = j$ then
2      repeat
3          $x_i \leftarrow$ SampleLeaf$(i)$;
4          $c_{\text{start}} \leftarrow$ CalculateLeaf$(\boldsymbol{x}_{\text{start}}, \boldsymbol{x}, i)$;
5          $c_{\text{goal}} \leftarrow$ CalculateLeaf$(\boldsymbol{x}, \boldsymbol{x}_{\text{goal}}, i)$;
6          $n_{2i} \leftarrow n_{2i} + 1$;
7      until $c_{start} + c_{goal} < c_{best}$;
8  else
9      $m \leftarrow \lfloor \frac{i+j}{2} \rfloor$;
10     repeat
11         HRS$(i, m, \boldsymbol{x}, c_{\text{start}}, c_{\text{goal}})$;
12         HRS$(m+1, j, \boldsymbol{x}, c'_{\text{start}}, c'_{\text{goal}})$;
13         $c_{\text{start}} \leftarrow$ Combine$(\boldsymbol{x}_{\text{start}}, \boldsymbol{x}, i, m, j, c_{\text{start}}, c'_{\text{start}})$;
14         $c_{\text{goal}} \leftarrow$ Combine$(\boldsymbol{x}, \boldsymbol{x}_{\text{goal}}, i, m, j, c_{\text{goal}}, c'_{\text{goal}})$;
15         $n_{2m+1} \leftarrow n_{2m+1} + 1$;
16     until $c_{\text{start}} + c_{\text{goal}} < c_{best}$;
```

Lines 9 - 16 handle interior nodes, while lines 1 - 7 handle leaf nodes. Interior nodes divide the considered set of dimensions in two parts. Line 9 calculates the split point. Lines 11 and 12 recursively call the algorithm to generate two partial samples. Using only the partial information available from dimensions i - j, lines 13 and 14 calculate a lower bound on the costs to move from the start to the sample and from the sample to the goal. If the sum of the lower bounds is larger than c_{best}, the sample is rejected and the process is repeated. The process for leaf nodes is similar. But instead of recursively sampling two partial samples, it just samples a scalar in line 3.

7.2.3 Ordering and Grouping of Dimensions

The algorithm assumes an ordering of the dimensions. It is advantageous to order dimensions such that those with a higher degree of dependency on each other in relation to cost are combined on lower levels of the hierarchy. In our experiments, position and velocity of a single joint have a higher dependency on each other than dimensions associated with different joints. We do not expect the ordering of the different joints to have a major effect. Therefore, in our experiments, we just use the

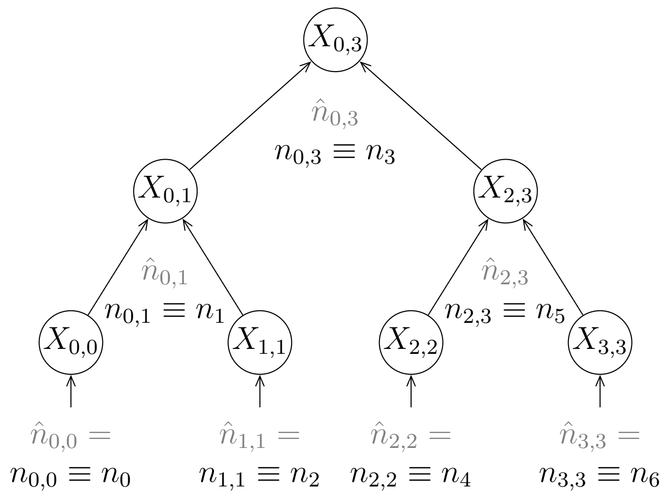


Figure 25: Visualization of numbers of explicit (n) and implicit samples ($\hat{n}$) for every node.

physical ordering of the joints within the arm.

For simplicity we assumed so far that leaf nodes consider only a single dimension. However, the algorithm works the same if leaf nodes represent multiple dimensions. In fact, in our experiments we do not sample position and velocity of a single joint hierarchically. They are both part of the same leaf node and sampled and rejected together.

7.2.4 Implicit Samples

For determining the connection threshold of the RRT* and for evaluation purposes, we need to know the total number of samples including rejected ones. With standard rejection sampling we can easily count the states we sampled. With hierarchical rejection sampling, however, it is not as easy, since we do not explicitly sample every state. Instead, we need to estimate the number of samples that would have been necessary if we had uniformly sampled complete states. We call these samples *implicit samples.*

During the sampling process we collect information that allows us to calculate the number of implicit samples generated since the last time the informed subset changed, i.e. the solution cost decreased. Every time the solution cost decreases the number of implicit samples is updated and the collected information is reset to zero. In the following we describe what information is collected and how the number of implicit samples since the last solution cost decrease is calculated. Note, the rest of this chapter only considers the number of implicit samples since the last solution cost decrease and, thus, assumes the informed subset does not change.

For each node $X_{i,j}$ in the tree we store the number $n_{i,j}$ of *explicit samples* it has generated. This means the number of times it has queried its two children for a sample and made an accept/reject decision. This is also the number of times its two children have accepted a sample. Note that both children accept the same number of samples, because they both get queried the same number of times for a sample by $X_{i,j}$. This is visualized in Figure 25. We can think of $n_{i,j}$ as two flows of samples from its children to $X_{i,j}$.

We can also assign a number $\hat{n}_{i,j}$ of *implicit samples* to each node. The numbers $\hat{n}_{i,j}$ do not need to be stored explicitly since they can be calculated from the numbers of explicit samples $n_{i,j}$ of all nodes. The number $\hat{n}_{i,j}$ of implicit samples of a node $X_{i,j}$ can be calculated recursively from the number $n_{i,j}$ of explicit samples and the numbers $\hat{n}_{i,m}$ and $\hat{n}_{m+1,\,j}$ of implicit samples of its child nodes. For each child node we can calculate the percentage of implicit samples it accepts. This percentage or probability is calculated as the quotient of accepted samples and total implicit samples

$$\frac{n_{i,j}}{\hat{n}_{i,m}} \quad \text{and} \quad \frac{n_{i,j}}{\hat{n}_{m+1,\,j}} \tag{79}$$

where $m = \lfloor \frac{i+j}{2} \rfloor$.

Imagine the node $X_{i,j}$ would just use any two implicit samples from its child nodes, i. e. samples that were not necessarily accepted by its child nodes. The probability that this would generate a sample that would get accepted by both child nodes is the product of the probabilities that each child node accepts one of its own implicit samples

$$\frac{n_{i,j}}{\hat{n}_{i,m}} \frac{n_{i,j}}{\hat{n}_{m+1,\,j}} \tag{80}$$

We then divide the number $n_{i,j}$ of samples the children of $X_{i,j}$ accepted by the probability that an implicit sample would get accepted. This gives us the total number of implicit samples that would have been necessary if we had sampled in the traditional way.

$$\hat{n}_{i,j} = n_{i,j} \Big/ \left(\frac{n_{i,j}}{\hat{n}_{i,m}} \frac{n_{i,j}}{\hat{n}_{m+1,\,j}} \right) \tag{81}$$

$$= \frac{\hat{n}_{i,j} \cdot \hat{n}_{m+1,\,j}}{n_{i,j}} \tag{82}$$

For leaf nodes with $i = j$ the number of implicit samples equals the number of explicit ones.

$$\hat{n}_{i,i} = n_{i,i} \tag{83}$$

Combining the recursive formula with the base case for leaf nodes, we can bring the formula into closed form.

$$\hat{n}_{0,d-1} = \frac{\prod\limits_{X_{i,j} \text{ is leaf node}} n_{i,j}}{\prod\limits_{X_{i,j} \text{ is interior node}} n_{i,j}} \tag{84}$$

In order to store the values of $n_{i,j}$ we can flatten the two indeces i and j into a single one: the index of the node is the in-order traversal of the tree, i.e.

$$\text{For } i \neq j: \quad n_{i,j} \equiv n_{2m+1} \tag{85}$$

$$\text{For } i = j: \quad n_{i,j} \equiv n_{2i} \tag{86}$$

Eq. 84 can then be written as

$$\hat{n}_{0,d-1} = \frac{\prod\limits_{k=0}^{d-1} n_{2k}}{\prod\limits_{k=0}^{d-2} n_{2k+1}} \tag{87}$$

7.3 Experiments

7.3.1 Example Problems

We evaluate our planner on three example problems. Table 5 shows parameters of the problems. The problems are described below. All problems assume goal states to be given in joint space. We automatically generate a set of joint-space goals from a given workspace goal. This conversion is not part of the algorithms and evaluation presented in this thesis.

