Sampling-Based Planning for Hybrid Systems via Reachability Guidance and Policy Approximation

by

Albert Wu

S.B., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author .....
Department of Electrical Engineering and Computer Science
May 12, 2020

Certified by .....
Russ Tedrake
Toyota Professor of EECS, Aero/Astro, MechE.
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Sampling-Based Planning for Hybrid Systems via Reachability Guidance and Policy Approximation

by

Albert Wu

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2020, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Hybrid systems arise in many robotic problems such as manipulation and locomotion, and efficient motion planning is critical for obtaining practical implementations of these systems. However, due to the complexity associated with multiple contact modes, existing motion planning techniques have limited success with hybrid systems. This thesis investigates the entire sampling-based motion planning stack with an emphasis on hybrid system applications. First, a reachability-based variant of the rapidly-exploring random tree algorithm (RRT) named R3T is introduced. R3T is suitable for planning in nonlinear and hybrid systems. It is probabilistically complete in kinodynamic systems and asymptotically optimal through rewiring. The advantages of R3T are demonstrated with case studies on nonlinear and contact-rich robotic systems. Efficient mathematical tools for solving the "nearest-polytope problem", as motivated by control application including R3T, are then discussed. The tools demonstrated logarithmic empirical complexity with respect to the dataset size in examples motivated by approximate explicit model predictive control and R3T. Finally, a novel framework for generating robot manipulation policies is proposed. In this framework, a formulation of RRT* for manipulation first explores the state space and discover plans to a given goal state. A neural network then learns a manipulation policy from the RRT* plans. The manipulation plans from RRT* are verified via playback in simulation, and training results of the neural network suggests the validity of this approach. The goal of this thesis is to provide algorithmic frameworks for developing applications requiring hybrid system motion planning.

Thesis Supervisor: Russ Tedrake
Title: Toyota Professor of EECS, Aero/Astro, MechE.

Acknowledgments

First and foremost, I would like to thank the amazing collaborators that made this thesis possible. Thanks to Sadra Sadraddini for introducing me to many important mathematical tools and contemporary research topics on polytopes. This background knowledge, along with great discussions and collaboration with Sadra, were crucial for bringing Chapters 2 and 3 to life.

Thanks to Hongkai Dai for his continued collaboration and support on the development of Chapter 4. Hongkai's inputs shaped much of the chapter and laid the groundwork for the ongoing motion planning research using the planar gripper system. I would also like to thank Rick Cory for developing the hardware and simulation systems on which Chapter 4 is based.

I would like to thank the members of the MIT Robot Locomotion Group for their continued feedback and insightful suggestions on the project. The group largely defined my M.Eng. experience, and it could not have be better.

A very special thank you goes to my advisor, Russ Tedrake. Russ have always provided me with the support and freedom to pursue projects that excite me the most. Meanwhile, his guidance and feedback helped me pinpoint what I should tackle during the limited time I have. Pointers from Russ was what made completing all the research work in this thesis within this time frame possible.

Finally, thank you to all of my wonderful friends for creating an amazing MIT experience, and my family for always being there for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Efficient motion planning is key to performing complicated tasks with robots. Whether it is avoiding obstacles, manipulating objects, or performing dynamic locomotion, motion planning is the backbone of such applications. However, many limitations exist with the state-of-the-art techniques today, especially in systems with complex hybrid dynamics. This thesis is an investigation on the entire motion planning stack with a focus on sampling-based planning. Topics covered include a novel sampling-based planning algorithm, mathematical tools motivated by applications such as the aforementioned algorithm, and a manipulation planning paradigm leveraging sampling-based planning and learning techniques. The ultimate objective of this thesis is to address the task of hybrid system motion planning, with an emphasis on robot manipulation.

1.1 Sampling-based Motion Planners

Sampling-based motion planning algorithms such as probabilistic road-maps (PRMs) [19] and rapidly-exploring random trees (RRTs) [23, 26, 17, 9] have been proven powerful in a broad range of planning problems. As opposed to optimization-based trajectory synthesis, where all the system dynamics and environment specifications are encoded in the constraints of an optimization problem, sampling-based methods are simpler to implement and sometimes faster in finding feasible trajectories in highly

cluttered environments.

Nevertheless, a number of limitations exist in these methods. The bases of RRTs are rapid exploration of the state space and connection of new states to explored states. When kinodynamic constraints are present, one needs to solve the expensive two point boundary value problem of finding an admissible trajectory to perform connection. For linear systems, the problem is manageable and implementations exist [50, 12]. A widely-adopted alternative for general systems is to simulate trajectories forward, then expand the explored states with the nearest produced point to the sample state [26, 18]. However, this approach is not probabilistically complete [24]. Existing RRT approaches also tend to perform poorly in hybrid systems. With hybrid systems, the extension strategy choice is not obvious, and a distance metric is often unavailable [4, 28, 40]. Hybrid systems will be discussed in detail in Section 1.2.

1.2 Hybrid System Motion Planning

1.2.1 Hybrid System Definition

Hybrid system are characterized by the mixture of discrete and continuous dynamical behaviors. In continuous time, a general hybrid system can be described as

$$\dot{x} = f(x, u, \sigma), (x, u) \notin \mathbb{G}, \tag{1.1a}$$

$$(\sigma, x)^+ = r(x, u, \sigma), (x, u) \in \mathbb{G}, \tag{1.1b}$$

where $x \in X \subset \mathbb{R}^n$, is the continuous system state, $u \in U \subset \mathbb{R}^m$, is the control input, $\sigma \in \Sigma$ is the system mode such that Σ is a finite set, and $\mathbb{G}$ is the (zero-measured) set of guards where mode transitions happen. Using a time step $\tau \in \mathbb{R}_+$ and a time-integration scheme such as time-stepping ([44]), (1.1) can be formulated as the following discrete time form:

$$x^+ = F_i(x, u), (x, u) \in \mathbb{S}_i, \tag{1.2}$$

where $\mathbb{S}_i, i = 1, \cdots, N$, are interior-disjoint sets corresponding to N modes described by a number of inequalities $S_i(x, u) \leq 0$. The constraint $(x, u) \in X \times U$ is also included in each $\mathbb{S}_i, i = 1, \cdots, N$.

1.2.2 Hybrid System in Robotics

Hybrid system arise in many robotics applications. In particular, many robotic problems involving contact can be modeled as hybrid systems. The different contact modes can be captured naturally with hybrid dynamics. Two major classes of such problems are locomotion and manipulation.

Motion planning in hybrid robotic systems is challenging in general. Crucial maneuvers with particular mode sequences are hard to discover as the planning space grows combinatorially with the number of hybrid modes. This renders exhaustive search techniques impractical. Difficulties associated with choosing distance metrics in hybrid state spaces exacerbate this issue as heuristics are not readily available [40].

While locomotion and manipulation are both common hybrid systems in robotics, the two problems have distinct structures. In locomotion, there are typically fewer contact modes. For instance, if slipping is not considered, a walking quadruped has $2^4 = 16$ contact modes: each leg can be in or out of contact with the ground. Meanwhile, a three-finger robotic hand manipulating a square object with no slip on a plane has $5^3 = 125$ contact modes: each finger can make contact with any face of the object or not make contact. Moreover, locomotion problems are often periodic, which allow for efficient analysis and techniques such as return maps and limit cycle [46]. Manipulation, on the other hand, does not posses such a property. This thesis will therefore focus on manipulation as the primary application of interest.

1.3 Manipulation Planning

Robust robotic manipulation outside of controlled environments can enable robotics applications that are currently impossible, such as versatile autonomous assembly and disaster response. However, manipulating objects with robots remains one of

the most significant unsolved tasks in robotics today. The main challenges with robot manipulation are twofold. First, generating manipulation plans in contact-rich space requires exploring different mode sequences, which is difficult to perform online [38][14]. Moreover, manipulation plans generated by existing methods are brittle against environmental variations [20]. Traditional control and planning methods rely heavily on model accuracy and state estimation, yet contact dynamics are challenging to model and observe [20]. While some methods based on machine learning may circumvent the modeling challenges, learning models are tedious to train. Consequently, most robot manipulation applications have yet to evolve beyond controlled laboratory settings [20].

1.4 Contributions and Organization

This thesis presents a series of work with the ultimate purpose of achieving efficient and robust hybrid systems motion planning through sampling-based planners. In particular, the thesis contains three major results:

1. A RRT-style planner named "R3T", which stands for **r**apidly-exploring **r**andom **r**eachable set **t**rees. R3T is designed for planning in kinodynamic and hybrid systems.

2. Mathematical tools for solving the "nearest polytope problem" as motivated by motion planning examples such as R3T.

3. A manipulation planning framework that combines RRT and learning techniques to achieve robust manipulation planning. This method is focused on the "planar gripper" robot system.

Chapter 2 presents and analyzes R3T. Chapter 3 introduces the mathematical tools for solving the nearest polytope problem. Chapter 4 discusses manipulation planing on the planar gripper system. Finally, Chapter 5 summarizes the impacts of these results.

Chapter 2

Rapidly-Exploring Random Reachable Set Tree (R3T)[1]

2.1 Background and Motivation

As discussed in Chapter 1.1, performance of sampling-based planners is limited in kinodynamic and hybrid systems. Consequently, many variants of RRT have been developed to improve planning performance. In these systems, the nearest state in the tree to the sample might not be associated with the nearest reachable state to the sample (see Fig. 2-1). Reachability guided methods, such as reachability guided RRT (RG-RRT) [41], address this problem by exploring reachable states in advance and then growing the tree to the nearest reachable state to the sample. However, this requires searching over a much larger number of points. A number of algorithms in the spirit of RG-RRT have been developed. Environment-guided RRT (EG-RRT) [16] combines RG-RRT with a sampling strategy biased toward more promising parts of the state-space; planning with motion cones [6] uses the notion of reachability in configuration space for in-hand manipulation.

In this chapter, a formal approach to reachability-guided sampling-based planning

[1] ©2020 IEEE. Reprinted, with permission, from Albert Wu, Sadra Sadraddini, and Russ Tedrake. R3t: Rapidly-exploring random reachability set trees. To appear in the proceedings of *2020 IEEE International Conference on Robotics and Automation*, 2020.

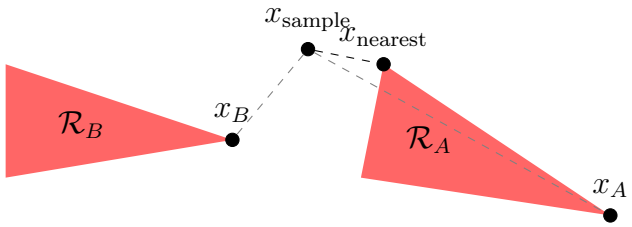


Figure 2-1: RRT extension and system reachability. While x_B is closer to x_{sample} than x_A, the reachable set of x_A, $\mathcal{R}_A$, has states closer to x_{sample} than any point in $\mathcal{R}_B$. A reachability guided algorithm would extend the tree to x_{nearest}, whereas traditional RRT would either extend in $\mathcal{R}_B$ or fail.

is proposed and discussed. A method to represent and plan on the whole reachable set is developed. The main contributions of this chapter are as follows.

- A framework to represent the forward (or backward) reachable set of a state as (the union of) polytope(s) using the linearized local dynamics. (Chapter 2.3.1).
- ***R**apidly-exploring **R**andom **R**eachable Set **T**ree (R3T)* sampling-based planning algorithm guided by (polytopic) reachable sets (Chapter 2.3.2). The algorithm is probabilistically complete in kinodynamic settings with approximated reachable sets(Chapter 2.4.2). R3T*, the version with rewiring (Chapter 2.3.3), retains the asymptotic optimality of traditional RRT*s (Chapter 2.4.3). The benefits of planning with reachable sets are demonstrated using kinodynamic and hybrid systems (Chapter 2.5).
- As a consequence of R3T, a tool for approximate (subject to linearization errors) reachability verification, as opposed to RRT-based falsification [33, 11].

A video summary, which accompanied the conference paper version of this chapter, is available on YouTube[2].

2.2 Problem Definition

A planning problem in a hybrid system, which is the problem of interest in this chapter, is defined in Problem 1.

[2]https://youtu.be/E8TICePNqE0

Problem 1. *Given a system defined by Equation* (1.1)*, an initial state* x_0 *and a goal state* x_G*, find a trajectory* $\zeta : [0, T] \to (x, u)$ *respecting Equation* (1.1)*,* $x(0) = x_0, x(T) = x_G$*,* $x(t) \in X$*,* $u(t) \in U, \forall t \in [0, T]$*, and* T *is finite. If optimality is desired, find the optimal trajectory such that it minimizes* $J = \int_0^T c(x, u)dt$*, where* $c : X \times U \to \mathbb{R}$ *is the running cost. Using a time step* $\tau \in \mathbb{R}_+$ *and a time-integration scheme such as time-stepping ([44]), Equation* (1.1) *can be replaced with Equation 1.2. In the rest of the chapter, usage of Equation* (1.2) *is assumed.*

The mathematical preliminaries on polytopes are given in Chapter 3.2.1.

2.3 The R3T Algorithm

2.3.1 Reachable Set Approximation

In order to perform sampling-based planning with reachable sets, a framework to approximate the forward (or backward) reachable sets of hybrid systems is first proposed.

Definition 1. *The forward reachable set of state* $\bar{x}$ *is defined as the set of all states that can be reached from* $\bar{x}$ *in less than time* τ *using valid control inputs*

$$R(\bar{x}) := \{x \in X | \exists [0, \tau] \to U, x(0) = \bar{x}, (1.2)\}. \tag{2.1}$$

While R3T supports planning with exact $R(\bar{x})$ (see Chapter 2.5.2), finding an explicit representation for $R(\bar{x})$ is generally difficult. However, an approximation can be found with time discretization and linearization. To achieve so, the dynamics and the constraints of each mode of the system is linearized. Following the notations defined in Equation (1.2), given $\eta = (\bar{x}, \bar{u})$, the linearized dynamics is $x^+ = A_i^\eta x + B_i^\eta u + c_i^\eta$, where $A_i^\eta = \frac{\partial F_i}{\partial x}|_\eta, B_i^\eta = \frac{\partial F_i}{\partial u}|_\eta$, and $c_i^\eta = F_i(\bar{x}, \bar{u}) - A_i^\eta \bar{x} - B_i^\eta \bar{u}$. The set of constraints $S_i(x, u) \leq 0$ is linearized to obtain the following polyhedral set in $X \times U$:

$$D_i^\eta x + E_i^\eta u \leq \zeta_i^\eta, \tag{2.2}$$

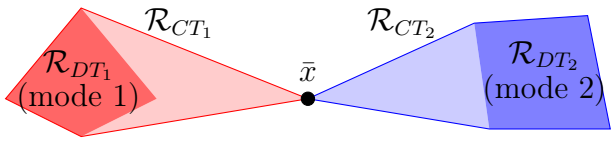


Figure 2-2: Schematic illustration of the reachable sets of a hybrid system. $\mathcal{R}_{DT_1}$ and $\mathcal{R}_{DT_2}$ represent discrete time reachable sets of $\bar{x}$ in 2 different modes. $\mathcal{R}_{CT_1}$ (red) and $\mathcal{R}_{CT_2}$ (blue), the continuous-time reachable set of each mode, is computed by taking the convex hull of $\bar{x}$ and the respective $\mathcal{R}_{DT}$.

where $D_i^\eta = \frac{\partial S_i}{\partial x}|_\eta$, $E_i^\eta = \frac{\partial S_i}{\partial u}|_\eta$, and $\zeta_i^\eta = D_i^\eta \bar{x} + E_i^\eta \bar{u} - S_i(\bar{x}, \bar{u})$. Note that (2.2) is an H-polytope if X and U are bounded sets, which is often the case. An approximation of the reachable set in discrete time is the following set:

$$\mathcal{R}_{DT}(\bar{x}) = \bigcup_{i=1}^{N} (A_i^\eta \bar{x} + c_i^\eta) + B_i^\eta \{u \mid E_i^\eta u \leq \zeta_i^\eta - D_i^\eta \bar{x}\}, \tag{2.3}$$

which is a union of AH-polytopes. It is possible that some of the polytopes in (1.2) are empty—not every mode is attainable at a given state. Following the straight line continuous-time approximation discussed earlier, the polytopic reachable set in continuous time $\mathcal{R}_{CT}$ is approximated by taking the convex hull of $\mathcal{R}_{DT}$ with $\bar{x}$. The convex-hull of a point and an AH-polytope is still an AH-polytope [36]. Thus,

$$\mathcal{R}_{CT}(\bar{x}) = \bigcup_{i=1}^{N} \mathbb{A}(\bar{x}, [B_i^\eta, A_i^\eta \bar{x} + c_i^\eta - \bar{x}], [E_i^\eta, D_i^\eta \bar{x} - \zeta_i^\eta], 0),$$

where $[\cdot, \cdot]$ stands for concatenating matrices horizontally. This construction of reachable sets is illustrated in Fig. 2-2.

2.3.2 The Main R3T Algorithm

Algorithm 1 provides an overview of R3T. First, the tree is initialized with the start state. The tree is then expanded through sampling the state space and performing the `Extend` routine. Whenever a new node is added, R3T may check for more optimal paths through `Rewire` if `optimality` is flagged true. R3T then checks whether the goal is reachable from the new node with `ExtendToPoint`. If it is the case, a path is

Algorithm 1 R3T

Require: $\mathcal{R}$, $\mathcal{S}$, x_0 and x_g ▷ reachable set oracle, sampling function, start state, goal state
Require: `optimality`, i_{max}, ϵ ▷ whether to rewire, max iterations, tolerance for reaching x_g
T.`add`$(x_0, \mathcal{R}(x_0))$ ▷ initialize tree with start state
$i \leftarrow 0$ ▷ reset iterations count
if $x_g \in \mathcal{R}(x_0)$ **then**
  $g \leftarrow$ `ExtendToPoint`$(\mathcal{R}(x_0), x_g)$
  if $g \neq \emptyset$ **then**
    T.`add`(g)
    return `BuildPath`(T, g) ▷ found path to goal
while $i < i_{max}$ **do**
  $x_s \leftarrow \mathcal{S}()$ ▷ sample the state space
  $R_c, x_c \leftarrow$ `FindNearest`(T, x_s)
  $x_n \leftarrow$ `Extend`(R_c, x_c)
  if $x_n \neq \emptyset$ **then** ▷ successful extension
    T.`add`$(x_n, \mathcal{R}(x_n))$ ▷ add new node
    if `optimality==True` **then**
      `Rewire`$(T, \mathcal{R}(x_n))$ ▷ optional rewiring
    if $x_g \in \mathcal{R}(x_n)$ **then**
      $g \leftarrow$ `ExtendToPoint`$(\mathcal{R}(x_n), x_g)$
      if $g \neq \emptyset \wedge$ `Dist`$(g, x_g) < \epsilon$ **then**
        T.`add`(g)
        return `BuildPath`(g) ▷ found path
  $i \leftarrow i + 1$
return $\emptyset$ ▷ R3T failed

found and R3T terminates.

Guiding Tree Growth with Random Sampling

R3T's extension routine is entirely based on the reachable set. First, the nearest reachable set in the tree $R_c \in T$ to the random state sample x_s is found. If $x_s \in R_c$, the tree is extended toward x_s. Otherwise, extension is performed through finding the nearest state $x_c \in R_c$. This sampling method favors states that are feasible and prevents sample rejection, similar to RG-RRT [41]. Moreover, the extension routine of R3T avoids an explicit distance metric if at least one reachable set contains x_s. Design of a distance metric is challenging in underactuated system and hybrid systems, and

Algorithm 2 `Extend` with approximated $\mathcal{R}(\cdot)$

Require: R, x ▷ Approximated reachable set, target state
Require: τ ▷ Time horizon
$u \leftarrow$ `CalcInput`(R, x) ▷ Control input to get to x
$x_n \leftarrow \int_{t=0,x(0)=R.x}^{t=\tau} f(R.x, u)dt$ ▷ Simulate trajectory using u. $R.x$ is the state from which R is generated.
if `CollisionFree`$(R.x, x_n)$ **then**
    return x_n
return $\emptyset$

the "nearest" state found with conventional distance metrics like L_2 and L_∞ can be unreachable or does not progress toward the sample, as illustrated in Fig. 2-1.

Computing Reachable Sets

In general, the reachable set oracle $\mathcal{R}(\cdot)$ computes reachable sets with AH-polytope approximations (Chapter 2.3.1). Some systems such as Dubins car (Chapter 2.5.2) have exploitable properties that allow system-specific reachable set computation.

Extension

`Extend` routine is trivial if R is a conservative approximation, as the target state must be reachable. Otherwise, Algorithm 2 can be used to ensure the extended path is indeed reachable. Notice that x_n may be obtained by a more precise simulation than when generating R.

For the polytopic reachable set approximation discussed in Chapter 2.3.1, `CalcInput` can be implemented using (2.4).

$$u = (B^\eta)^\dagger (x_c - x - A^\eta x - c^\eta)|_{x=R_c.x}, \tag{2.4}$$

where $(.)^\dagger$ stands for Moore-Penrose inverse. Equation (2.4) is implemented in all subsequent experiments. For proving probabilistic completeness, another `CalcInput` is proposed and discussed in Chapter 2.4.2.

A variant of `Extend`, `ExtendToPoint`, has an additional constraint that `Dist`(x_n, x_c) is small. `ExtendToPoint` is used in rewiring and goal checking. This routine is

Algorithm 3 `ExtendToPoint` with input enumeration

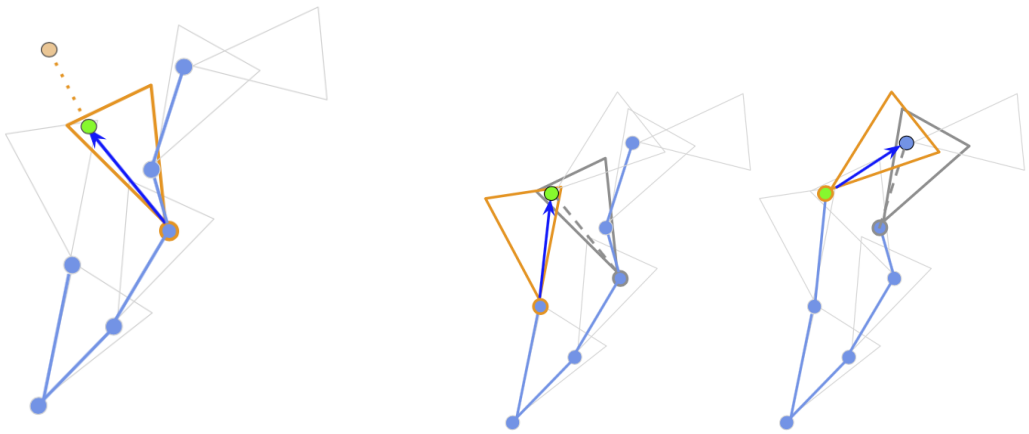
Require: R, x ▷ Approximated reachable set, target state
Require: $\mathcal{U}, \tau$ ▷ Possible control inputs, time horizon
1: $\mathbf{u} \leftarrow \texttt{Sample}(\mathcal{U})$ ▷ Sample control inputs
2: **for** $u_i \in \mathbf{u}$ **do**
3: $\quad \mathbf{x_i} \leftarrow \int_{t=0,x(0)=R.x}^{t=\tau} f(R.x, u_i)dt$ ▷ Simulate trajectory using u_i. $R.x$ is the state from which R is generated.
4: $\quad$ **if** $\texttt{Dist}(x, \mathbf{x_i}) \approx 0 \wedge \texttt{CollisionFree}(R.x, x)$ **then**
5: $\quad\quad$ **return** x
6: **return** $\emptyset$

also trivial if the R is a conservative approximation. If R is an over-approximation, solving the two-point boundary value from $R.x$ to x_n problem is necessary. Algorithm 3 provides an approach where the solution is obtained through sampling the input space. Note that by keeping a reachable set approximation, we can weed out most unreachable states with little computation.

Generally, the explicit trajectory between tree nodes only needs to be calculated during the `BuildPath` routine. Even with rewiring, the algorithm can maintain optimality as long as the cost-to-go is consistent. If there are obstacles in addition to the system dynamics, the collision checking routine `CollisionFree` is performed during extension. Standard collision checking routines used by other RRT algorithms may be applied. The extension routine is illustrated in Figure 2-3a.

2.3.3 Rewiring

R3T may maintain asymptotic optimality (Chapter 2.4.3) through a rewiring procedure similar to [18]. `Rewire` consists of finding the best parent of a newly added node, and using the new node as a potential parent. Algorithm 4 describes `Rewire` in detail. Notice that the first `for` loop is only relevant when multiple reachable sets contain the same state. Moreover, if the nearest reachable set R_c is selected during extension to favor the reachable set with the least cost-to-go, this loop can be skipped entirely. The rewiring routine is illustrated in Figure 2-3b.



(a) Extension in R3T. The nearest reachable set (orange triangle) to the sample (orange dot) is identified. The tree is then grown toward the nearest state (green dot) to the sample in this reachable set.

(b) Rewiring in R3T. The best parent of the new node (green) is identified. If the new node provides a better cost for some existing node, the new node is assigned as the new parent.

Figure 2-3: Extension and rewiring in R3T.

Goal Checking

Whenever a new reachable set is added to the tree, `ExtendToPoint` is performed to see whether the goal state is in the new reachable set. If it is, the goal node is added to the tree, and the algorithm terminates.

2.3.4 Optimizations

Information from the reachable sets can be exploited to accelerate R3T. Here, two techniques that apply to planning with polytopic reachable sets are proposed: computing convex hull with originating states and exploring deterministic states. Both techniques were implemented in Sec. 2.5.

When evaluating the reachable set $\mathcal{R}(x)$ with the method in Sec. 2.3.1, a discrete time step $\tau > 0$ is required to generate $\mathcal{R}$. This may lead to a large distance between x and $\mathcal{R}(x)$. However, due to the linearity of $\mathcal{R}$, the reachable set for any time horizon $0 \leq \tau' \leq \tau$ is the convex set of x and $\mathcal{R}(x)$. This allows choosing a coarse τ without missing subsets of the state space. For comparison, vertex-based traditional RRT and RG-RRT has a resolution bounded by the Nyquist sampling criterion. For a state

Algorithm 4 `Rewire`

Require: T, R ▷ R3T, newly added reachable set
Require: $\mathcal{C}$ ▷ Cost-to-go function
$\mathbf{R}_R \leftarrow \texttt{Contains}(T.\mathbf{R}, R.x)$ ▷ Find all reachable sets containing R
for $R_i \in \mathbf{R}_R$ **do**
  if $\mathcal{C}(R.x) > \mathcal{C}(R_i.x) + \mathcal{C}(R_i.x, R.x)$ **then**
    $x' \leftarrow \texttt{ExtendToPoint}(R_i, R.x)$
    if $x' \neq \emptyset \wedge \texttt{Dist}(x', R.x) \approx 0$ **then**
      $R.x.\text{parent} \leftarrow R_i.x$ ▷ Rewire $R.x$ with R_i
$\mathbf{x}_R \leftarrow \texttt{Contains}(R, T.\mathbf{x})$ ▷ Find all explored states contained in R
for $x_j \in \mathbf{x}_R$ **do**
  if $\mathcal{C}(x_j) > \mathcal{C}(R.x) + \mathcal{C}(R.x, x_j)$ **then**
    $x'_j \leftarrow \texttt{ExtendToPoint}(R, x_j)$
    if $x'_j \neq \emptyset \wedge \texttt{Dist}(x'_j, x_j) \approx 0$ **then**
      $x_j.\text{parent} \leftarrow R.x$ ▷ Rewire x_j with R
return

spacing of Δx, an ϵ-ball in the state space is not guaranteed to be discovered unless $\Delta x < 2\epsilon$.

Another useful technique in kinodynamically constrained system is to exploit deterministic dynamics. If the input matrix $B = \mathbf{0}$ in some dynamic modes, the control inputs do not affect state evolution. In this case, the node can be explored further with little computational cost through simulating forward dynamics until $B \neq \mathbf{0}$.

2.4 Analysis of R3T

2.4.1 Correctness of R3T

The correctness of R3T follows by construction. Since all paths planned belong in the reachable set of some state, the path found by the algorithm is feasible. In the case where the reachable set is approximated, Algorithm 2 can be applied to ensure all tree edges are feasible.

2.4.2 Probabilistic Completeness (PC) in Kinodynamic Systems

While PC in geometric settings has been established in literature (see [25, 19, 5]), PC in kinodynamic settings has not reached the same level of generality. However, PC in kinodynamic systems can be shown in R3T. Assume there exists a *robustly feasible* (see [17]) solution $\mathcal{P}$ to Problem 1. Furthermore, assume the system is Lipschitz continuous with 1 dynamic mode, and the support of the sampling function $\mathcal{S}$ contains $\cup\{R(x) \mid \forall x \in \mathcal{P}\}$.

Theorem 1. *R3T with exact reachable sets is PC.*

Proof. Leveraging the results and notations from [21], we only need to show the probability of a successful propagation into $\mathcal{B}_{\kappa\delta}(x_1)$ from $x_0' \in \mathcal{B}_\delta(x_0)$ is nonzero. Since the path is robustly feasible, $\mathcal{B}_{\kappa\delta}(x_1) \cap R(x_0')$ has nonzero volume and nonzero probability of being sampled. □

For R3T with approximated reachable sets $\bar{R}(\cdot)$ as discussed in Algorithm 3, the following additional assumptions are necessary.

Assumption 1. *There exists $\mathcal{M}_x : \bar{R}(x) \to R(x)$ defining the following relationship: for $\forall y \in R(x), \exists \bar{y} \in \bar{R}(x)$ such that if $\bar{y}$ is sampled, y is extended to from x. An alternative assumption is every feasible input $u \in U$ and time $t \in [0, \tau]$ has a nonzero probability of being sampled.*

Assumption 2. *In the case where a state is contained in multiple $\bar{R}(x)$, we break ties randomly. For convenience, we assume the tie breaking probability is uniform.*

Under these assumptions, the following theorem is proposed.