7.3.1.1 Problem 1: Hammering

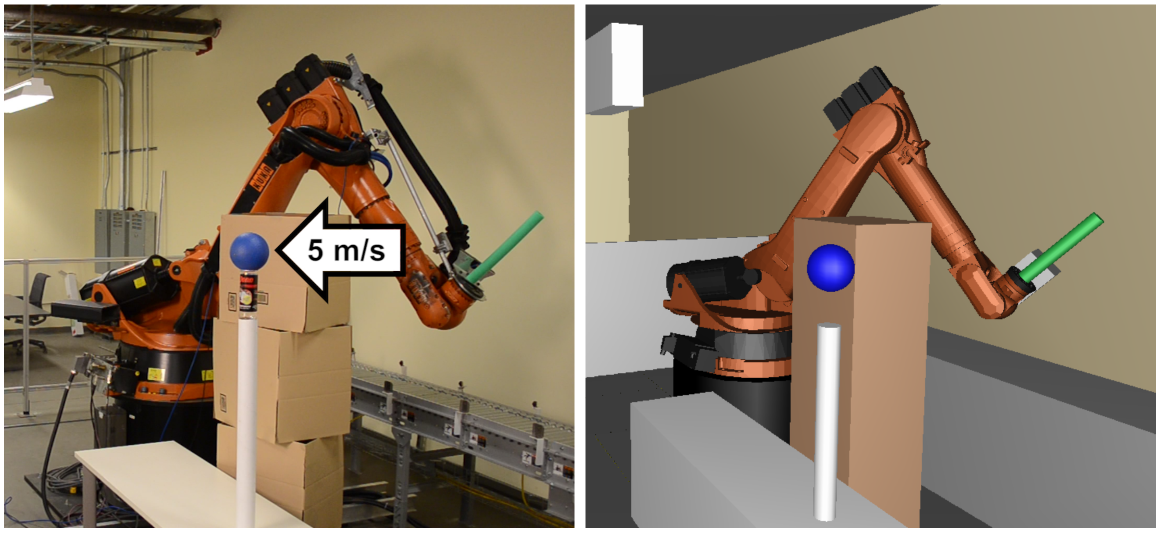
The first problem is the same as the one described in Section 6.3. A simulated 7-DOF robot arm is given the task to hit a nail at a given velocity while avoiding an obstacle. Because of the non-zero goal velocity the problem cannot be solved by a geometric planner. The state space, consisting of joint positions and velocities, is 14-dimensional. To the best of our knowledge the highest-dimensional space a kinodynamic RRT* has been applied to before is 10-dimensional [56]. The problem and a solution trajectory are shown in Figure 28.

7.3.1.2 Problem 2: Batting

Unlike the first problem, the second example problem involves a real robot. We use a KUKA KR 210 robot arm. This problem demonstrates that a trajectory generated by an acceleration-limited planner can be executed on a real robot at high velocity. Again, the goal is to hit an object, a ball in this case, at a velocity of 5 m/s. The

Table 5: Parameters of the example problems

task	DOF	state space dimensions	# goals	end-effector goal velocity	velocity limit	acceleration limit
hammering	7	14	100	0.6 m/s	90 °/s	45 $°/s^2$
batting	5	10	50	5.0 m/s	90 °/s	60 $°/s^2$
pick and place	7	14	2	0.0 m/s	90 °/s	45 $°/s^2$



(a) Reality (b) Simulation

Figure 26: Batting problem

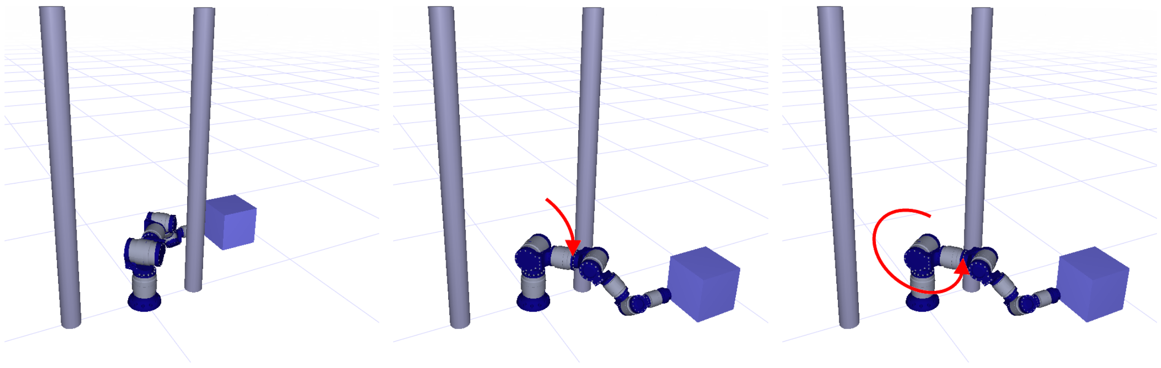
problem is visualized in Figure 26. A stack of boxes is placed strategically between the robot and the ball such that the shortest trajectory is in collision and the robot needs to plan around the obstacle. While the robot has 6 DOF, we only make use of 5 of them. Thus, the planning problem has 10 dimensions.

7.3.1.3 Problem 3: Pick and Place

While the previous two example problems involve non-zero goal velocities, in this problem, both, start and goal velocity are zero, which is typical for pick-and-place operations. The robot has to move a box through two vertical obstacles. There are two goal states, which are almost identical. Only the first joint differs by 360 degrees. This results in the planner being able to choose the direction in which to rotate the first joint. The problem is shown in Figure 27.

7.3.2 Computational Efficiency

We ran the kinodynamic RRT* algorithm described in Section 2.2.2.2 in combination with the double integrator steering method described in Chapter 5 on the three motion planning problems described in Section 7.3.1. We deviate from the standard



(a) Start state (b) Goal state 1 (c) Goal state 2

Figure 27: Pick-and-place problem

kinodynamic RRT* [23] by using an infinite connection threshold and by extending the tree all the way to a sample instead of just by a small time step. In addition, we employ hierarchical rejection sampling presented in Section 7.2 to improve efficiency. Figures 28, 29, 30 and 31 show solution trajectories for the example problems retrieved after 60 s of computation time.

Below we evaluate the convergence of the RRT* algorithm and the effect of hierarchical rejection sampling by comparing it to both, uninformed sampling and standard rejection sampling. Uninformed sampling does not reject any samples. Standard rejection sampling samples a whole state and then makes a accept/reject decision. For a fair comparison we implement standard rejection sampling as efficient as possible. Even standard rejection sampling might make reject decisions early before having finished calculating the lower bound on the cost to move through the sample. To do that it passes the current solution cost to the steering method to allow the steering method to abort the calculations as soon as the current solution cost is exceeded. However, unlike hierarchical rejection sampling, standard rejection sampling always rejects and resamples the complete sample. We compare the performance of standard and hierarchical rejection sampling with and without additional graph pruning (described in Section 2.2.4.2). All results shown below are averages over 100 runs on a single core of an Intel Xeon E5-1620 CPU at 3.6 GHz. Some graphs also show standard deviations as vertical bars.

Figure 33 shows the convergence of the RRT* algorithm for the three example problems. While informed standard rejection sampling improves over uninformed sampling, hierarchical rejection sampling improves efficiency even further, leading to the planner finding lower-cost solution trajectories faster. This is the case with or without graph pruning. For the hammering and pick-and-place problems the improvement is significant, while for the lower-dimensional batting problem the improvement is small.

To measure the efficiency improvement, we compare the average computation time required by different sampling strategies to achieve the same solution cost. Figure 34 shows this. This figure is similar to Figure 33 but flips the two axes. Figure 35 shows the efficiency improvement of hierarchical rejection sampling. The efficiency improvement is the factor by which the computation time of standard rejection sampling is larger than the one of hierarchical rejection sampling to achieve the same solution cost. This data is also shown in Table 7. The improvement grows larger as time goes by and the solution cost shrinks, since more and more samples are rejected and a larger and larger fraction of time is spent sampling. Hierarchical rejection sampling improves efficiency by up to two orders of magnitude.

The following figures give additional insight into why hierarchical rejection sampling improves performance. Figure 36 shows the percentage of samples that get accepted by standard rejection sampling. For the hammering problem more than 99.99 % of samples get rejected. This leads to the planner spending most of its computation time on rejecting samples as Figure 37 shows. Hierarchical rejection sampling reduces the time required for finding valid samples. Because hierarchical rejection sampling spends less time sampling, the planner can generate a lot more samples in the same amount of time. Table 8 lists the number of samples generated within 60 seconds by the planning algorithm with different sampling strategies with and without graph pruning. For hierarchical rejection sampling the table lists the number of implicit samples, i.e. the number of samples that standard rejection sampling would have to generate in order to accept the same number of samples. This gives a measure on how densely the informed subset is covered with samples. The density of samples is directly correlated to the quality of the solution trajectory. A denser sample distribution results in a lower-cost solution. The RRT* with hierarchical rejection sampling generates up to 35 billion implicit samples in 60 seconds. I.e. the samples fill the informed subset as densely as if we had sampled 35 billion samples in the whole state space.