Theorem 2. *R3T with reachable set approximations satisfying Assumptions 1 and 2 is still PC.*

Proof. By Assumption 1, $\exists \bar{\mathcal{B}}_{\kappa\delta}(x_1) := \{y \mid \mathcal{M}_{x_0'}(y) = x, x \in \mathcal{B}_{\kappa\delta}(x_1) \cap R(x_0')\} \neq \emptyset$. Denote the probability of sampling in $\bar{\mathcal{B}}_{\kappa\delta}(x_1)$ as p, $0 < p \leq 1$. There may be tree

nodes $Z = \{z_1, z_2, \ldots, z_m\} \subset T$ such that $\bar{\mathcal{B}}_{\kappa\delta}(x_1) \cap \bar{R}(z_i) \neq \emptyset$. By Assumption 2, the probability of choosing x_0' is at least $\frac{1}{m+1}$. The probability of propagating from x_0' into $\bar{\mathcal{B}}_{\kappa\delta}(x_1)$ is $\frac{p}{m+1} \geq \frac{p}{|\mathcal{T}|}$, where $|\mathcal{T}|$ is the tree size. Suppose initially $|T| = k$ and n extensions are performed. The probability of *failing* to propagate into $\bar{\mathcal{B}}_{\kappa\delta}(x_1)$ is upper bounded by $\prod_{i=0}^{n-1}(1 - \frac{p}{k+i})$. For $0 < a < 1$,

$$\ln(1-a) = -a - \frac{a^2}{2(1-\epsilon)^2} \ \leq -a, \ 0 < \epsilon < a. \tag{2.5}$$

The logarithm of the failure probability after infinite steps is

$$\lim_{n\to\infty} \sum_{i=0}^{n-1} \ln(1 - \frac{p}{k+i}) \leq \lim_{n\to\infty} \sum_{i=0}^{n-1} -\frac{p}{k+i} = -\infty, \tag{2.6}$$

thus

$$\lim_{n\to\infty} \prod_{i=0}^{n-1} (1 - \frac{p}{k+i}) = 0. \tag{2.7}$$

In virtue of (2.7), after infinite steps, the propagation to $\bar{\mathcal{B}}_{\kappa\delta}(x_1)$ succeeds almost surely. □

To achieve PC with polytopic reachable set $\bar{R}(\cdot)$, consider an alternative `CalcInput` satisfying Assumption 1. Observe that extending to $x_s \in \bar{R}(x_c)$ is a polytopic constraint on the time step $\bar{t} = \beta\tau$ and reparameterized input $v := \beta u$.

$$x_s = (A^\eta x + c^\eta)\beta + (B^\eta)v + x_c, v \in \beta U, \beta \in [0,1]. \tag{2.8}$$

Since x_s may be any state in $\bar{R}(\cdot)$, if a random polytope sampler such as [31] is used to choose $u, \bar{t}$ for Algorithm 2, all feasible $u, \bar{t}$ can be chosen and the setup is PC.

2.4.3 Asymptotic Optimality with Rewiring

Definition 2. *An R3T tree T is "optimal" if the edges connecting the root x_r and all $x_t \in T$ form a path no more costly than any other path from x_r to x_t via waypoints $\{x_i\} \in T$, where each $x_{i+1} \in R(x_i)$.*

Theorem 3. *Given an R3T tree T constructed with the rewiring procedure, the tree is optimal.*

Proof. The proof is done using an inductive argument. Given a optimal tree T, suppose a new node x_n is added. If a suboptimal path appears after adding x_n, it must contain x_n. The rewiring procedure checks for better paths involving $\{x \mid x_n \in R(x) \wedge x \in T\}$ and $\{x \mid x \in R(x_n) \wedge x \in T\}$, which covers all path segments involving x_n with duration $\leq \tau$. Therefore, $T \cup \{x_n\}$ is still optimal. The "base case" of a tree with 1 node is optimal, so the proof is complete.

□

With PC, all possible paths will be explored by R3T given a long enough running time. Therefore, it is speculated that R3T posses the asymptotic optimality given in [17, 43]. While bounding the complexity of rewiring is not in the scope of this chapter, empirical evidence suggests that rewiring is beneficial in practice.

2.5 Empirical Evaluation

To evaluate the performance of R3T, R3T, RG-RRT [41], and RRT [26] were implemented for testing. The RRT and RG-RRT implementations sample 3 evenly-spaced inputs for tree extension using their respective strategies. All scripts are available on GitHub[3]. All tests were performed on a personal computer with i7-7820HQ CPU.

2.5.1 Pendulum Swing-Up

Consider a single-link torque-limited pendulum system with damping. The pendulum was started at rest, and the goal is to swing up the pendulum to rest at $\theta = \pi$. The tolerance for reaching the goal is $||x - x_g||_2 \leq 0.05$. For R3T and RG-RRT, a reachable set time horizon of 0.2s was used. The time step size for RRT and forward dynamics was 0.01s. 10 consecutive planning tries were done, and the results are summarized in Table 2.1. The system parameters used are mass $m = 1$kg, link length $l = 0.5$m,

[3]https://github.com/wualbert/r3t

gravity constant $g = 9.8\text{m/s}^2$, joint damping coefficient $b = 0.1\text{kg/s}$, and torque limit $|\tau| \leq 1\text{N}$.

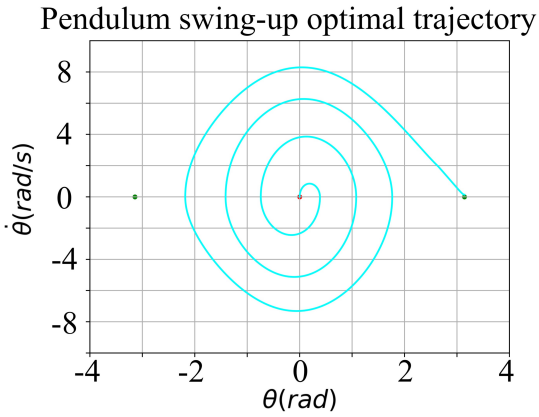
Table 2.1: Path planning statistics with the pendulum.

	R3T		RG-RRT		RRT	
	Time(s)	Nodes	Time(s)	Nodes	Time(s)	Nodes
Mean	5.92	559	21.4	4352	45.8	11134
Median	4.07	472	6.6	1381	31.0	8349
Max	17.26	1218	110.8	22600	97.5	19360
Min	2.14	284	1.5	336	28.0	7677
S.D.	5.09	326	33.8	4353	23.6	4273

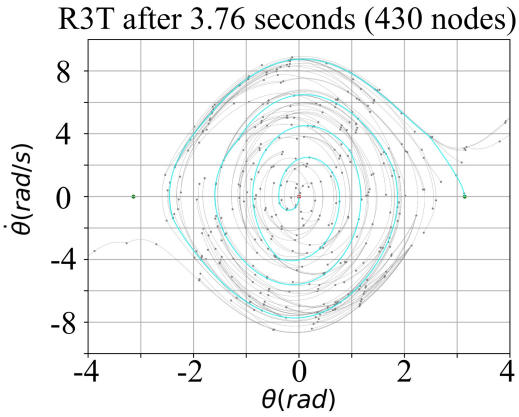
R3T significantly outperformed RG-RRT and RRT in both runtime and nodes explored. This is attributed to the large simulation timestep allowed by planning with reachable sets. Moreover, by using the convex hull technique discussed in Sec. 2.3.4, choosing a large simulation time step does not cause R3T to "overlook" the goal states, unlike in RG-RRT and RRT where the exploration often passed by the goal but not terminate. One way to mitigate this pitfall is to add a possibly expensive routine in RG-RRT and RRT to check if the goal lies on the path between two nodes. Fig. 2-5 shows the reachable sets explored by R3T as time progresses.

2.5.2 Dubins Car

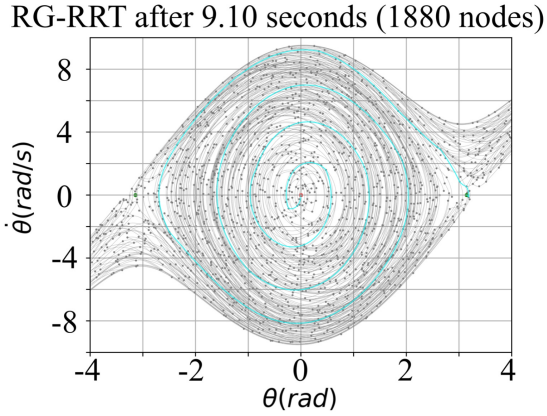
To demonstrate asymptotic optimality of R3T, the "Dubins car" problem is considered. The problem is to find a path subject to a car's kinematic constraints. The problem can be simplified to finding a path that consists of curvature-limited Dubins curves [25]. Observing that the reachable set from each state is just a linear transformation of a "base" reachable set, a fixed-horizon exact reachable set was precomputed through simulating the system dynamics forward. Fig. 2-6a and 2-6b shows the solution improvements as time progressed. Fig. 2-6c plots the path length against time. As expected from asymptotic optimality, the cost is monotonically decreasing.



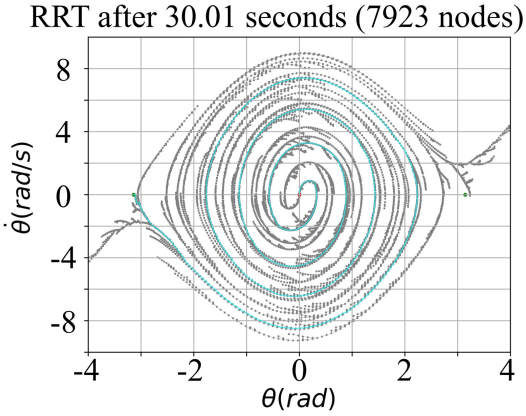
(a) Optimal trajectory from a bang-bang controller.



(b) Trajectory found by R3T.

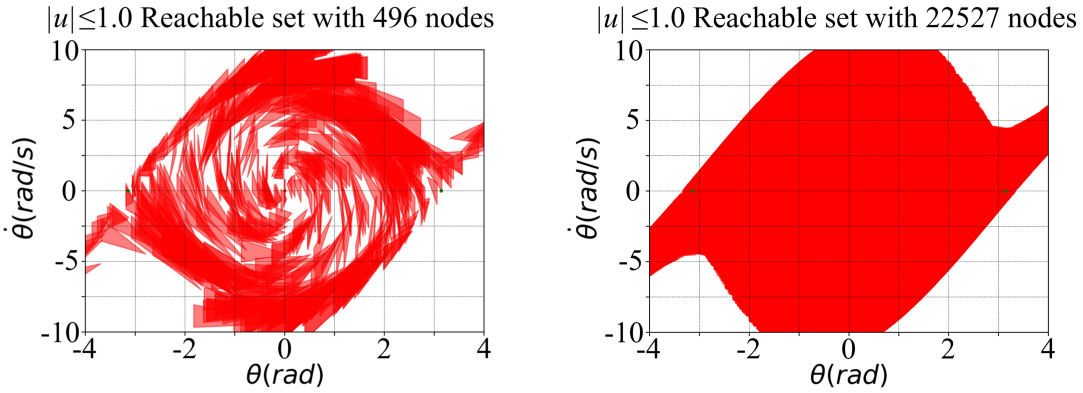


(c) Trajectory found by RG-RRT.



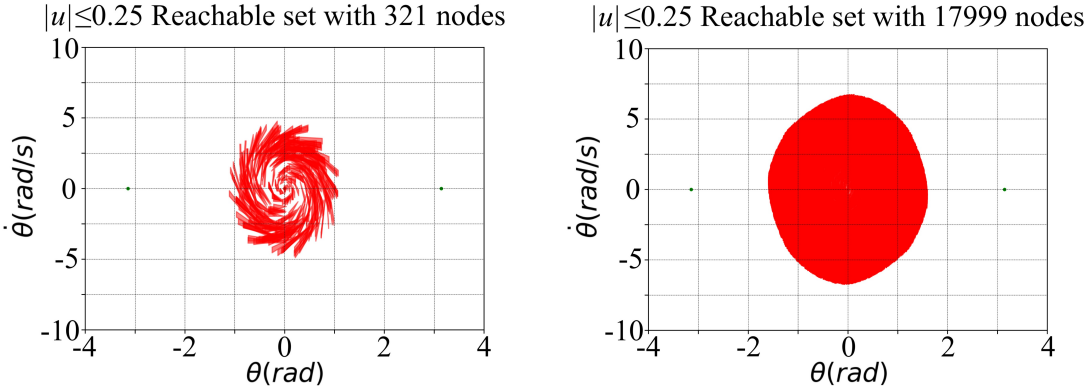
(d) Trajectory found by RRT.

Figure 2-4: Pendulum swing-up trajectories found by various planning algorithms. R3T found a solution significantly faster with fewer nodes explored.



(a) Reachable set with $|u| < 1$N after 5 seconds.

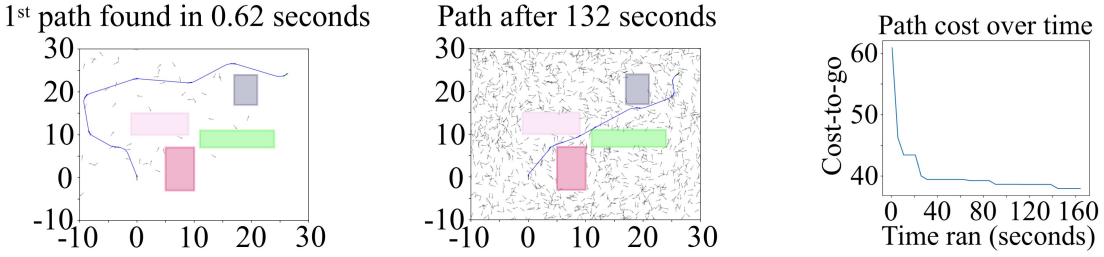
(b) Reachable set with $|u| < 1$N after 3905 seconds.



(c) Reachable set with $|u| < 0.25$N after 5 seconds.

(d) Reachable set with $|u| < 0.25$N after 3905 seconds.

Figure 2-5: Polytopic approximation of the pendulum reachable set as explored by R3T. Swing-up is impossible with the input limit and the damping coefficient in 2-5c and 2-5d. As the runtime increased, nearly all feasible states were covered, proving probabilistic completeness empirically.



(a) First path found. Cost-to-go is 60.9.

(b) Path after 132s. Cost-to-go is 38.6.

(c) Cost-to-go plotted over time.

Figure 2-6: Results of R3T on Dubins car. Figures 2-6a, 2-6b are snapshots of the explored states (grey) and solution (blue). As shown in Fig. 2-6c, the longer the runtime, the less costly the solution was.