7.3.3 Optimality

The hammering and batting problems involve non-zero goal velocities and, thus, cannot be solved by a geometric planner. The pick-and-place problem, however, can be solved by a geometric planner. In this section we show that, unlike acceleration-limited planning, combining a geometric planner with acceleration-limited post-processing does not always result in an optimal solution. Table 6 compares the solution cost obtained by different planners. The table shows the average solution cost (and standard deviation) obtained by running the acceleration-limited planner 100 times for 60 seconds. We compare this to a geometric planner combined with different post-processing strategies: First, following the geometric path exactly, coming to a complete stop at

Table 6: Pick and place: Unlike geometric planning with post-processing, acceleration-limited planning converges to the optimal solution.

algorithm	solution cost	
geometric RRT*	9.88 s	
+ circular blends + path following (Chapter 3)	7.08 s	
+ acceleration-limited shortcuts [18]	5.54 s	
acceleration-limited RRT* (this chapter)	4.93 s	(± 0.018 s)

every waypoint. Second, using the method presented in Chapter 3 of first adding circular blends around waypoints and then finding the optimal time-parameterization along the geometric path. Third, using the acceleration-limited shortcut method proposed by Hauser and Ng-Thow-Hing [18]. We ran the geometric algorithms only once but for a long time (10 minutes) to make sure they converged. However, the solution costs they converge to are much higher than the cost of the solution found by the acceleration-limited planner.

Figure 32 shows the trajectory generated by the geometric planner with acceleration-limited shortcuts. It significantly differs from the one generated by the acceleration-limited planner in Figure 31. The geometric planner chooses to rotate the first joint in the opposite direction than the acceleration-limited planner. While the acceleration-limited post-processing can smooth the path, it cannot fundamentally change the solution, will not change the rotation direction of the first joint and, thus, will not find the optimal solution.

Table 7: Average computation time required to reach given solution costs

task	solution cost	without graph pruning: uninformed sampling	without graph pruning: standard rejection sampling	without graph pruning: HRS	without graph pruning: eff. gain factor	with graph pruning: standard rejection sampling	with graph pruning: HRS	with graph pruning: eff. gain factor
hammering	7.0 s	22.4 s	0.9 s	1.0 s	0.9	0.5 s	0.6 s	0.9
	6.5 s	176.9 s	1.3 s	1.5 s	0.9	0.7 s	0.8 s	0.9
	6.0 s		2.6 s	2.3 s	1.1	1.6 s	1.4 s	1.2
	5.5 s		23.3 s	5.3 s	4.4	21.4 s	3.4 s	6.4
	5.2 s		216.2 s	12.5 s	17.3	206.6 s	9.9 s	20.9
	5.0 s		1793.2 s	40.4 s	44.4	1806.1 s	33.9 s	53.2
batting	5.5 s	6.06 s	0.45 s	0.46 s	1.0	0.30 s	0.32 s	1.0
	5.0 s	77.48 s	0.58 s	0.59 s	1.0	0.38 s	0.38 s	1.0
	4.5 s		1.18 s	0.93 s	1.3	0.77 s	0.57 s	1.4
	4.26 s		36.18 s	7.73 s	4.7	31.51 s	5.05 s	6.2
pick and place	8.0 s	95.3 s	8.5 s	10.2 s	0.8	3.1 s	2.3 s	1.3
	7.0 s		14.6 s	16.0 s	0.9	3.9 s	3.2 s	1.2
	6.0 s		26.9 s	24.4 s	1.1	7.0 s	4.0 s	1.7
	5.5 s		67.7 s	37.6 s	1.8	33.3 s	4.8 s	6.9
	5.2 s					284.5 s	6.8 s	42.1
	5.0 s					2828.8 s	17.5 s	161.4

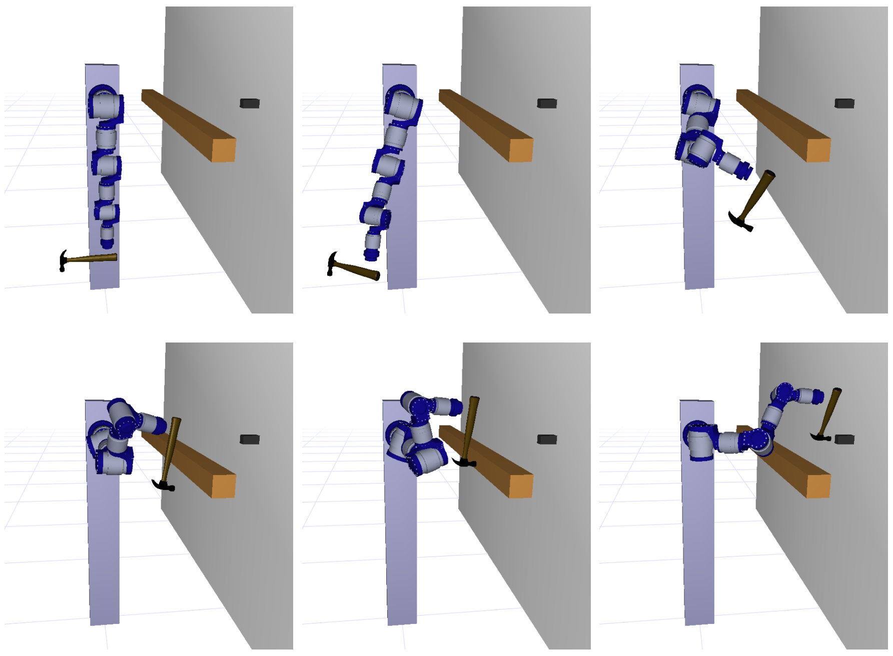
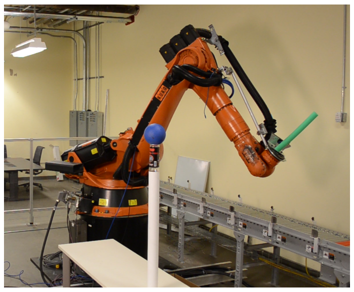
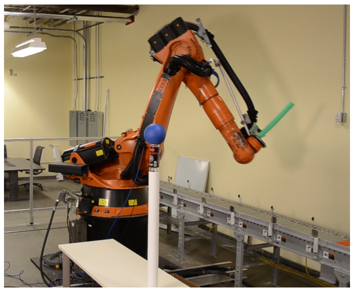
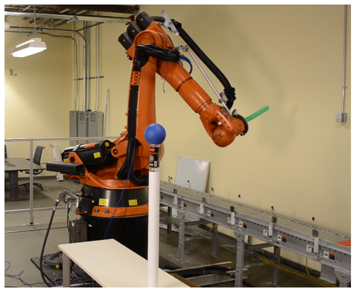
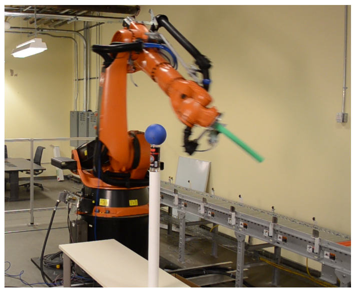


Figure 28: Hammering: Solution trajectory after 60 s of computation time


(a) 0.0 s


(b) 0.7 s


(c) 1.4 s


(d) 2.1 s

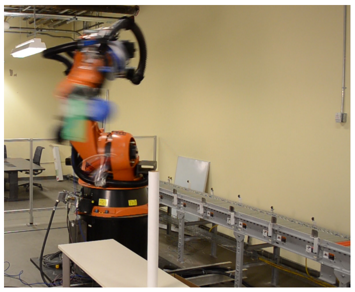
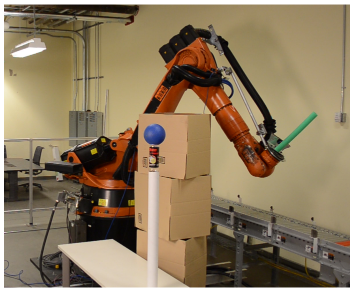
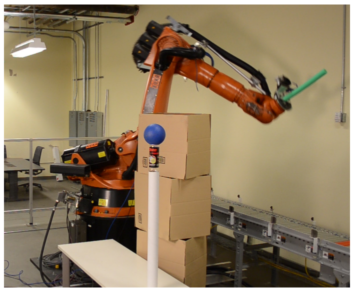
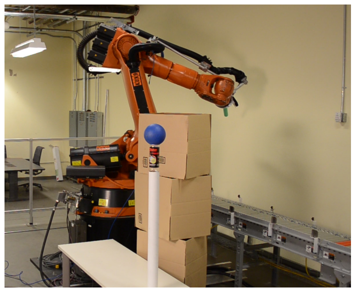
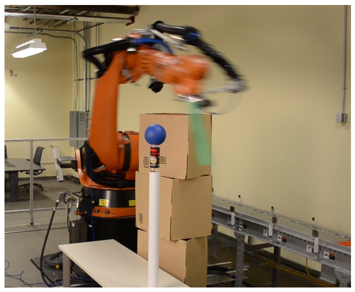

(e) 2.5 s

Figure 29: Batting without obstacle: Solution trajectory after 60 s of computation time


(a) 0.0 s


(b) 1.4 s


(c) 2.8 s


(d) 4.2 s

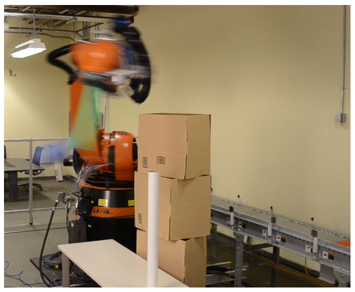

(e) 4.4 s

Figure 30: Batting: Solution trajectory after 60 s of computation time

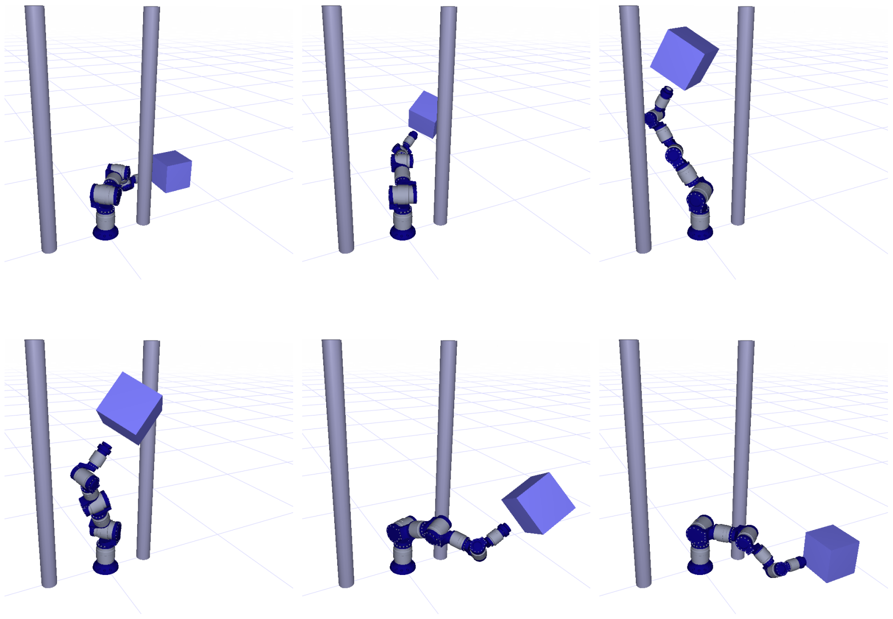


Figure 31: Pick and place: Solution trajectory after 60s of computation time

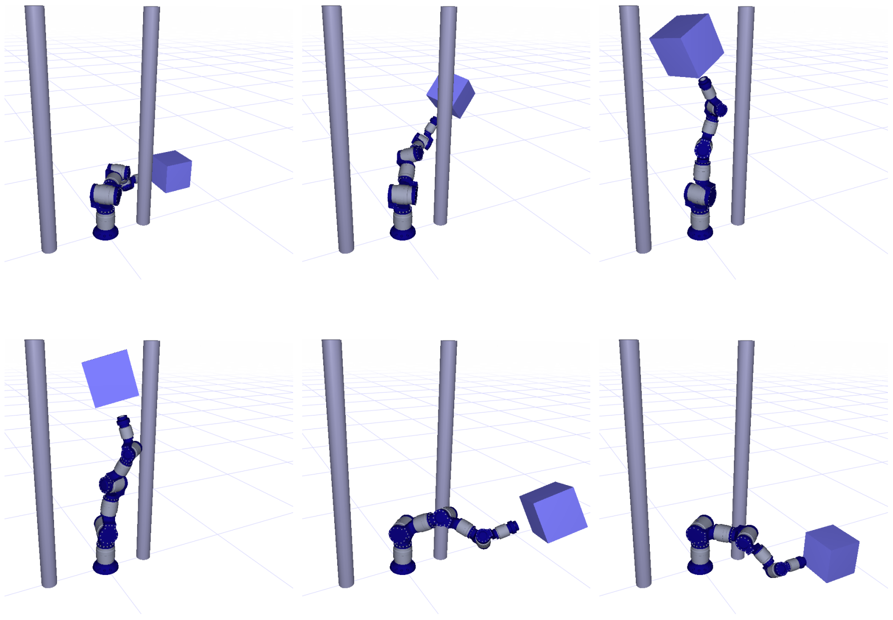
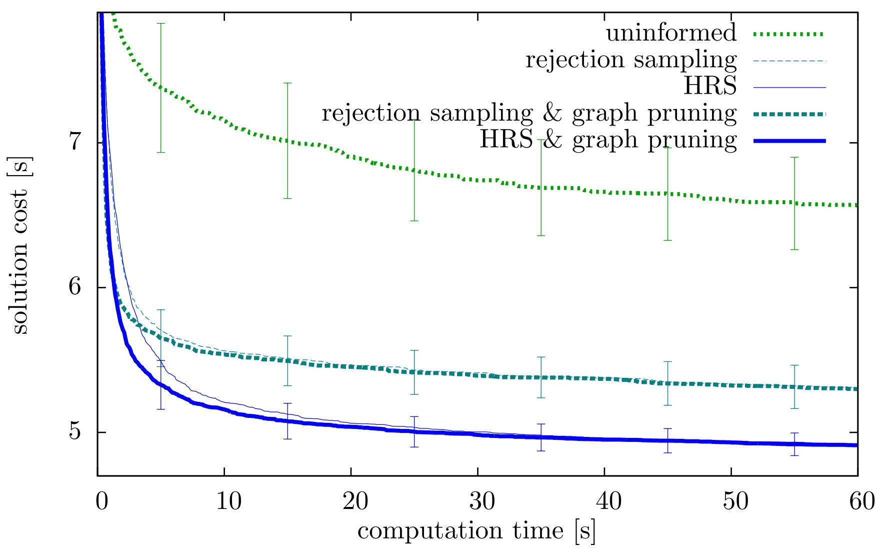
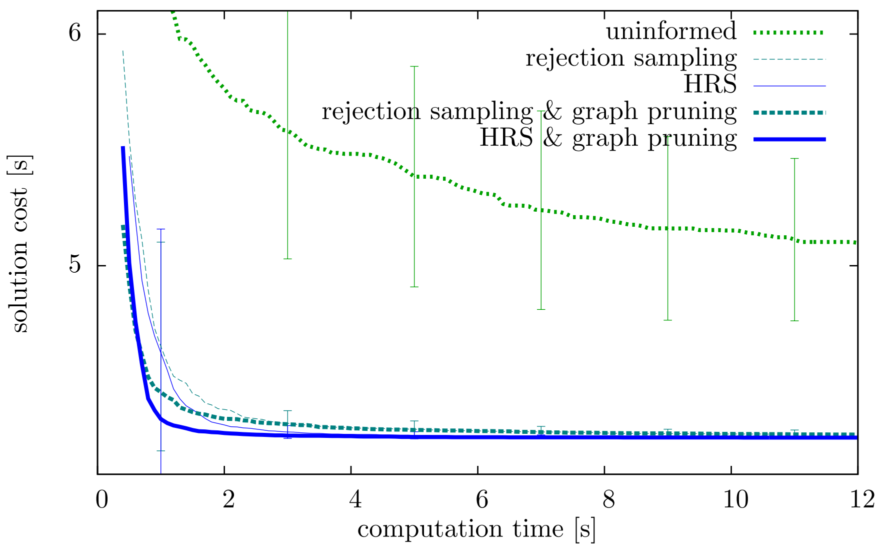


Figure 32: Pick and place: Solution trajectory obtained by combining a geometric planner with acceleration-limited shortcutting

(a) Hammering



(b) Batting



(c) Pick and place

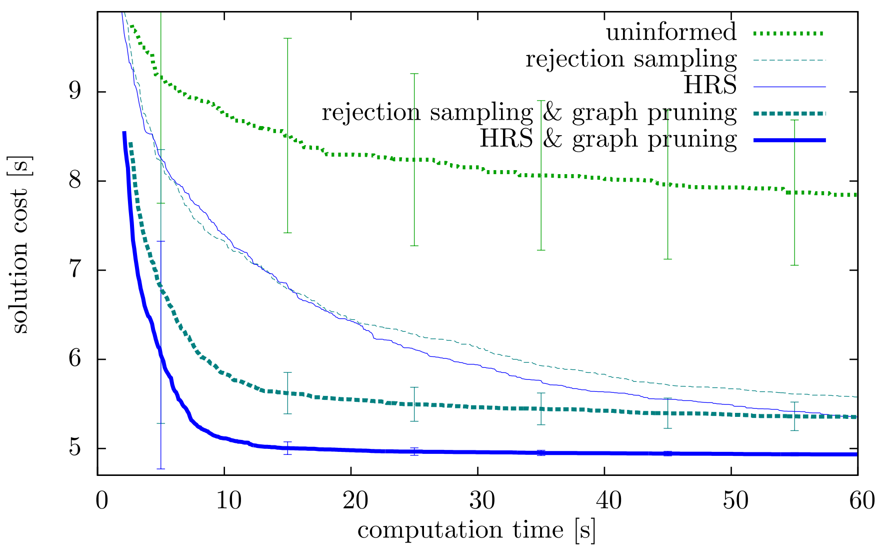
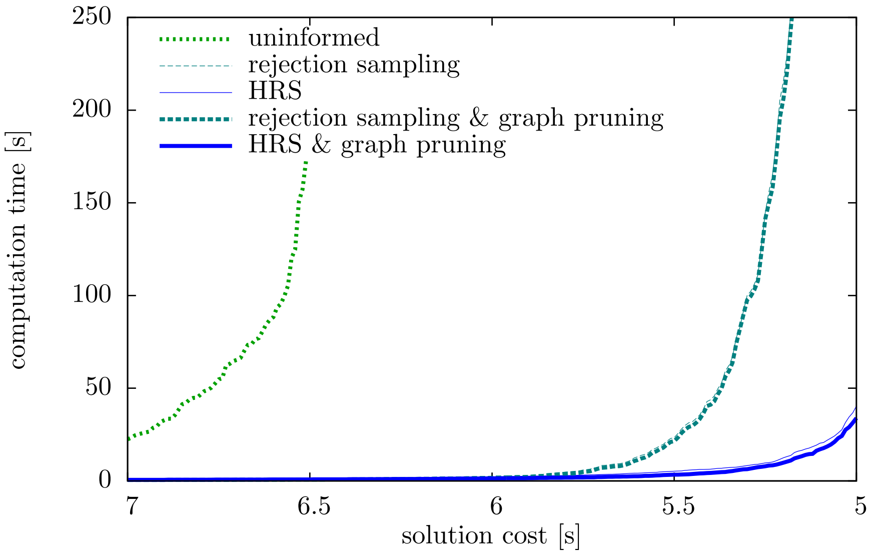
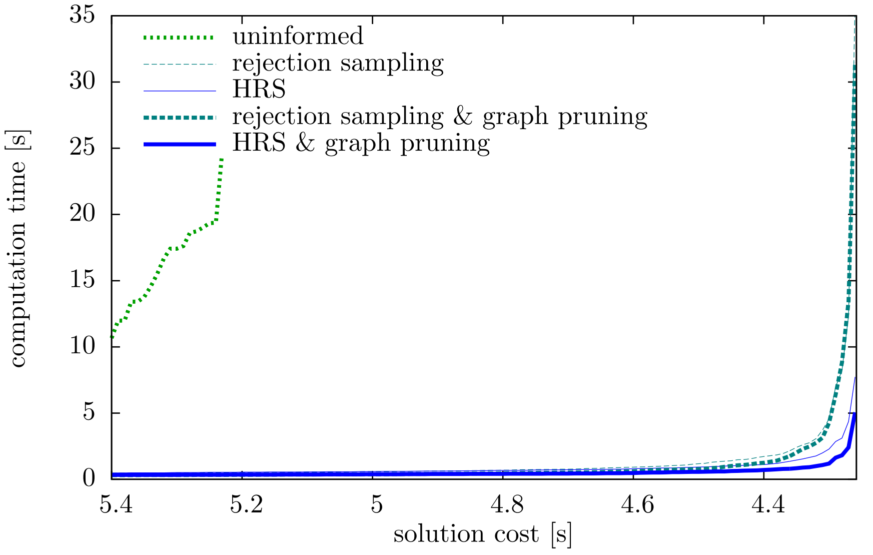


Figure 33: Convergence for different sampling strategies (lower is better)

(a) Hammering



(b) Batting



(c) Pick and place

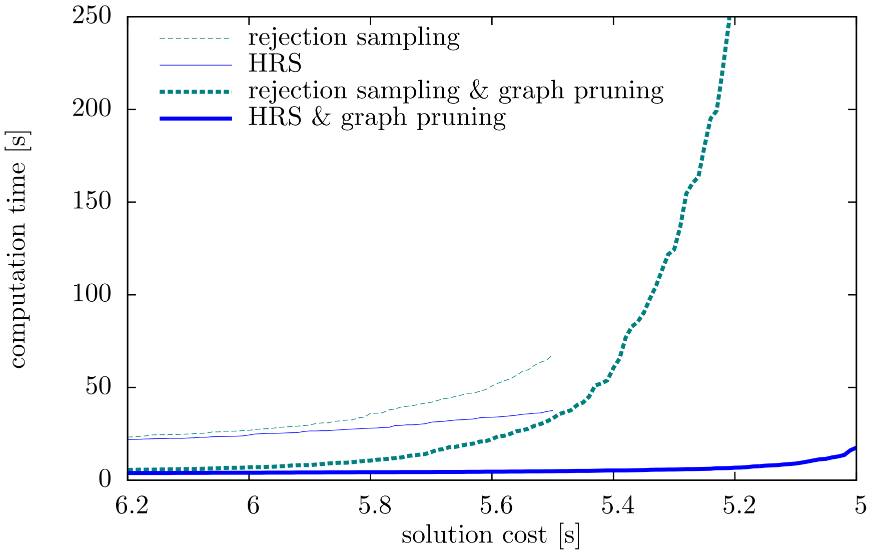
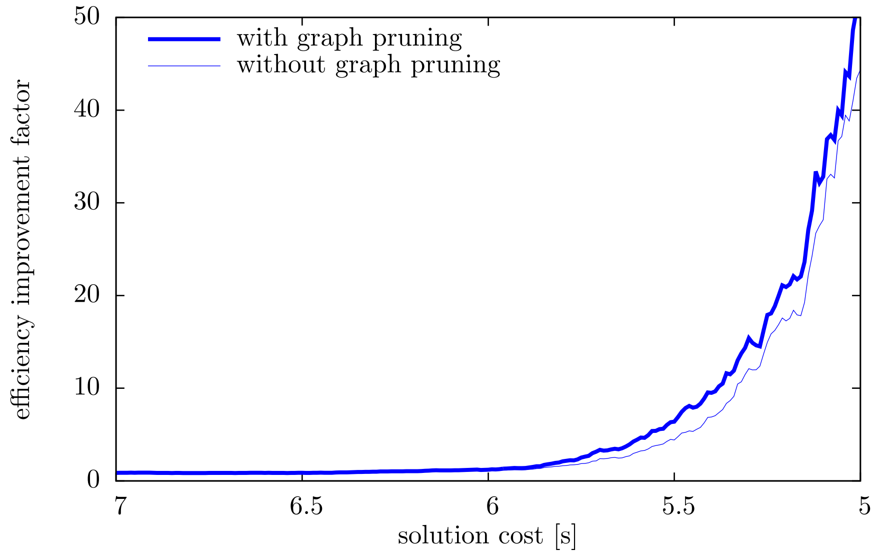
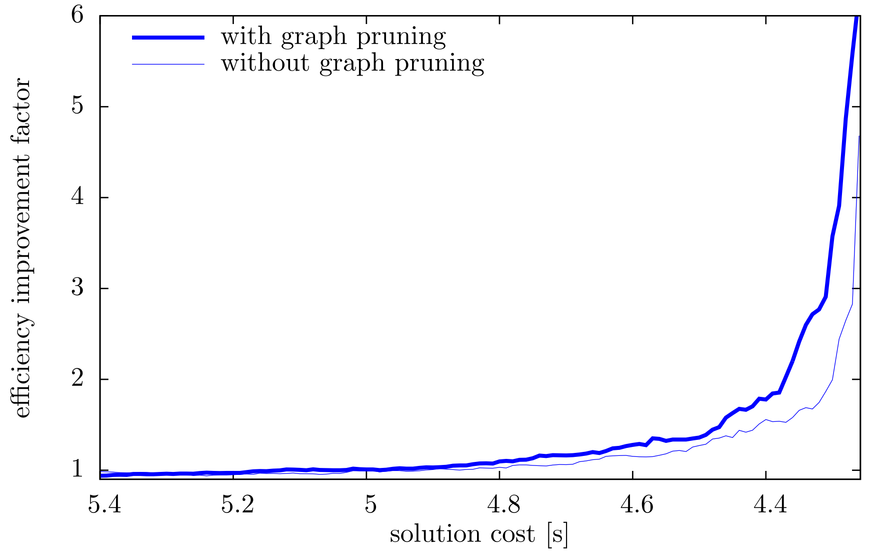


Figure 34: Computation time required to reach solution cost (lower is better)