2.5.3 1D Hopper

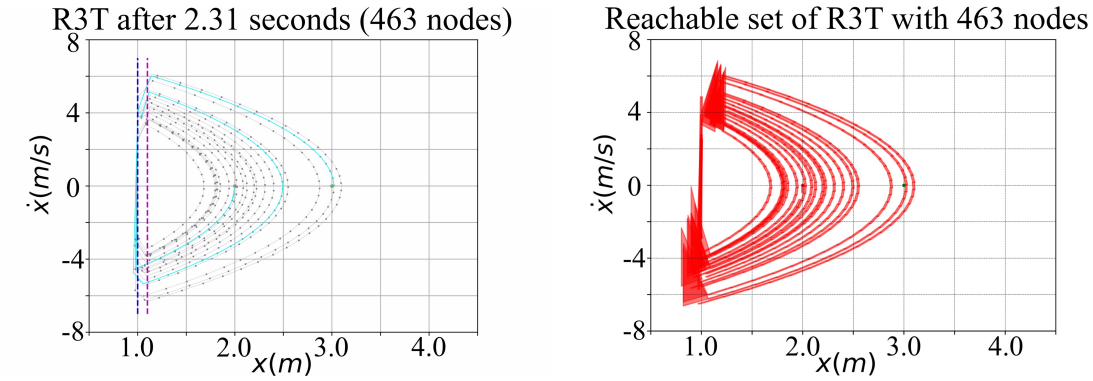
The 1D hopping robot is used to compare R3T and other method's ability to plan on hybrid systems. The model has 2 states, 2 continuous dynamic modes (flight, soft ground contact), and 1 discrete dynamic mode (inelastic collision with the ground). The hopper has a body mass of $m = 1$kg, a retractable piston leg of length $l = 1$m and piston length p, $0 \leq p \leq p_{max} = 0.1$m. The piston is force-limited to $F < F_{max} = 80$N. The system has 3 dynamic modes: flight, soft contact, and hard contact. When $p = p_{max}$ and $x > l + p_{max}$, the hopper is in flight. During soft contact, $0 < p \leq p_{max}$ and $x = l + p$. The piston can exert force on the ground. During hard ground contact, $p = 0$, $x \leq l$, $\dot{x} < 0$. The hopper performs an instantaneous partially inelastic collision with the ground with damping factor $b = 0.85$.

For the tests, the hopper should hop from 2m to 3m. The tolerance for reaching the goal is $||s - s_g||_2 \leq 0.05$. For R3T and RG-RRT, a reachable set time horizon of 0.04s was used. The time step size for RRT and forward dynamics was 0.01s. 10 consecutive planning tries were done with a maximum runtime of 100s. The statistics on successful tries of finding the first path are summarized in Table 2.2.

Table 2.2: Path planning statistics with the 1D hopper.

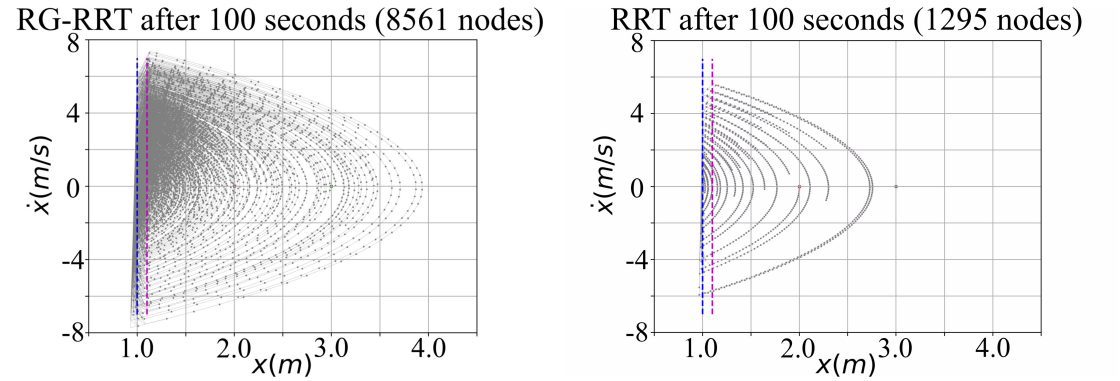
	R3T		RG-RRT		RRT	
	Time(s)	Nodes	Time(s)	Nodes	Time(s)	Nodes
Mean	2.39	530	47.23	4288		
Median	2.18	497	35.54	3403		
Max	5.01	1021	96.49	8006	N/A	N/A
Min	0.37	75	19.74	2194		
S.D.	1.28	265	30.92	2432		
Fails	0		2		10	

Fig. 2-7 shows the trees explored by each algorithm. R3T is the only algorithm that found a path consistently and quickly. Two properties from planning with reachable sets contribute to this result. Since the 1D hopper has no control input during



(a) Nodes (grey) and trajectory (cyan) found by R3T.

(b) Reachable set corresponding to the path in Fig. 2-7a.



(c) Instance where RG-RRT failed. With a small goal region, RG-RRT failed to acknowledge the goal between two nodes.

(d) Instance where RRT failed. To match the simulation granularity of R3T, RRT needed a much smaller time step.

Figure 2-7: 1D hopper trajectories found by various planning algorithms. R3T finds a solution consistenly and quickly. RG-RRT and RRT each suffer from different failure modes.

flight phase, R3T can exploit this property using methods described in Sec. 2.3.1. In addition, maintaining reachable sets allows for more accurate distance-to-goal calculation and goal identification, as discussed in Sec. 2.5.1.

2.5.4 2D Hopper

The 2D hopper robot [35, 46] has 10 states, 2 control inputs, and 2 contact modes [46]. The task is to make the robot hop from $x = 0$ to $x = 10$. A body-attitude controller was used in flight and the leg was modeled as a constant-k spring during compression. R3T was used to plan push-off and hip torque during contact. 10 consecutive runs were performed. R3T found a path in all runs using a median time of 106.3s and

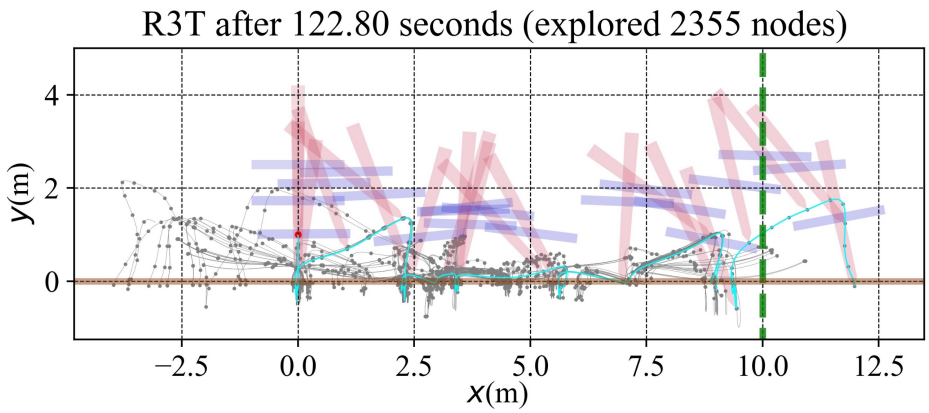


Figure 2-8: 2D hopper trajectory found by R3T. The solution trajectory is shown in cyan. The tree explored and the nodes are in grey. Snapshots of the hopper configurations (red for leg, purple for body) are shown.

2163 nodes. Fig. 2-8 shows a successful run.

2.6 Chapter Summary and Future Work

In this chapter, the R3T algorithm, a variant of RRT which takes advantage of reachable sets, is proposed. In addition, a framework for R3T planning in nonlinear hybrid systems using local linearization, a proof for probabilistic completeness of R3T in kinodynamic systems, and a rewiring procedure that provides asymptotic optimality for R3T are discussed. Case studies showed that R3T outperforms previous RRT methods in speed and nodes explored, suggesting that R3T is a superior candidate algorithm for planning in kinodynamic and hybrid systems.

For future work, one potential direction is to introduce robust planning to R3T. Since the reachable sets are used explicitly in R3T, it is natural to consider possibilities of introducing robustness through modifying the reachable sets. As an example, the reachable set could be shrunk to avoid requiring extreme control inputs that may be difficult to execute.

Chapter 3

The Nearest Polytope Algorithms [1]

3.1 Background and Motivation

As motivated by the routine of finding the nearest polytopic reachable set approximation from Chapter 2, this chapter discusses the geometrical problem of finding the nearest polytope to a point. Given a finite list of polytopes in n-dimensional space and a query point, the *point location problem* is finding the set of polytopes that contain the point. When this set is empty, a more general and difficult problem, the *nearest polytope problem*, is determining the nearest polytope given a metric. The solution to the point location problem is contained in the solution to the nearest polytope problem - the set of polytopes with zero distance from the point. The naive solution to both problems is an exhaustive check over all polytopes. However, evaluating point membership or computing the distance from a polytope to a query point are expensive operations.

Both the point location problem and the nearest polytope problem are purely geometrical but have profound applications in control theory. The point location problem has long been studied in explicit linear model predictive control (MPC) literature, where polytopes represent pre-computed sets that correspond to linear control laws [42, 2, 48, 1]. Given a "query" state, one needs to locate the corresponding "active"

[1] ©2020 IEEE. Reprinted, with permission, from Albert Wu, Sadra Sadraddini, and Russ Tedrake. The nearest polytope problem: Algorithms and application to controlling hybrid systems. To appear in the proceedings of *2020 American Control Conference*, 2020.

polytope to execute the associated control law. This scheme is typically faster than solving the MPC optimization problem online. In "exact explicit MPC", the polytopes are non-overlapping and are described by their hyperplanes, a representation known as H-polytopes [53]. Infeasibility is returned if the query state is not within any of the polytopes. The nearest polytope problem is relevant in "approximate explicit MPC", where the nearest but not necessarily query point-containing polytopic feasible set is used as a heuristic to compute control inputs [39].

Most previous work on the point location problem requires the disjoint space partitioning condition to hold. The work in [7] is able to handle overlapping polytopes. Surprisingly, to the best of the author's knowledge, there has been no development on efficient algorithms for the nearest polytope problem. For instance, in [39], an exhaustive search over the whole set of polytopes was performed.

This chapter presents 2 algorithms for solving the nearest polytope problem. The first algorithm is based on axis-aligned bounding boxes (AABB) of the polytopes. The second algorithm is based on precomputing distances to a finite set of key points. The algorithms are analyzed and benchmarked with synthetic and real-world datasets motivated by MPC and R3T applications.

3.2 Problem Definition

3.2.1 Introduction to Polytopes

First, the definition of polytopes are provided. Denoted by $\mathbb{R}^n$ the set of n-dimensional real values. Given $a \in \mathbb{R}^n$, the transpose of a is denoted by a^T. Given two matrices A, B with appropriate dimension, their vertical stacking is $[A, B]$. Given a finite set S, $|S|$ is its cardinality.

Definition 3. *(H-Polytope) [53]. An H-polytope $P \subset \mathbb{R}^n$ is a bounded set described by its hyperplanes:*

$$P = \{x \in \mathbb{R}^n \mid Hx \leq h\},$$

where $H \in \mathbb{R}^{q \times n}$ and $h \in \mathbb{R}^q$, q is the number of hyperplanes, the kernel of H is only

the origin, and all the inequalities are interpreted element-wise.

Definition 4. *(AH-Polytope) [37]. An AH-polytope $P \subset \mathbb{R}^n$ is a polytope given by an affine transformation $G \in \mathbb{R}^{n\times m}$, $g \in \mathbb{R}^n$, of polytope $S \subset \mathbb{R}^m$:*

$$P = \{g + Gx \mid Hx \leq h\},$$

where $H \in \mathbb{R}^{q\times m}$ and $h \in \mathbb{R}^q$.

AH-polytopes arise naturally when approximating reachable sets of bounded-input dynamical systems through linearization, which will be discussed in detailed in Chapter 2.3.1. The AH-polytope representation is the most general of the forms discussed above. Therefore, the discussion in this thesis will focus on AH-polytopes.

Given a AH-polytope $P \subset \mathbb{R}^n$, a point $q \in \mathbb{R}^n$, a metric $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_{\geq 0}$, the point-polytope distance function is defined as:

$$d(P, q) := \min_{p\in P} d(p, q), \tag{3.1}$$

and the closest point in $p^* \in P$ to q is

$$p^* = \arg\min_{p\in P} d(p, q). \tag{3.2}$$

The minimization in (3.1) can be cast as the following optimization problem:

$$\begin{aligned} d(P, q) = \quad & \min . && \|\delta\|_\rho \\ & \text{subject to} && Gx + g = q + \delta, \\ & && Hx \leq h, \end{aligned} \tag{3.3}$$

which is a linear program for $\rho = 1, \infty$.For $\rho = 2$, a quadratic program can be constructed by optimizing for the squared norm $\|\cdot\|_2^2$. Other types of norms such as polytopic norms can be also used. $q \in P$ if and only if $d(P, q) = 0$. In this section $\rho = 2$ is used. Note that this section focuses on reducing the *number* of $d(P, q)$ evaluations. Hence, while the complexity of $d(P, q)$ scales with the dimension and

the shape of the polytope, it is considered as a constant in the subsequent discussion. In practice, computing $d(P, q)$ requires solving a convex optimization problem, which its complexity grows polynomially with dimensions and the complexity of the representation of the polytope.

3.2.2 Formulation of the Nearest Polytope Problem

This chapter seeks to solve the following problem:

Problem 2. *Given a set of AH-polytopes* $\mathbf{P} = \{P_i\}_{i=1,\cdots,N}$, $P_i \subset \mathbb{R}^n$, *and a query point* $q \in \mathbb{R}^n$, *find the nearest* $P^* \in \mathbf{P}$ *such that*

$$P^*(q) = \arg\min_{P \in \mathbf{P}} d(P, q). \tag{3.4}$$

If $P^*(q)$ *is not unique, return one of the minimizers.*

Since $\mathbf{P}$ is often given in advance, the task is further split into "offline" and "online" components. The "offline" component consists of preprocessing $\mathbf{P}$ and constructing data structures. In some applications such as motion planning [52], $\mathbf{P}$ is expanded incrementally. The "online" component is the actual query and is more speed critical. The subsequent algorithms focus on creating a fast data structure offline to allow sub-$\mathcal{O}(|\mathbf{P}|)$ online computation. When the minimizer is not unique, a control theoretic tiebreaker such as the (approximate) value function associated with the polytopes [39] could be used. In the subsequent sections, a minimizer is chosen arbitrarily if multiple exist.

3.3 The Axis-Aligned Bounding Box (AABB) Algorithm

3.3.1 Preliminaries

For obtaining an approximation of a polytope that supports quick querying and construction, consider the axis-aligned bounding box, or AABB, given in Definition 5:

Definition 5. *(Axis-Aligned Bounding Box, AABB) [15]. An axis-aligned bounding box (AABB) $B \in \mathbb{R}^n$ can be described by its the "lower" and "upper" corner points $(l, u) \in \mathbb{R}^n$.*

$$B(l, u) = \{x \mid l_i \leq x_i \leq u_i, \forall i = 1, 2, ..., n\}. \tag{3.5}$$

Finding the axis aligned bounding box (l, u) of an AH-polytope $P \in \mathbb{R}^n$ can be formulated as a linear optimization problem. Let $\mathbb{1}_i$ be the column vector with 1 at the ith entry and 0 at all the other entries. The optimization problem is given in Proposition 1.

Proposition 1. *(AABB of an AH-Polytope). The AABB $B(l, u)$ of $P = \{g + Gx \mid Hx \leq h\}$ can be found by solving the following linear programs:*

$$\begin{aligned} l_i &= \min_{x|Hx \leq h} \mathbb{1}_i \cdot (g + Gx), \\ u_i &= \max_{x|Hx \leq h} \mathbb{1}_i \cdot (g + Gx). \end{aligned} \tag{3.6}$$

3.3.2 The AABB algorithm

This algorithm leverages AABBs to create a data structure for fast nearest polytope querying. During the offline phase, for each of the polytopes $P_i \in \mathbf{P}$, AABBs B_i are constructed such that $P_i \subseteq B_i$. Through the `BuildFastAABBStructure` routine, the AABBs are stored in a data structure $\mathbf{B}$ such as R-tree [13] that supports fast AABB querying. A data structure $\mathbf{K}$ is maintained using the `BuildFastPointStructure` routine for a constant number of key points from each polytope. An example of this structure is a k-d tree [3]. The only requirement for the key points K is they must each be inside a polytope. In Section 3.5, arbitrary points that are by construction inside the polytopes are used. The offline precomputation is summarized in Algorithm 5.

Given a query point q online, the closest key point $p^* = \arg\min_{p \in \mathbf{K}} d(p, q)$ is first found. The "pivot" polytope P^* that contains p^* is identified, then $d^* = d(P^*, q)$ is

Algorithm 5 `AABB-Precompute`

Require: $\mathbf{P}$ ▷ List of polytopes
Require: $K \in \mathbf{P}$ ▷ Key points inside the polytopes
1: $B \leftarrow \emptyset$
2: **for** $P_i \in \mathbf{P}$ **do**
3: $\quad B_i \leftarrow \texttt{ComputeAABB}(P_i)$ ▷ Compute AABB
4: $\quad B \leftarrow B \cup \{B_i\}$ ▷ Store AABB
5: $\mathbf{B} \leftarrow \texttt{BuildFastAABBStructure}(B)$
6: $\mathbf{K} \leftarrow \texttt{BuildFastPointStructure}(K)$

computed. A "heuristic box" B_h with side length $2d^*$ is constructed, and $\mathbf{B}$ is queried for all overlapping boxes $\mathbf{B}_v$ with B_h. A polytope P_r is then chosen randomly from all the polytopes corresponding to $\mathbf{B}_v$, compute the distance $d(P_r, q)$, then compare to P^*. If P_r is closer, P^* is replaced with P_r, B_h is reconstructed using the new P^*, and $\mathbf{B}_v$ is updated. This is repeated until either $\mathbf{B}_v$ is exhausted, which means the nearest polytope is P^*, or a polytope containing q is found. The online computation is summarized in Algorithm 6.

3.3.3 Analysis of the AABB algorithm

This algorithm is valid for L1, L2 and L-infinity distance metrics. This is due to the fact that all AABB operations are intersection checks and requires no distance metric. As long as the heuristic box B_h is constructed using a side length of $2d_\rho(\cdot,\cdot), \rho = 1, 2, \infty$, B_h will encompass all polytopes that can be closer to q than the pivot polytope. Additionally, the algorithm can potentially be generalized to arbitrary convex objects as long as the AABB of such object can be computed.

The algorithm differ from [7] in that it can handle the case where no polytope contains the query point q. This is done by the construction of the heuristic box. Since the heuristic box must contain at least one polytope and encompasses everywhere that has closer distance to q than the polytope, the true closest polytope is guaranteed to be found through searching over boxes overlapping with the heuristic box. Figure 3-1 visualizes this process.

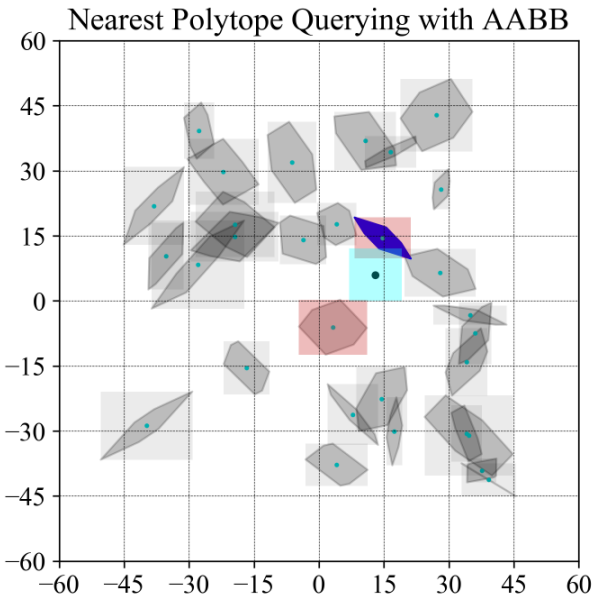


Figure 3-1: Example of a nearest polytope query with AABB. $|\mathbf{P}| = 30$. L2 distance was used. The query point q is marked in black, while the key points K are marked in blue. Through constructing a heuristic box B_h (cyan) to prune impossible polytopes and gradually shrinking it using Algorithm 6, only a small subset of $d(P, q)$ (red boxes, $|\mathbf{B}_v| = |\mathbf{P}_{cand}| = 2$) needs to be evaluated.

Algorithm 6 `AABB-Query`

Require: $\mathbf{B}$, $\mathbf{K}$, q ▷ AABBs, key points, query point
$p^* \leftarrow \arg\min_{p \in \mathbf{K}} d(p, q)$ ▷ Find closest key point
$P^* \leftarrow p^*.P$ ▷ Choose corresponding polytope as pivot
$d^* \leftarrow d(P^*, q)$ ▷ Compute distance to pivot
$B_h \leftarrow \texttt{AABB}(q, 2d^*)$ ▷ Construct AABB centered at q with side length $2d_p$
$\mathbf{B}_v \leftarrow \{B \mid B \in |T|, B \cap B_h \neq \emptyset\}$ ▷ Find AABBs intersecting B_h
$\mathbf{P}_{cand} \leftarrow \mathbf{B}_v.P$ ▷ Get candidate polytopes corresponding to $\mathbf{B}_v$
while $|\mathbf{P}_{cand}| > 0$ **do**
    if $d^* = 0$ **then**
        return P^*, d^* ▷ Found a polytope containing q
    $P_s \leftarrow \texttt{Sample}(\mathbf{P}_{cand} \backslash P^*)$
    $d_s \leftarrow d(P_s, q)$
    if $d_s \geq d^*$ **then**
        $\mathbf{P}_{cand} \leftarrow \mathbf{P}_{cand} \backslash P_s$
    else
        $P^*, d^* \leftarrow P_s, d_s$
        $B_h \leftarrow \texttt{AABB}(q, 2d^*)$
        $\mathbf{B}_v \leftarrow \{B \mid B \in |T|, B \cap B_h \neq \emptyset\}$ ▷ Update the list of candidates
return P^*, d^*

3.4 The Triangle Inequality (TI) Algorithm

In this approach, the distance between a query point q and each of the polytopes $P \in \mathbf{P}$ is lower bounded by the triangle inequality (Theorem 4). The bound is used to prune the list of polytopes to evaluate online.

3.4.1 Preliminaries

Theorem 4. *(Point-Polytope Triangle Inequality). Given points $u, v \in \mathbb{R}^n$, a polytope $P \subset \mathbb{R}^n$, and metric $d(\cdot, \cdot)$, $d(u, v) + d(v, P) \geq d(u, P)$*

Proof. Suppose the closest point to u in P is $u^* \in P$, the closest point to v in P is $v^* \in P$. By the triangle inequality,

$$d(u, v) + d(v, v^*) \geq d(u, v^*).$$

By definition of v^*,

$$d(u, v^*) \geq d(u, u^*).$$

Therefore,

$$\begin{aligned} d(u, v) + d(v, P) =& d(u, v) + d(v, v^*) \\ \geq& d(u, u*) = d(u, P) \end{aligned}$$

□

If the polytopes are given in advance, Theorem 4 can be used to perform precomputations that reduce online computation. Specifically, evaluating $d(P, q)$ is avoided for polytopes P that cannot possibly be the closest. Given two polytopes P_1, P_2, a key point k, and a query point q, Theorem 5 provides a lower bound on $d(P_2, q)$ given $d(P_1, q)$ and $d(k, q)$.

Theorem 5. *(Point-Polytope Distance Lower Bound). Given two polytopes P_1, P_2, a key point k and a query point q, the following condition implies a lower bound on $d(P_2, q)$ by $d(P_1, q)$:*

$$d(P_2, k) \geq d(k, q) + d(P_1, q) \implies d(P_2, q) \geq d(P_1, q) \tag{3.7}$$

Proof. From Theorem 4,

$$d(P_2, q) \geq d(P_2, k) - d(k, q)$$

Thus if $d(P_1, q) \leq d(P_2, k) - d(k, q)$, $d(P_1, q) \leq d(P_2, q)$. □

3.4.2 The TI Algorithm

Algorithms 7 and 8 describe a procedure for finding the closest polytope leveraging Theorem 5. When a new set of polytopes $\mathbf{P}$ is given to the algorithm, `TI-Precompute`

Algorithm 7 `TI-Precompute`

Require: $\mathbf{P}$ ▷ List of polytopes
Require: $K \in \mathbf{P}$ ▷ Key points inside the polytopes
$\mathbf{D} \leftarrow \emptyset$ ▷ Initialize distance map
for $k_i \in K$ **do**
  for $P_j \in \mathbf{P}, P_j \neq P(k_i)$ **do**
    $\mathbf{D}(k_i, P_j) \leftarrow d(k_i, P_j)$
  `sort`$(\mathbf{D}(k_i, \cdot))$ ▷ Sort key point distances
$\mathbf{K} \leftarrow$ `BuildFastPointStructure`(K)

Algorithm 8 `TI-Query`

Require: q ▷ Query point
Require: $\mathbf{P}, \mathbf{K}$
$k^* \leftarrow \arg\min_{k \in \mathbf{K}} d(k, q)$ ▷ Find closest key point
$d^* \leftarrow d(q, P(k^*))$ ▷ Compute distance to polytope corresponding to k^*
$\mathbf{P}_{cand} \leftarrow \{P | P \in \mathbf{P}, d(k^*, P) < d^* + d(k, q)\}$ ▷ Select polytopes satisfying Theorem 5
$P^* \leftarrow \arg\min_{P \in \mathbf{P}_{cand}} d(q, P)$ ▷ Find closest among candidates
return $P^*, d(P^*, q)$

is run offline to construct the necessary data structures. When a query point q is given, `TI-Query` is run to compute the closest polytope P^*. This algorithm is applicable to any metric as long as Theorem 5 holds.

3.4.3 Analysis of the TI Algorithm

The algorithm is guaranteed to not miss the closest polytope by construction. The reason is the closest distance from the query point q to a polytope is upper bounded by $d(P(k^*), q)$. Thus, only polytopes that can possibly be closer than $P(k^*)$ needs to be checked.

Offline complexity of this algorithm is $O(|K||\mathbf{P}| \log |\mathbf{P}|)$ plus the complexity for `BuildFastPointStructure`. This is a consequence of computing and sorting pairwise distances between all the key points and polytopes. In practice, the number of key points can be chosen arbitrarily to avoid large precomputations, but this will negatively impact online performance. In Section 3.5, we set $|K||\mathbf{P}| \leq 10^6$ to limit precomputation time. Insertion is $\mathcal{O}(|\mathbf{P}| \log \mathbf{P})$.

3.5 Performance Evaluation

Two synthetic datasets of zonotopes distributed uniformly and along an axis ("linearly distributed"), and two real datasets of AH-polytopes from R3T (Chapter 2) and MPC [39] were used to evaluate the algorithms. All code for the experiments can be found on GitHub[2]. All tests were performed on a personal computer with i7-7820HQ CPU.

3.5.1 Synthetic Dataset Scalability Evaluation

Precomputation Scalability

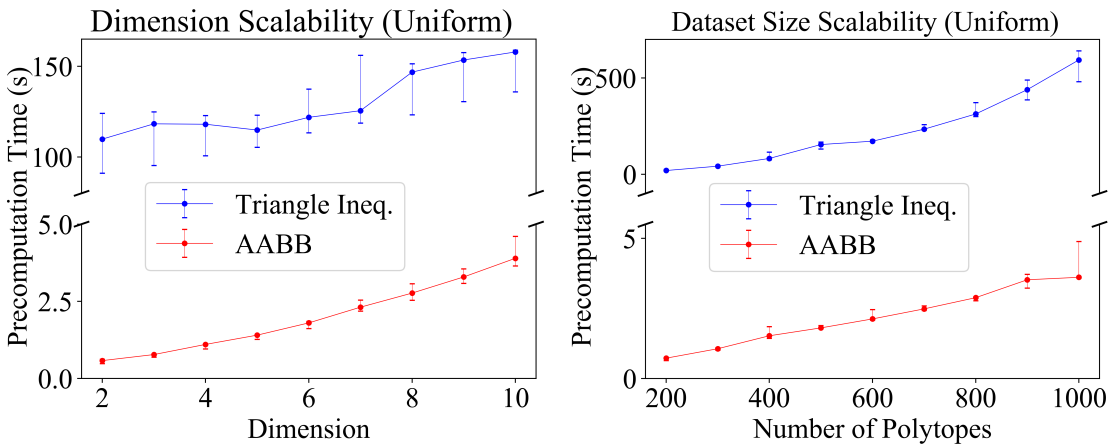
The precomputation performances of the AABB algorithm and the triangle inequality algorithm were tested on synthetic datasets. Specifically, scalability against dimension and dataset size on randomly generated polytopes were tested. Figure 3-2 shows the precomputation performance on uniformly distributed random polytopes, as motivated by sampling-based planning such as [52]. Figure 3-3 shows the precomputation performance on random polytopes distributed along an axis ("linearly distributed"), as motivated by trajectory stabilization techniques such as LQR trees [47]. All experiments were triplicated.

The AABB algorithm is much faster in precomputation. Both algorithms require longer precomputation in higher dimensions due to the increase in complexity of the optimization problems. Note that the number of key points for triangle inequality was limited to $|K||\mathbf{P}| < 10^6$ in all experiments to maintain a reasonable precomputation time.

Online Scalability

The online performances of the AABB and triangle inequality algorithms were tested against dimension and dataset size. Performance was measured with the number of polytope distances evaluated on each query. Figure 3-4 shows the query performance on uniformly distributed random polytopes. Figure 3-5 shows the query performance on random polytopes distributed along an axis ("linearly distributed").

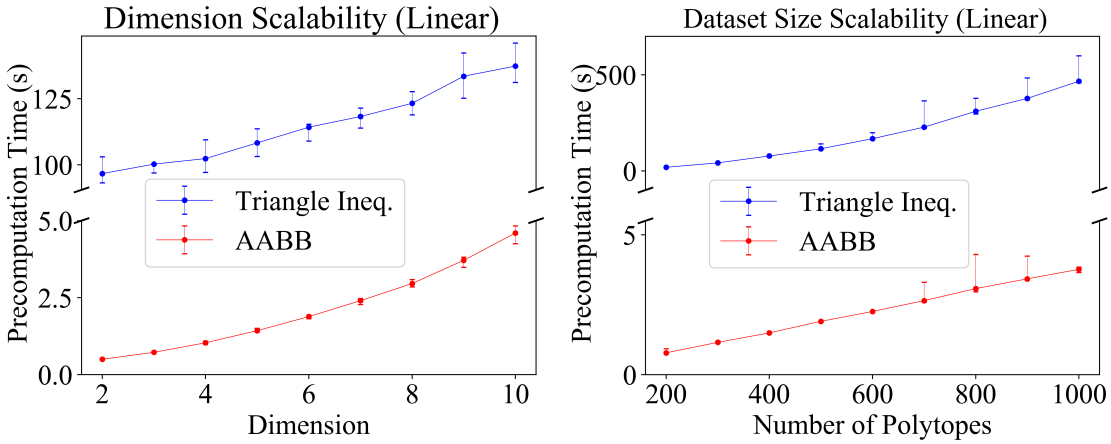
[2] https://github.com/wualbert/closest_polytope_algorithms.git



(a) Precomputation time dimension scalability with 500 polytopes.