(a) Hammering



(b) Batting



(c) Pick and place

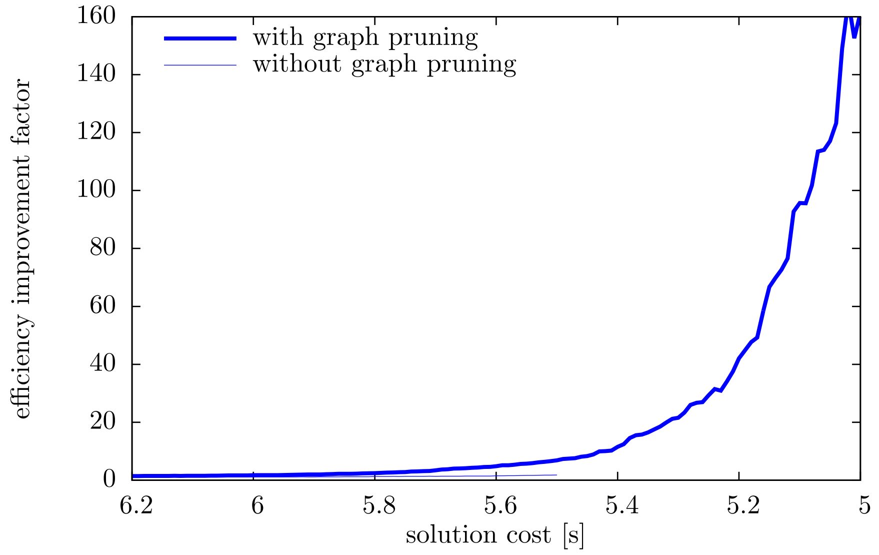
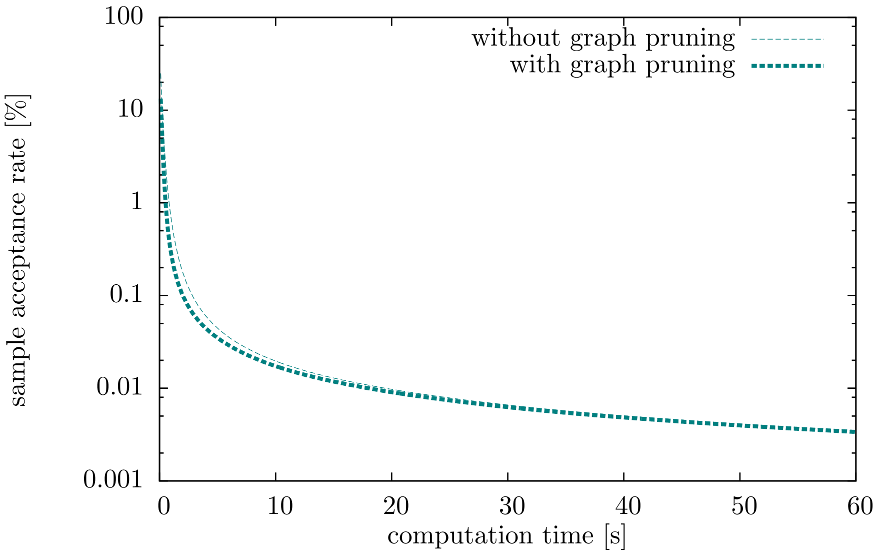
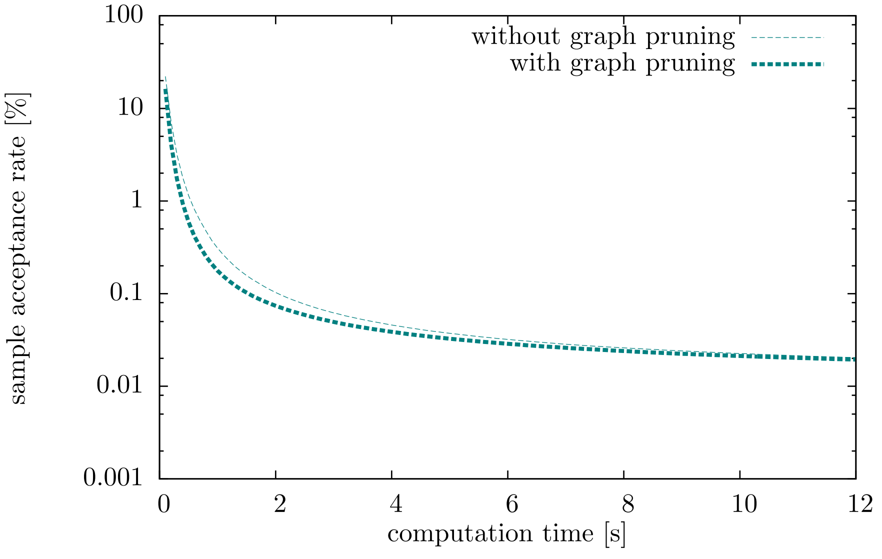


Figure 35: Efficiency gain of HRS

(a) Hammering



(b) Batting



(c) Pick and place

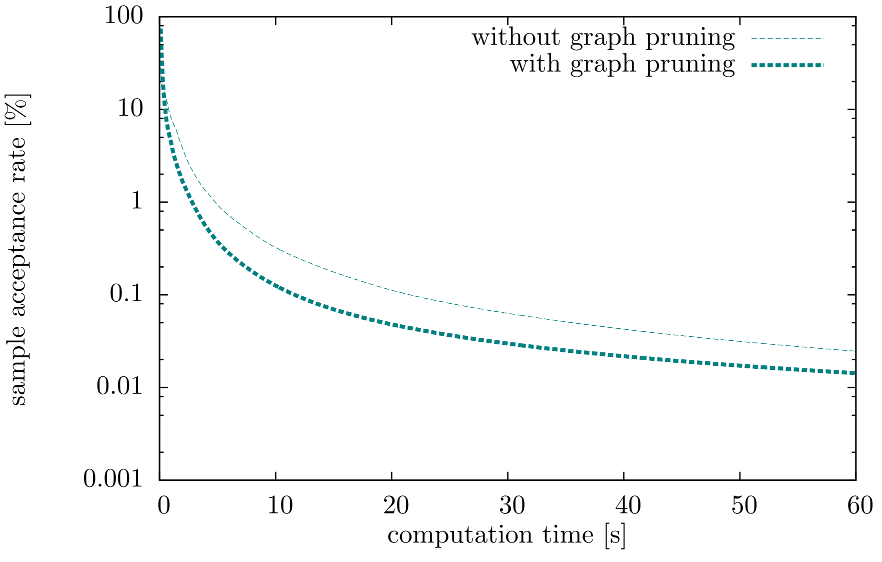
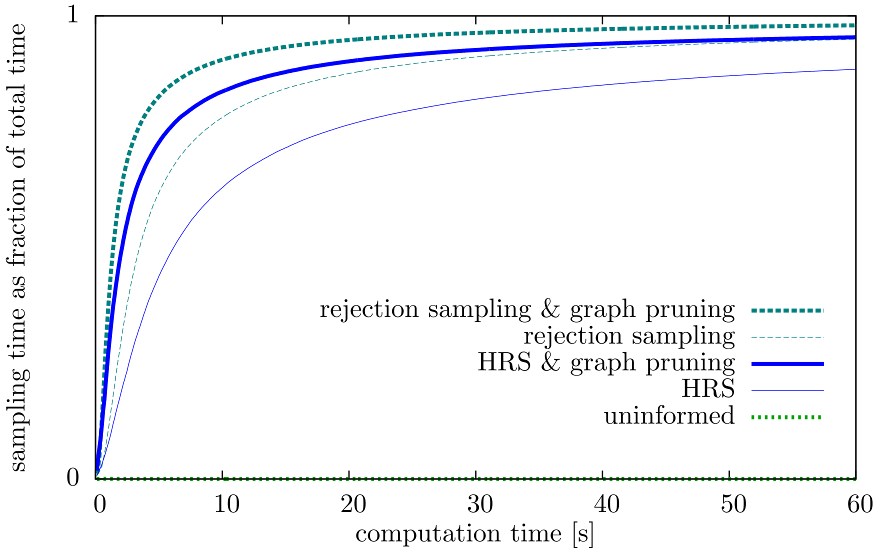
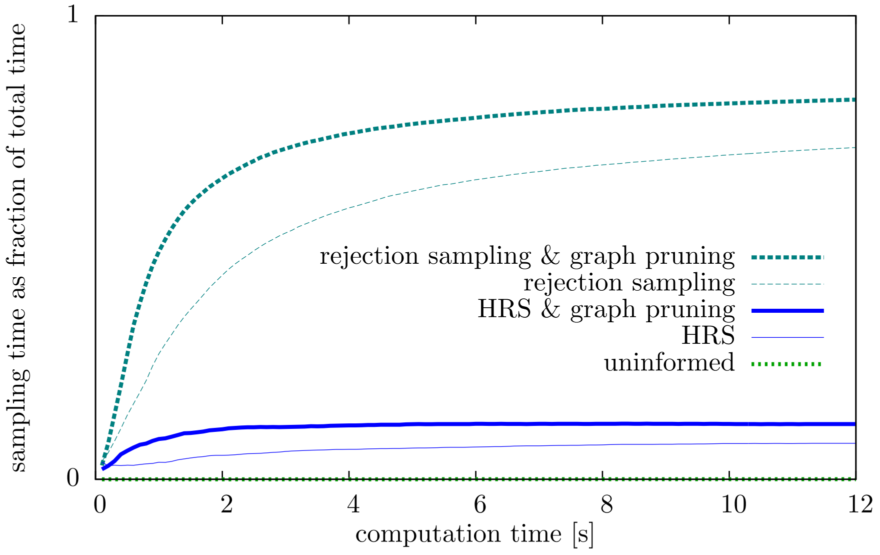


Figure 36: Sample acceptance rate of standard rejection sampling

(a) Hammering



(b) Batting



(c) Pick and place

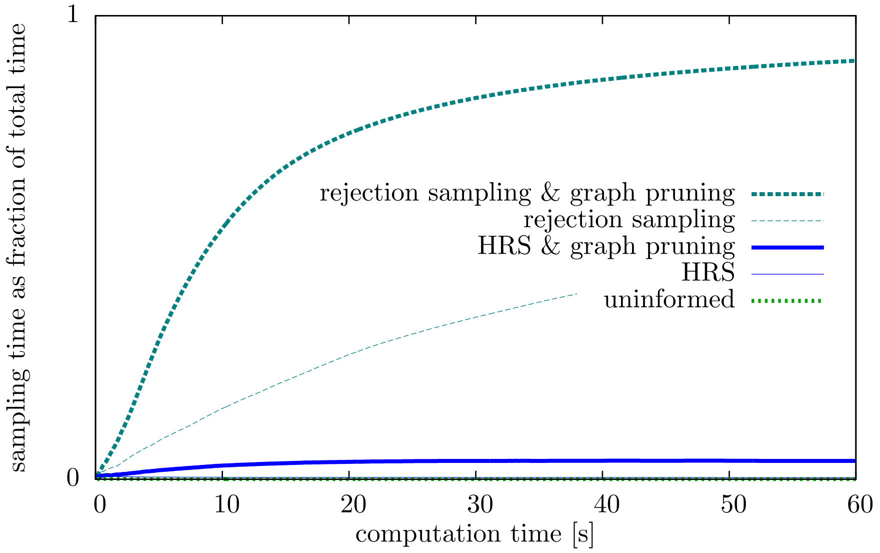


Figure 37: Time spent sampling as fraction of total computation time

Table 8: Average number of (implicit) samples generated within 60 s

task	algorithm	(implicit) samples
hammering	uninformed	7,340
	rejection sampling	11,600,000
	rejection sampling & graph pruning	12,100,000
	HRS	1,230,000,000
	HRS & graph pruning	1,370,000,000
batting	uninformed	15,200
	rejection sampling	23,700,000
	rejection sampling & graph pruning	25,600,000
	HRS	103,000,000
	HRS & graph pruning	144,000,000
pick and place	uninformed	26,100
	rejection sampling	40,400,000
	rejection sampling & graph pruning	72,600,000
	HRS	1,180,000,000
	HRS & graph pruning	35,700,000,000

CHAPTER VIII

CONCLUSION

8.1 Contributions

8.1.1 Acceleration-Limited Planning

We presented acceleration-limited planning as a middle ground between pure geometric planning and planning with full dynamics. Acceleration-limited planning considers robot dynamics more closely than pure geometric planning (which does not at all). This allows to solve problems involving non-zero start or goal velocities, such as the hammering task we presented or online replanning. In addition, even if start and goal velocities are zero, acceleration-limited planning has some advantages over geometric planning with post-processing for some problems. Unlike geometric planning with post-processing, acceleration-limited planning is asymptotically optimal in regard to the system with acceleration limits. In a few cases this results in acceleration-limited planning finding better solutions than geometric planning combined with post-processing. We presented one such case.