(b) Precomputation time dataset size scalability with polytopes in 6D.

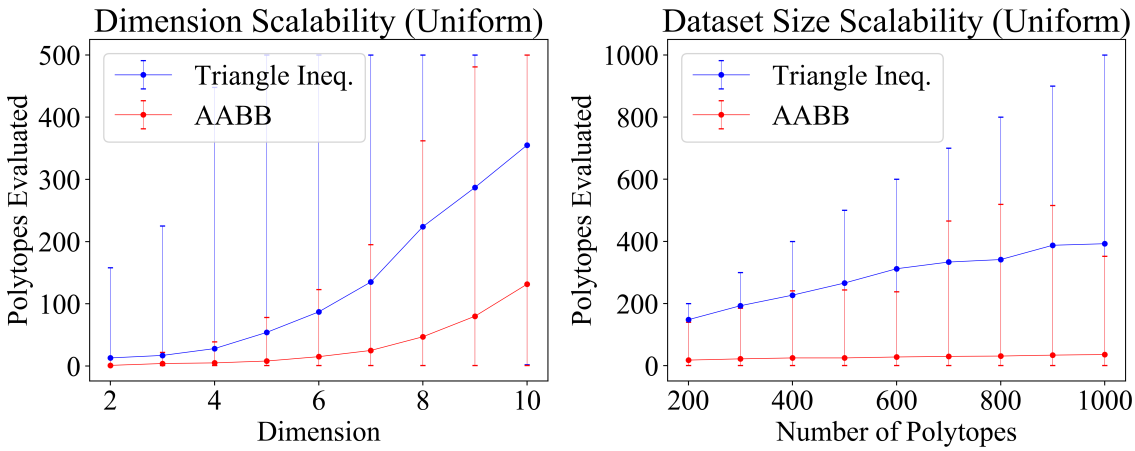
Figure 3-2: Precomputation time of the nearest polytope algorithms with uniformly distributed random polytopes.



(a) Precomputation time dimension scalability with 500 random polytopes.

(b) Precomputation time dataset size scalability with random polytopes in 6D.

Figure 3-3: Precomputation time of the nearest polytope algorithms with random polytopes distributed along an axis ("linearly distributed").



(a) Number of polytopes evaluated per query with 500-polytope synthetic datasets.

(b) Number of polytopes evaluated per query with synthetic datasets in 6D.

Figure 3-4: Number of polytopes evaluated by the nearest polytope algorithms per query with uniformly distributed random polytopes.

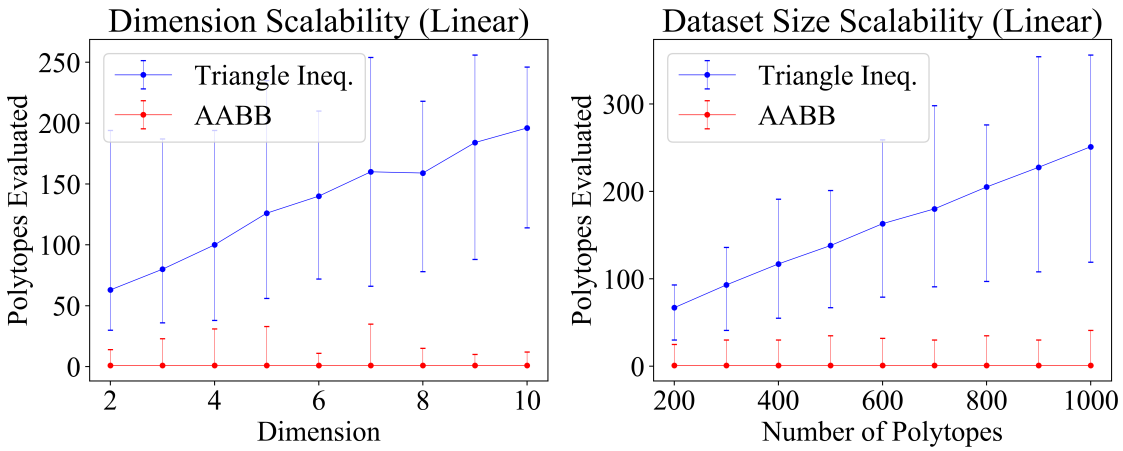
The AABB algorithm outperformed the triangle inequality algorithm in both median and worst-case performances. The worst-case of both algorithms did not scale well with dimension, with both algorithms requiring checking over 40% of the polytopes starting in 7D. With the dataset size, scaling performance is much better in AABB. Section 3.5.2 contains more discussions on AABB complexity.

Notably, these performances are highly dependent on the structure of the dataset and the query point choice. For instance, if a query point is very far from all of the polytopes, both algorithms will check nearly all of the polytopes. This dependency on the specific query point is supported by the much smaller median values.

3.5.2 Real-World Dataset Performance Evaluation

Performance on R3T Datasets

The AABB and triangle inequality algorithms were tested on real polytope data from R3T (Chapter 2). These polytopes are reachable sets generated by motion planning in pendulum swing-up (2D continuous system) and planner hopper (10D, 2 contact modes) problems [35, 46]. In this context, only the query performance is relevant since the polytopes are added sequentially during runtime as R3T explores the state



(a) Number of polytopes evaluated per query with 500-polytope synthetic datasets.

(b) Number of polytopes evaluated per query with synthetic datasets in 6D.

Figure 3-5: Number of polytopes evaluated by the nearest polytope algorithms per query with random polytopes distributed along an axis.

space. Figure 3-6 summarizes the results.

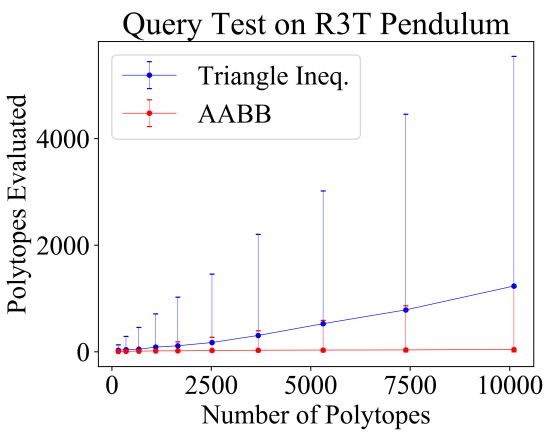
The AABB algorithm yielded significantly better performance on R3T datasets. We attribute this result to the process of generating these polytopes. In R3T, the polytopes are generated through simulating forward dynamics for a specific time horizon [52]. Therefore, the polytopes seldom possess elongated shapes that are unfavorable for AABB approximations. With the AABB algorithm, often less than 5% of all polytopes were checked online.

Performance on MPC Datasets

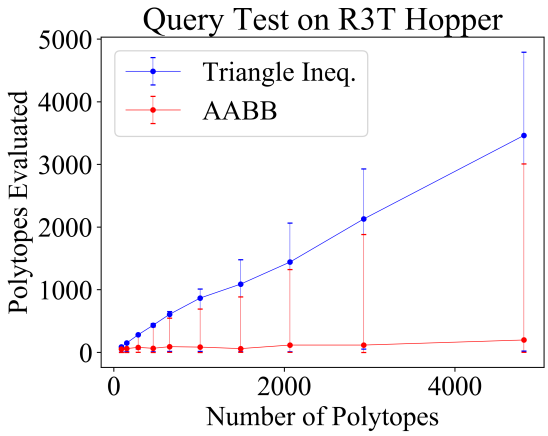
The AABB and triangle inequality algorithms were tested on two datasets generated by MPC [39]. The first dataset originates from stabilizing an inverted pendulum by bouncing against a vertical wall (2D, 2 contact modes) [30]. The second dataset represents manipulating a rod with 2 fingers[3] (10D, 16 contact modes). Similar to RRT, we considered different exploration stages of the MPC problem. The results are summarized in Figure 3-7.

With MPC, the AABB algorithm still outperformed the triangle inequality algo-

[3]The scripts and details of this example are available at https://github.com/sadraddini/PWA-Control.git

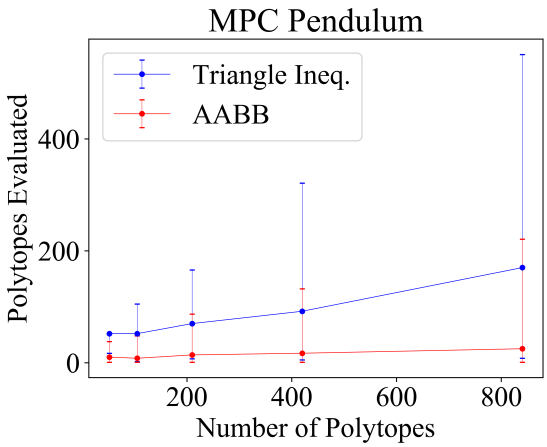


(a) Number of polytopes evaluated per query on the R3T pendulum dataset.

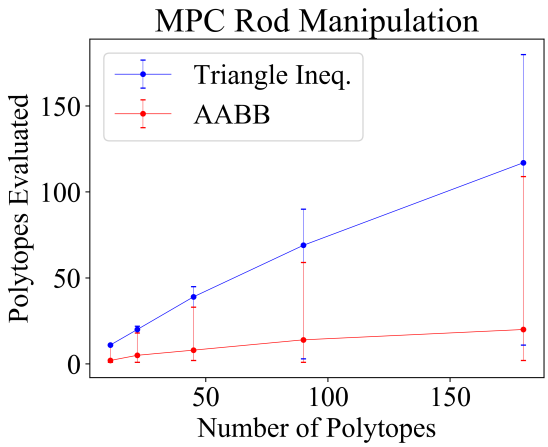


(b) Number of polytopes evaluated per query on the R3T hopper dataset.

Figure 3-6: Testing results of the nearest polytope algorithms on R3T datasets.

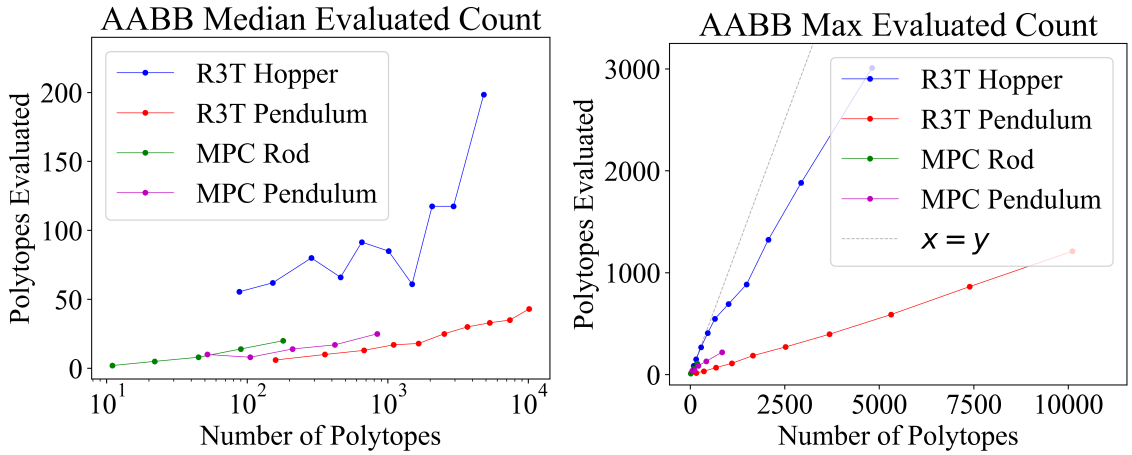


(a) Number of polytopes evaluated per query on the MPC pendulum dataset.



(b) Number of polytopes evaluated per query on the MPC rod manipulation dataset.

Figure 3-7: Testing results of the nearest polytope algorithms on MPC datasets.



(a) Median number of polytopes evaluated per query. This is roughly logarithmic to the size of the dataset.

(b) Maximum number of polytopes evaluated per query. This is roughly linear to the dataset size.

Figure 3-8: Empirical complexity analysis of the AABB algorithm.

rithm by a significant margin. Using the AABB algorithm, typically less than 20% of all polytopes were evaluated, which is a significant improvement from the naive algorithm.

Empirical Online Complexity of the AABB Algorithm

Figure 3-8 shows the AABB algorithm's empirical online complexity in R3T and MPC datasets. The median computation time is roughly logarithmic to the number of polytopes, while the worst-case is linear. However, even in the worst case, only around 50% of all polytopes were evaluated.

3.6 Chapter Summary and Future Work

In this chapter, two algorithms for efficiently solving the nearest polytope problem are proposed and discussed. Through testing on synthetic, R3T, and MPC datasets, it is concluded that the AABB algorithm is the superior. Both algorithms provide significant querying performance improvements over the naive exhaustive search, with AABB demonstrating roughly logarithmic median performance empirically. The algorithms developed can be applied to control applications that require solving the

nearest polytope problem. The AABB algorithm was leveraged in Chapter 2 to accelerate the R3T algorithm.

Future work will focus on improving the heuristics, combining the algorithms, and further implementations. For instance, while the methods discussed in this chapter are purely geometrical, exploiting the underlying dynamics of the polytope data source (such as hybrid system dynamics) can potentially lead to performance improvements.

Chapter 4

Learning a Manipulation Policy from Sampling-based Planning

4.1 Background and Motivation

Traditional path planning algorithms often have poor performance when applied to robot manipulation problems. This is a consequence of two major obstacles: complexity and brittleness. In contact rich systems, different hybrid modes produce combinatorial complexity in the system. Having such a large number of modes results in brittleness. Each dynamic mode may require a different tracking policy, which greatly reduces the error tolerance.

A popular approach to circumvent these challenges in the locomotion setting is to perform trajectory optimization with predetermined contact modes [34, 10]. However, the same approach does not generalize well to manipulation. Unlike locomotion, where a periodic mode sequence is often available, the mode sequences in manipulation problems are seldom obvious [34]. In [34], the authors formulated the contact-rich trajectory optimization as a mathematical program with complementarity constraints to avoid an explicit mode schedule. In [10], to avoid specifying a mode schedule, the authors used a smooth contact model to obtain an unconstrained, continuous trajectory optimization problem where all modes can be considered. Nevertheless, these methods are often too slow to react to disturbances online, and they lack global

optimality guarantees due to the non-convex nature of the optimization formulations [8]. One way to obtain global optimality is to formulate discrete modes as a mixed-integer program, which can then be solved by branch-and-bound [8]. Yet solving for global optimality may be prohibitively slow for online planning, and the policies found at runtime may not be consistent among each other [8, 49].

More recently, learning methods have begun to emerge as a candidate for manipulation planning. [32] presents a Rubik's cube solving robot trained in simulation via automatic domain randomization. A review of robot manipulation with learning methods can be found in [22]. The common pitfall of learning based approaches are large sample complexities and task specificity. To avoid these issues, one way is to incorporate system models into the learning process. In [8], the authors proposed "LVIS" for creating contact-aware controllers. This approach involves learning the optimal value function with costs-to-go generated by partially-solved mixed-integer programs offline. Online, the learned costs-to-go are used to rapidly generate a one-step model-predictive controller to control the system. In Guided Policy Search (GPS) [27], the challenge with high sample complexity is tackled by assisting policy learning with trajectory optimization. While GPS can effectively direct policy learning and avoid poor local optima, there is no guarantee on the solution completeness under GPS' exploration strategy. Furthermore, in both LVIS and GPS, each trajectory optimization problem solved offline is independent and does not inform one another. This leads to poor handling of scenarios with multiple optimal policies. Each optimization may select a different optimal policy, leading to inconsistencies especially during learning. As a comparison, graph-based methods such as RRT* avoid this issue by utilizing all previously-computed policies. Through the tree extension and rewiring processes, the policies obtained previously, which are encoded in the topology of the tree, are used to tie-break the newly-obtained policies. Only the policy that establishes a tree edge to the best neighboring node will be chosen.

In this chapter, a novel framework that aims to address the shortcomings of existing robot manipulation planning techniques is introduced. The discussion focuses on a planar 3-finger robot manipulator. Using a model of the system, RRT* [17] first

generates a manipulation policy offline to move an object from explored initial conditions to a specified target pose. A neural network is then trained to approximate the policy obtained from RRT*. The ultimate goal is to produce a dense policy with fast online evaluation for manipulating objects starting from arbitrary initial conditions.

4.2 Problem Definition

4.2.1 Hardware Setup

In this chapter, the robot system of interest is the "planar gripper" shown in figure 4-1. The planar gripper has 3 fingers, each with 2 degrees of freedom. All motions of the fingers and the object are constrained on a horizontal 2D plane. No gravity is present. The object is assumed to be a polygon with known geometry. A quasi-static model is used in this chapter, which leads to the state definition in Definition 6. The quasi-static model is further discussed in Chapter 4.3.

Definition 6. *The 9D state x of the planar gripper system is given by*

$$x = [q_{finger}\ y\ z\ \theta]^T \tag{4.1}$$

where $q_{finger} \in \mathbb{R}^6$ contains the position of the 6 finger joints and $y, z, \theta \in \mathbb{R}$ describe the pose of the object.

The planar gripper system has a motion capture system to accurately measure the state x. Each fingertip has a force sensor for measuring contact forces. A low level controller developed by the Toyota Research Institute (TRI) is available to track the given motion plans, which consist of finger positions and contact configurations.

4.2.2 Task Definition

This chapter aims to solve the following problem.

Problem 3. *Given a goal state x_G offline, compute a policy such that when given a initial state x_0 online, the policy can produce a motion plan to reach x_G.*

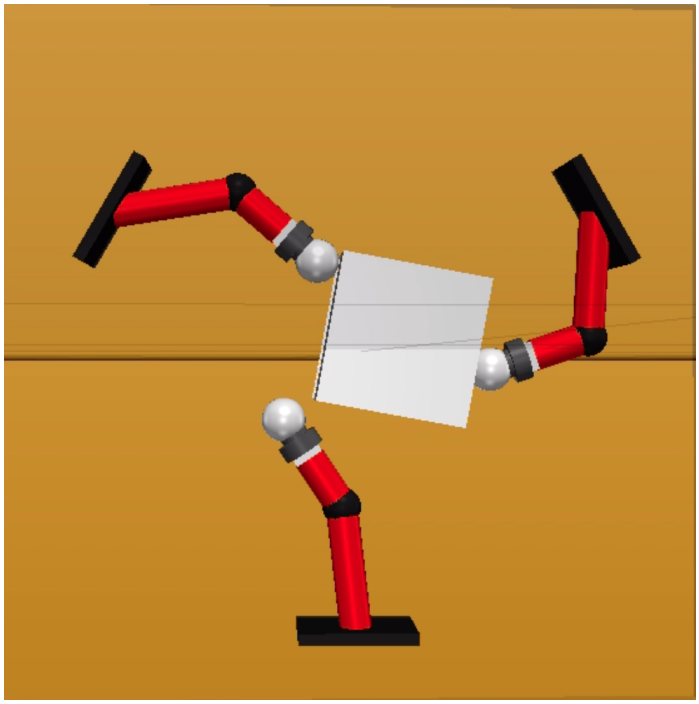


Figure 4-1: The planar gripper system in simulation with a square object. Each of the fingers have 2 joints and a force sensor at the fingertip.

The main distinction between the offline and online phases is that the offline phase has no restriction on computation time, whereas the online phase requires fast computation.

4.2.3 Software Setup

All programs in this chapter were written in `Python` using the `Python` bindings of `Drake`[1], `pydrake`[2]. The optimization problems in Chapters 4.4.2 and 4.4.3 were implemented using the `pydrake` bindings of `SNOPT`. The neural networks in Chapter 4.5 were implemented with `Pytorch`[3].

[1]https://drake.mit.edu/
[2]https://drake.mit.edu/pydrake/index.html
[3]https://pytorch.org/

4.3 System Modeling

First, the modeling of the planar gripper system is discussed. This system model is used by the RRT* planner in Chapter 4.4 for generating motion plans.

4.3.1 The Quasi-Static Model

Throughout this chapter, a quasi-static dynamics model is adopted for the planar gripper. This eliminates the kinodynamic constraints and simplifies the system. The model is justifiable as the planar gripper moves at a relatively low speed.

4.3.2 Motion Cones

Under the quasi-static model, the reachable set of the object given a specific set of contact configurations can be computed via the motion cone approach described in [6]. Here, a brief summary of the motion cone computation is provided.

By the quasi-static assumption, the total force on the object is always zero. Therefore,

$$\sum_{i=1,2,3} {}^{B}f^{c_i} + {}^{B}f_{table} = 0, \tag{4.2}$$

$$\sum_{i=1,2,3} {}^{B}p^{c_i} \times {}^{B}f^{c_i} + \tau_{table} = 0. \tag{4.3}$$

${}^{B}f^{c_i} \in \mathbb{R}^2$ denotes the force from finger i, which contacts the object at c_i, in the object body frame. ${}^{B}p^{c_i} \in \mathbb{R}^2$ is the moment arm of contact position c_i. $f_{table} \in \mathbb{R}^2$ and $\tau_{table} \in \mathbb{R}$ are the friction force and torque from the table in the object body frame.

Assuming each contact force ${}^{B}f^{c_i}$ lies within a friction cone FC_i, the set of wrench W_i generated by each friction force is a cone. The summation of W_i, which is the total wrench generated by all fingers, should be equal to the wrench generated by the

table W_{table}. Hence,

$$\begin{aligned} W_{table} = \{[^W f_{table}, \tau_{table}]^T \mid \exists {}^B p^{c_i} \in FC_i \\ s.t.\ {}^W R^B(\theta) \sum_{i=1,2,3} {}^B f^{c_i} + {}^W f^{table} = 0, \sum_{i=1,2,3} {}^B p^{c_i} \times {}^B f^{c_i} + \tau^{table} = 0\}. \end{aligned} \tag{4.4}$$

The notation ${}^W(\cdot)$ denotes the world frame representation of vector quantity $(\cdot)$.

Additionally, from the limit surface model,

$$ {}^W f_{table}^T \cdot {}^W f_{table} + \tau_{table}^2/(rc)^2 = (\mu mg)^2. \tag{4.5}$$

r is the equivalent radius of the object, c is a constant typically around 0.6, μ is the dynamic friction coefficient, m is the mass of the object, and g is the gravity constant. Under the maximal dissipation assumption, the table force ${}^W f_{table} \in \mathbb{R}^2$, torque $\tau_{table} \in \mathbb{R}$, and the sliding velocity ${}^W v := [\dot{y}\ \dot{z}]^T \in \mathbb{R}^2$ should satisfy

$$\begin{aligned} {}^W f_{table} &= -{}^W v \cdot \frac{\mu mg}{\sqrt{{}^W v^T \cdot {}^W v + (rc)^2 \dot{\theta}^2}}, \\ \tau_{table} &= -\dot{\theta} r^2 c^2 \cdot \frac{\mu mg}{\sqrt{{}^W v^T \cdot {}^W v + (rc)^2 \dot{\theta}^2}} \end{aligned} \tag{4.6}$$

based on the derivation in [6].

Regard $k := \frac{\mu mg}{\sqrt{{}^W v^T \cdot {}^W v + (rc)^2 \dot{\theta}^2}}$ as a scaling factor, Equation (4.6) provides the relationship between the object velocity and the table force

$$ [^W f_{table}\ \tau_{table}]^T = k \cdot diag(1, 1, r^2 c^2) \cdot [^W v\ \dot{\theta}]^T, \tag{4.7}$$

where $diag(1, 1, r^2 c^2) \in \mathbb{R}^{3\times 3}$ is the diagonal matrix with diagonal entries $1, 1, r^2 c^2$. Finally, leveraging the fact that $[^W f_{table}\ \tau_{table}]^T \in W_{table}$, the polyhedral cone given in Equation (4.4), the possible object velocities $\left\{[^W v\ \dot{\theta}]^T\right\}$ can be described by the object motion cone W_v.

$$ \left\{[^W v\ \dot{\theta}]^T\right\} = W_v = diag(1, 1, 1/(r^2 c^2)) \cdot W_{table}. \tag{4.8}$$

In practice, given a object pose $[y, z, \theta]^T$ and contact configurations c_1, c_2, c_3, the feasible new positions of the object can be computed by an Euler integration argument

$$\left\{[y^+, z^+, \theta^+]^T\right\} = [y, z, \theta]^T + W_v. \tag{4.9}$$

Notice the time step size is absorbed into the cone representation W_v. Likewise, the *backward* motion cone, which represents the set of poses that can lead to the current pose, can be computed using the negated version of W_v. The backward motion cone is relevant to Chapter 4.4.1.

4.3.3 Contact Modeling and Transition

Since the object is assumed to be a polygon with known geometry, the contacts between the object and the fingers are specified explicitly. In particular, each finger i can be in contact with face j of the object at a position ${}^B c_i \in face_j$. The fingers can also be out of contact with the object.

Between each pair of discretized *waypoints* during planning, the fingers are allowed to make or break contact with the object independently of each other. However, switching between faces across adjacent waypoints are not allowed. As an example, a finger cannot jump from contacting face j_1 to j_2 between adjacent waypoints w_1 and w_2. Instead, the finger must break contact from j_1 during the transition from w_1 to w_2, then make contact with j_2 on the next waypoint w_3.

4.4 Obtaining a Manipulation Plan with RRT*

To solve Problem 3, an manipulation policy is first generated offline via RRT*. In particular, the following problem is solved.

Problem 4. *Assume a starting state x_G is given. For as many states in the state space as possible, obtain a manipulation policy that can lead the system to x_G.*

The policy is then approximated using a neural net, as discussed in Chapter 4.5.

4.4.1 The Backward Tree

In order to efficiently leverage the RRT* exploration, the "backward" tree formulation is considered. Using the goal state x_G, the standard RRT* is performed with the reversed-time system dynamics. This way, through backtracking from a leaf node to the root node in the obtained RRT* tree, a path in forward time from the state at the leaf to x_G can be retrieved.

Under this formulation, every new state explored by RRT* represents a new state that leads to x_G, as opposed to the standard forward-time tree which produce a single-source solution from the initial state x_0 to x_G. Moreover, given that a neural net is ultimately trained to provide online policy evaluation, using the backward tree allows for generating training datasets across the state space offline for the neural net to approximate.

4.4.2 RRT* Formulation

Due to the choice of a quasi-static model, there are no differential constraints and Problem 4 can be solved with purely geometric RRT* [23, 18]. Recall that in geometric RRT*, a sample is taken in the state space, and the nearest node is identified. An "extension" procedure is then performed to grow the tree from the nearest node to the sample state. The tree is then rewired to achieve asymptotic optimality. In order to ensure the path obtained from extension is indeed feasible, a trajectory optimization problem is solved on every extension and rewiring. The objectives and constraints of the optimization are discussed below.

Waypoints and Nodes

For a more granular enforcement of the constraints, an additional level of discretization is introduced in the RRT* tree. Between each pair of nodes in the tree, there are $2\tilde{4}$ additional *waypoints*. The nodes are also waypoints themselves. Waypoints are solely used for trajectory optimization and the RRT* tree can only branch off nodes. All constraints discussed below are enforced on the waypoint level.

Distance Metric

For both extension and rewiring, the squared weighted L2 norm given in Equation (4.10) is used for distance computation.

$$\begin{gathered} dist(x_1, x_2) := (x_1 - x_2)^T M (x_1 - x_2) \\ x_1, x_2 \in \mathbb{R}^9, M \in \mathbb{R}^{9\times 9} \text{ is a diagonal matrix with positive entries.} \end{gathered} \tag{4.10}$$

For the coordinates of x that represent angles, a "wrap-around" is performed. In other words, if the ith coordinate is an angle,

$$|x_1[i] - x_2[i]| := \min\left\{|x_1[i] - x_2[i]| \bmod (2\pi), 2\pi - |x_1[i] - x_2[i]| \bmod (2\pi)\right\}. \tag{4.11}$$

If waypoints exist between the states of interest, the distance is calculated as the sum of the pairwise distances between adjacent waypoints.

Motion Cone Constraints

The motion cone constraints detailed in Chapter 4.3.2 specify the directions the object can move toward. In the RRT* setup, this is enforcing Equation (4.9). However, due to the backward time formulation, the constraint is instead

$$\left\{[y^-, z^-, \theta^-]^T\right\} = [y, z, \theta]^T + \bar{W}_v, \tag{4.12}$$

where $\bar{W}_v$ is the backward motion cone computed from $[y, z, \theta]^T$. Notice that this is a linear constraint as the motion cone is a polyhedral cone.

Rolling Constraints

Pure rolling between the fingers and the object is assumed, so there is no slip between the fingertips and the object. Notably, this requires the contact position to change as the object moves because the fingertips have nonzero radii.

Displacement Bounds

The motion cone is an *instantaneous* reachable direction from a given state. Under the pure rolling assumption, the contact points change as the object moves, modifying the motion cone in the process. However, as an approximation, only the initial motion cone constraint is enforced. To ensure the quality of the approximation, the displacement between waypoints x_1, x_2 are bounded by

$$|x_1 - x_2| \leq \delta x_{max} \tag{4.13}$$

where $\delta x_{max} \in \mathbb{R}^9$ and the $\leq$ is interpreted element-wise.

Planar Gripper Feasibility Constraints

The planar gripper joint limits are enforced. In addition, no collision is allowed except between the fingertips and the object.

Contact Modes

The contact model and transition rules described in Chapter 4.3.3 are applied on the waypoint level. Moreover, the mode switch between node pairs is artificially constrained to between the child node (the node *farthest* away from the root node) and its immediately adjacent preceding waypoint. As a concrete example, consider an edge on the RRT* tree with 3 waypoints between the parent and child nodes, `parent-w1-w2-w3-child`. `parent`, `w1`, `w2`, `w3` must share an identical contact mode, while `child` can have a different contact mode as long as the mode switch between `w3` and `child` obeys the rules given in Chapter 4.3.3.

In order to account for the motion cone change during mode switches, a more conservative motion cone is computed on waypoint pairs with mode switches. Denote not making contact as $\emptyset$. For two contact mode vectors m_1, m_2, the *intersection*

$m_1 \cap m_2$ is defined element-wise as

$$m_{1i} \cap m_{2i} = \begin{cases} m_{1i}, & m_{1i} = m_{2i}, \\ \emptyset, & m_{1i} = \emptyset \vee m_{2i} = \emptyset, \\ \text{undefined}, & \text{otherwise.} \end{cases} \tag{4.14}$$

The undefined case never happens if the contact transition rules in Chapter 4.3.3 are obeyed. Continuing on the example, Equation (4.14) will be applied to compute the motion cone constraint between `w3-child`. This way, the trajectories computed is guaranteed to be feasible even if the contact modes are changing.