At the same time, acceleration-limited planning is a less powerful problem formulation than planning with full dynamics. For example, it cannot find a swing-up trajectory for a weakly actuated simple pendulum. Also, an optimal trajectory considering acceleration limits is not optimal when considering torque limits. However, unlike planning with full dynamics, acceleration-limited planning can be solved systematically. Unlike any existing planner for arbitrary dynamics, our acceleration-limited planner is efficient, parameter-free, probabilistically complete and asymptotically optimal. In contrast, as we showed, the standard kinodynamic RRT for planning with full dynamics is not efficient and in general not probabilistically complete. While optimization-based methods are very powerful, they require parameter tuning.

We used the hammering task, which involves non-zero goal velocities, as an example problem to show the advantage of acceleration-limited planning. As we showed in Chapter 6, neither a geometric planner nor existing sampling-based planners for planning with full dynamics can solve this problem. On one side, geometric planning is unable to consider the required goal velocity. On the other side, the standard kinodynamic RRT with a Euclidean distance metric, which can consider arbitrary dynamics, is too inefficient to solve the problem in a reasonable amount of time. Our acceleration-limited planner, in contrast, can find a feasible solution for the hammering task in less than 100 ms. In Chapter 7 we also presented an asymptotically optimal planner based on the RRT* algorithm that converges to the optimal solution and provides good-quality solutions in less than one minute. We presented one example task where this asymptotically optimal planner provides a benefit even if start and goal velocities are zero. While existing approaches that combine a geometric planner with post-processing to satisfy acceleration limits might yield near-optimal solutions in most cases, we demonstrated that for this example task they do not. We also applied our planner to a real industrial robot arm to demonstrate that our approach leads to smooth trajectories that can be directly executed on a real robot.

8.1.2 Contributions to General Kinodynamic Planning

In addition to proposing acceleration-limited planning, we made contributions that are of interest not only to acceleration-limited planning but to kinodynamic planning in general. For time-optimal path following in Chapter 3, we pointed out the importance of numerical considerations as well as the problem of choosing the optimal acceleration at one of the switching point types. These findings are important even if using torque limits instead of acceleration limits. Our proof of the incompleteness of the standard kinodynamic RRT in Chapter 4 will hopefully spur further research into the requirements to guarantee probabilistic completeness and lead to a better understanding how to apply sampling-based planners to arbitrary dynamic systems. Hierarchical rejection sampling presented in Chapter 7 can improve the efficiency of any asymptotically optimal sampling-based planner for problems with differential constraints. In our experiments that improvement was up to two orders of magnitude.

8.2 Future Work and Open Problems

In addition to our contributions, further improvements to the double-integrator minimum-time steering method can be made as described in Sections 8.2.1 and 8.2.2. The advantage of acceleration-limited planning could be further demonstrated by applying it to online replanning as mentioned in Section 8.2.3. Section 8.2.4 points out a limitation of our current approach. We consider acceleration-limited planning as a step towards planning with full dynamics. Sections 8.2.5 and 8.2.6 present open problems that we think need to be solved to achieve efficient and optimal sampling-based planning with full dynamics.

8.2.1 Simpler Steering Method for Double Integrators

The steering method for solving the double-integrator minimum-time problem can be simplified, resulting in faster, more compact and more robust code. Since asymptotically optimal planners spend a lot of time calculating distances using the steering method, this can significantly speed up the planning process.

Our work and all the previous work find the minimum time by setting up a system of two equations with the duration of the two min and max acceleration segments t_{a1} and t_{a2} as unknown variables. This results in Eq. 66 and 67. This requires solving a quadratic equation of the form

$$ax^2 + bx + c = 0 \tag{88}$$

However, by setting up the equations differently, the problem can be solved more easily. Instead of using the duration of the two minimum and maximum acceleration segments t_{a1} and t_{a2}, one can use the extremal velocity of the trajectory as unknown variable. From the extremal velocity and the acceleration limits it is easy to calculate the duration of the two segments. Setting up the problem in this way leads to a quadratic equation of the form

$$ax^2 + c = 0 \tag{89}$$

This can be solved by just taking a square root. This is easier than solving the full quadratic equation above.

8.2.2 Consider Joint Limits within Steering Method

Like Hauser et al. [18] our steering method currently does not consider joint limits. Instead, the collision checker rejects trajectories that exceed joint limits. However, this might result in a connection being rejected, even though a trajectory satisfying joint limits would have existed. Joint limits can be directly incorporated into the steering method, minimizing acceleration with the additional constraint of the joint limits. This might lead to a small efficiency improvement.

Kröger et al. [29] choose the trajectory such that it always satisfies joint limits if possible. However, while we and [18] minimize acceleration when choosing trajectories for each joint to reach the goal at the determined minimum time, [29] constrains the trajectories to only use extremal and zero accelerations. This leads to larger-than-necessary accelerations.

8.2.3 Online Replanning

One application of acceleration-limited planning that we have used as a motivational example but have not explored is online replanning. Acceleration-limited planning is very suitable for online replanning, because online replanning involves a non-zero start state, which geometric planning cannot deal with. At the same time, a steering method is available for acceleration-limited planning. This results in efficient algorithms, which is required to achieve the short reaction times necessary for online replanning.

8.2.4 Task Constraints

Our work does not consider task constraints. This is a disadvantage compared with optimization-based approaches (see Section 2.3), which can easily consider both actuator limitations and task constraints. Whether there is a way to incorporate task constraints is an open problem.

8.2.5 Connection Threshold for Asymptotically Optimal Planners

As described in Section 2.2.2.1, it is not obvious how to choose the connection threshold of asymptotically optimal sampling-based planners for problems with differential constraints such that optimality is guaranteed. More work is necessary on this problem. We avoided this problem in our work by choosing an infinite connection threshold, which guarantees optimality but might have made our planner less efficient.

In addition, for problems with or without differential constraints, it is unknown how to choose the connection threshold optimally. Existing work only deals with finding the lowest possible threshold that still guarantees asymptotic optimality. While choosing the connection threshold as low as possible is a reasonable choice, it has not been studied whether this actually leads to the fastest convergence in all cases or whether a higher connection threshold might sometimes be better.

8.2.6 Distance Functions and Probabilistic Completeness of RRTs

Our work focuses on a problem for which a steering method is available. If a steering method is available, we have access to an high-quality distance function, i.e. the cost to move between two states without obstacles. However, for many systems a steering method is not available. Instead, kinodynamic RRTs make use of an incremental simulator, simulating the systems dynamics forward for multiple control input trajectories and then selecting one of them. This selection is usually made based on which control input results in the new state being closest to the sampled state. Closeness is measured by a user-provided distance function. For systems without a steering method, good distance functions usually do not exist. Instead, a Euclidean distance metric is often used. There have been attempts at using better-informed distance functions, for example using the cost of the optimal trajectory for a linearized system as the distance function for a nonlinear system [14, 45, 17, 56] (see Section 2.2.3). However, there is little understanding in what constitutes a good distance function other than some closeness to the real optimal cost to move between two states. There is no measure to decide which of two distance functions is better other than testing them by running an RRT on an example problem. A better understanding of what properties of a distance function cause good performance of an RRT could help find better distance functions. For example, one may wonder whether it is helpful for a distance function to give an upper or lower bound on the real cost.

Not only do we not know what constitutes a good distance function, as we have shown in Chapter 4, we do not even know what requirements a distance function needs to satisfy to guarantee probabilistic completeness of the standard kinodynamic RRT. Studying the requirements for probabilistic completeness could provide a foundation for understanding what constitutes a good distance function.

8.3 Final Remarks

As described in Sections 8.2.5 and 8.2.6, there are still a lot of open problems to solve on the way to achieve efficient and optimal motion planning considering the full dynamics of robot manipulators and other systems. In the meantime, acceleration-limited planning provides a good approximation for many robots, e. g. industrial manipulators, that can be efficiently solved today.

REFERENCES

[1] Akgun, B. and Stilman, M., "Sampling heuristics for optimal motion planning in high dimensions," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2011.

[2] Arslan, O. and Tsiotras, P., "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *IEEE Int. Conf. on Robotics and Automation*, 2013.

[3] Bhatia, A. and Frazzoli, E., "Incremental search methods for reachability analysis of continuous and hybrid systems," in *Hybrid Systems: Computation and Control* (Alur, R. and Pappas, G., eds.), vol. 2993 of *Lecture Notes in Computer Science*, pp. 142–156, Springer, 2004.

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

[5] Box, G. E. and Draper, N. R., *Empirical model-building and response surfaces.* John Wiley & Sons, 1987.

[6] Cheng, P. and LaValle, S. M., "Resolution complete rapidly-exploring random trees," in *IEEE Int. Conf. on Robotics and Automation*, 2002.

[7] Craig, J. J., *Introduction to Robotics: Mechanics and Control (3rd Edition).* Prentice Hall, 2004.

[8] Dubins, L. E., "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[9] Esposito, J. M., Kim, J., and Kumar, V., "Adaptive RRTs for validating hybrid robotic control systems," in *Algorithmic Foundations of Robotics VI*, pp. 107–121, Springer, 2005.

[10] Ferguson, D. and Stentz, A., "Anytime RRTs," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006.

[11] Frazzoli, E., Dahleh, M. A., and Feron, E., "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[12] Gammell, J., Srinivasa, S., and Barfoot, T., "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2014.

[13] Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D., "BIT*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs," *CoRR*, vol. abs/1405.5848, 2014.

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

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

[16] Goerzen, C., Kong, Z., and Mettler, B., "A survey of motion planning algorithms from the perspective of autonomous UAV guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1-4, pp. 65–100, 2010.

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

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

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

[20] Janson, L. and Pavone, M., "Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions," in *Int. Symposium on Robotics Research*, 2013.

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

[22] Kalisiak, M. and van de Panne, M., "RRT-blossom: RRT with a local flood-fill behavior," in *IEEE Int. Conf. on Robotics and Automation*, 2006.

[23] Karaman, S. and Frazzoli, E., "Optimal kinodynamic motion planning using incremental sampling-based methods," in *IEEE Conf. on Decision and Control*, 2010.

[24] Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., and Teller, S., "Anytime motion planning using the RRT*," in *IEEE Int. Conf. on Robotics and Automation*, 2011.

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

[26] Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M., "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

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

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

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

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

[31] Kunz, T., Kingston, P., Stilman, M., and Egerstedt, M., "Dynamic chess: Strategic planning for robot motion," in *IEEE Int. Conf. on Robotics and Automation*, 2011.

[32] Kunz, T., Reiser, U., Stilman, M., and Verl, A., "Real-time path planning for a robot arm in changing environments," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2010.

[33] Kunz, T. and Stilman, M., "Manipulation planning with soft task constraints," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2012.

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

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

[36] Kunz, T. and Stilman, M., "Probabilistically complete kinodynamic planning for robot manipulators with acceleration limits," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2014.

[37] Kurniawati, H. and Hsu, D., "Workspace importance sampling for probabilistic roadmap planning," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2004.

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

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

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

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

[42] LaValle, S. M., *Planning algorithms.* Cambridge University Press, 2006.

[43] Leven, P. and Hutchinson, S., "Using manipulability to bias sampling during the construction of probabilistic roadmaps," *IEEE Trans. on Robotics and Automation*, vol. 19, no. 6, pp. 1020–1026, 2003.

[44] Nasir, J., Islam, F., Malik, U., Ayaz, Y., Hasan, O., Khan, M., and Muhammad, M. S., "RRT*-SMART: A rapid convergence implementation of RRT*," *Int. Journal of Advanced Robotic Systems*, vol. 10, 2013.

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

[46] Petti, S. and Fraichard, T., "Safe motion planning in dynamic environments," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2005.

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

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

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

[50] Shiller, Z. and Lu, H. H., “Computation of path constrained time optimal motions with dynamic singularities,” *Journal of dynamic systems, measurement, and control*, vol. 114, pp. 34–40, 1992.

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

[52] Siciliano, B., Sciavicco, L., and Villani, L., *Robotics: modelling, planning and control*. Springer, 2009.

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

[54] Teeyapan, K., Wang, J., Kunz, T., and Stilman, M., “Robot limbo: Optimized planning and control for dynamically stable robots under vertical obstacles,” in *IEEE Int. Conf. on Robotics and Automation*, 2010.

[55] van den Berg, J. P. and Overmars, M. H., “Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners,” *The Int. Journal of Robotics Research*, vol. 24, no. 12, pp. 1055–1071, 2005.

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

[57] Zlajpah, L., “On time optimal path control of manipulators with bounded joint velocities and torques,” in *IEEE Int. Conf. on Robotics and Automation*, 1996.