The Complete Optimization Problem

The complete optimization problem solved on each extension is the nonlinear optimization problem in Equation (4.15).

$$\begin{aligned} &\min \mathit{dist}(\text{sample}, \text{nearest node}) \\ &s.t. \text{ constraints from Chapter 4.4.2} \end{aligned} \tag{4.15}$$

using the distance metric and constraints described above.

The complete optimization problem solved on each rewiring is a two-point boundary value problem given in Equation (4.16), with the boundary values being the states of `potential parent`, `new node` when rewiring a new node's parent, and `new node`, `potential child` when evaluating the new node as other nodes' parent. For more details on the RRT* rewiring procedure, please refer to Algorithm 6 of [17].

$$\begin{aligned} &\min \mathit{dist}(\text{boundary 1}, \text{boundary 2}) \\ &s.t. \text{ constraints from Chapter 4.4.2.} \end{aligned} \tag{4.16}$$

While the boundary values are fixed, the waypoints between the two boundary values are free to move and are the decision variables in this problem.

4.4.3 RRT* Tree Post Processing

While RRT* guarantees asymptotically optimality in theory, practically only finite samples are drawn. As a result, rewiring RRT* may still yield suboptimal behaviors. The problem is especially significant due to the nonlinear optimization problem described in Chapter 4.4.2. The chance of finding a node suitable for rewiring while satisfying all the constraints is slim, and RRT* tends to find suboptimal solutions in practice. For instance, the fingers out of contact often "dance" around. Therefore, a post processing procedure is performed on the RRT* tree. Specifically, this procedure extracts the paths from each of the leaf nodes to the root node using depth first search (DFS) [45]. The paths are then "smoothed" in a large optimization problem. For a path containing n nodes, Equation (4.17) specifies the optimization setup.

$$\begin{aligned} \min \quad & \sum_{i=1,2,\ldots,n-1} dist(\text{node}_i, \text{node}_{i+1}) \\ s.t. \quad & \text{constraints from Chapter 4.4.2,} \\ & \text{all waypoints retain their original contact configurations.} \end{aligned} \tag{4.17}$$

Here "original contact configuration" refers to the fingers making contact with the same *faces* of the object, as opposed to keeping the contact locations ${}^{B}c_i$ identical. Essentially, Equation (4.17) is a contact-explicit trajectory optimization problem with the contact modes obtained from RRT*.

The outcome of the post processing procedure is a list of paths from a state explored by RRT* to the goal state x_G. A neural network is then used to approximate the policies encoded by these paths, as detailed in Chapter 4.5.

4.5 Learning a Policy from RRT*

A neural network was implemented to approximate the policy produced by RRT* from Chapter 4.4.

Neural Network Input-Output

The neural network takes in a state and predicts the next state in forward time. Both the 9 dimensional continuous state and the 3 dimensional discrete contact configuration (finger-object face mapping) are fed into the network, yielding a 12 dimensional input. The neural network has 2 outputs: the 9 dimensional desired state, and the desired contact configuration of dimension $3 \times (1 + \text{number of object faces})$. Each entry of the latter output represents the log-likelihood of a finger making contact with one of the faces or not making contact on the next step.

During training and validation, the datasets are generated by extracting waypoint pairs from the RRT* tree and formatted into the required structure for the neural network.

Neural Network Structure

The neural network is fully connected with leaky ReLU activation. There are 4 layers including the input and output layers, with input-output dimensions 12×32, 32×64, 64×64, and $64 \times (9 + 3 \times (1 + \text{number of object faces}))$.

Loss Functions

A mean squared error loss is imposed on the outputs corresponding to the next state. A negative log likelihood loss is imposed on the outputs corresponding to the contact mode prediction.

4.6 Results and Discussions

4.6.1 RRT* Tree Generation

Figure 4-2 shows an instance of the RRT* tree generated for the planar gripper system. In this particular instance, the object is an equilateral triangle. As expected from the symmetry of the object and the planar gripper system, the RRT* tree generated is symmetric.

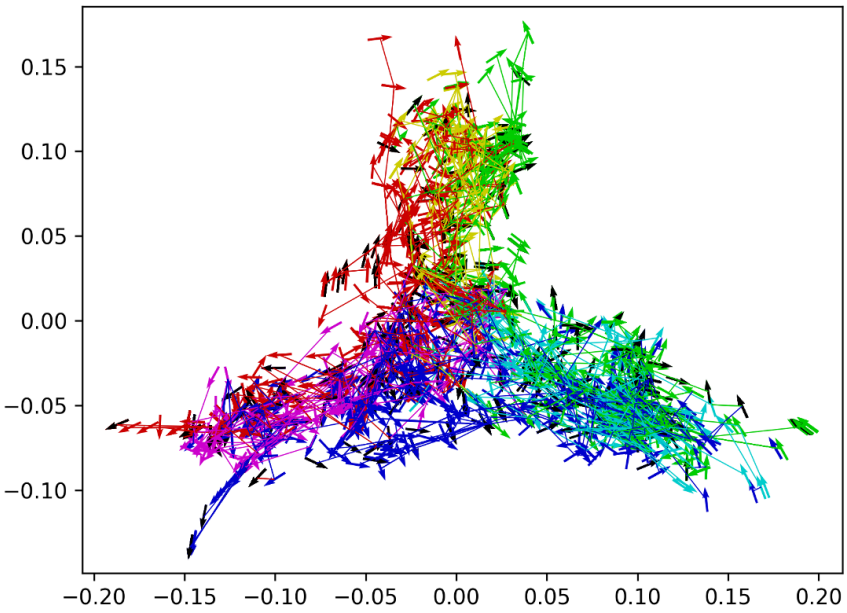


Figure 4-2: Planar gripper RRT* tree. The horizontal and vertical axes are meters away from an arbitrary origin. Each arrow represents the location of the object at a tree node, with the position and orientation of the arrow corresponding to the position and orientation of the object. Each color denotes a contact mode.

4.6.2 RRT* Plan Playback

Figure 4-3 shows a series of snapshots of playing back a plan found by RRT* in `Drake`'s simulator. The plan is played back in forward time, which means starting from a leaf node and ending at a root node. From the plan playback, it is evident that RRT* can plan complex mode sequences.

4.6.3 RRT* Tree Post Processing

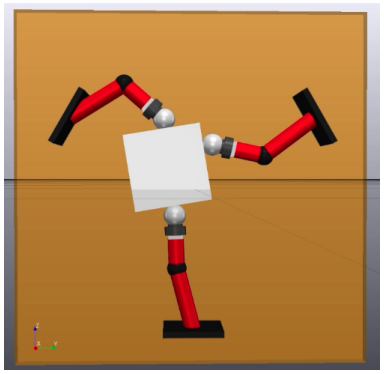
The result of smoothing the tree, as described in Chapter 4.4.3, is demonstrated in Figure 4-4. In this figure, the plans before and after smoothing are compared side-to-side.

After smoothing, much of the redundant finger "dancing" motion was eliminated, showing that the post-processing step effectively improves the quality of the RRT* plans. However, the smoothing process did introduce challenges during implementation, as described below.

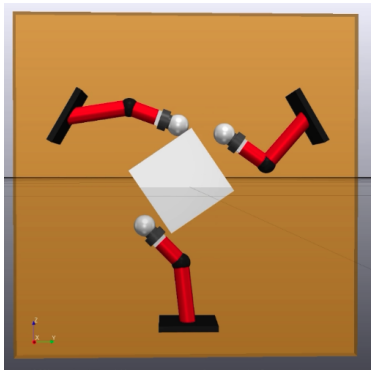
Issues with Nonlinear Optimization

In general, there are no guarantees on finding a solution to a nonlinear optimization problem. The optimization problem in Equation (4.17) is particularly challenging to solve. One major culprit is the distance calculation for collision constraints. In the model used in this thesis, the planar gripper fingers are modeled as cylinders, while the object is modeled as a polygon. However, the underlying `FCL`[4] library used for distance calculation approximates cylinders as polygons, which often lead to approximation errors and ultimately failures from the `SNOPT` solver. A potential solution is to model the fingers as a polygon instead. As the discussion in this chapter is limited to 2D and the projection of a cylinder onto a 2D plane is a rectangle, a polygonal approximation is reasonable.

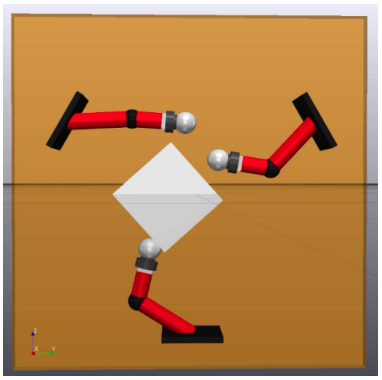
[4]https://github.com/flexible-collision-library/fcl



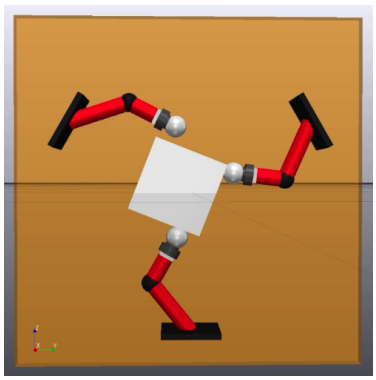
(a) RRT* plan-1



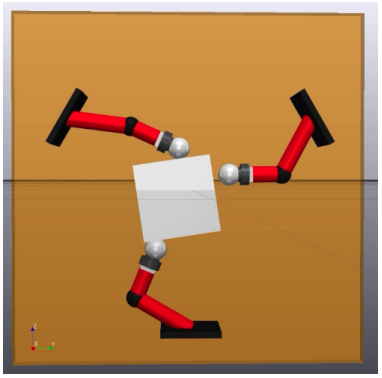
(b) RRT* plan-2



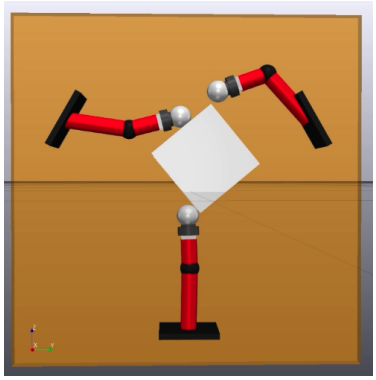
(c) RRT* plan-3



(d) RRT* plan-4

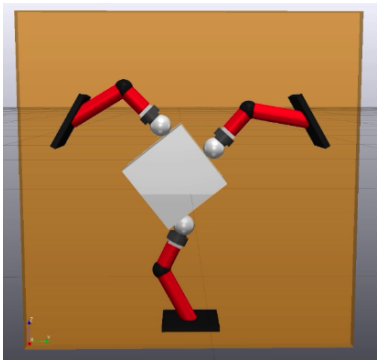


(e) RRT* plan-5

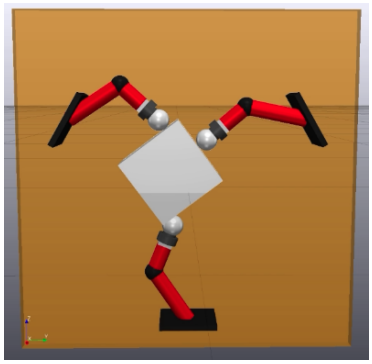


(f) RRT* plan-6

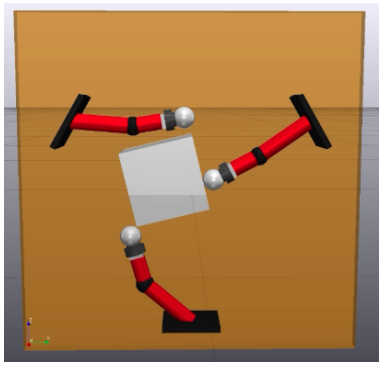
Figure 4-3: Playback of an RRT* plan. The fingers switch between contact modes to move the object to the desired position. The object position in Figure 4-3f would have not been attainable had the fingers kept the contact mode in Figure 4-3a.



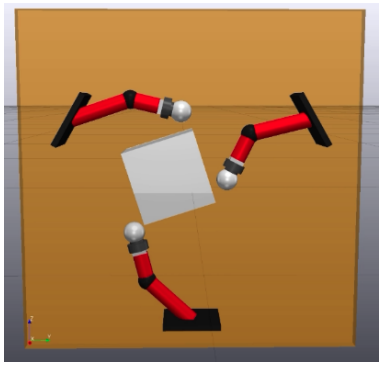
(a) RRT* plan before smoothing-1



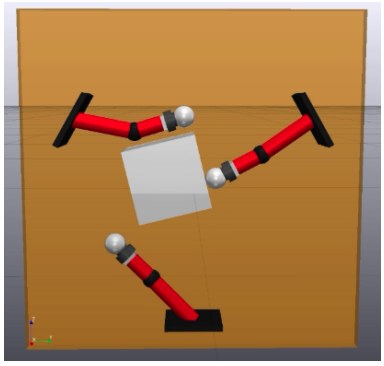
(b) RRT* plan after smoothing-1



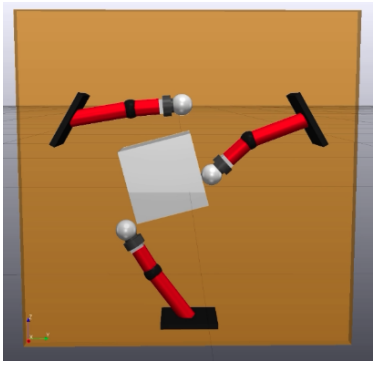
(c) RRT* plan before smoothing-2



(d) RRT* plan after smoothing-2



(e) RRT* plan before smoothing-3



(f) RRT* plan after smoothing-3

Figure 4-4: Comparison of an RRT* plan before and after smoothing. Figures 4-4a, 4-4c, 4-4e are before smoothing, whereas Figures 4-4b, 4-4d, 4-4f are after smoothing. Smoothing eliminated the unnecessary finger movements, such as the top-left finger bending upwards in Figure 4-4c, and the bottom finger swinging to the left in Figure 4-4e.

(a) Hypothetical RRT* tree before smoothing.

(b) Hypothetical RRT* tree after smoothing.

Figure 4-5: Illustration of possible loss of probability completeness after smoothing. The grey polygons are obstacles, x_G is the root node of the tree, $x_{L1}, x_{L2}, x_{L3}, x_{L4}$ are leaf nodes. Prior to smoothing, x connects x_{L1} and x_{L2} to x_G, and the RRT* policy is defined in the vicinity of all the nodes. After smoothing with the procedure described in Chapter 4.4.3, x may be replaced by x_1 and x_2. The original x is lost and the RRT* policy no longer covers regions around x. As x_1, x_2 are close to x_{L3}, x_{L4}, they provide less information for policy learning than x. Notice that under the smoothing procedure, the total number of nodes on each path is not changed.

Loss of Probability Completeness After Smoothing

A potential issue with smoothing using the procedure in Chapter 4.4.3 is the loss of RRT*'s probability completeness, as illustrated in Figure 4-5. One way to circumvent this issue is to perform smoothing for paths starting at every node of the tree instead of just the leaf nodes. However, despite being an offline procedure, the computation cost of doing so could still be prohibitively high. The feasibility of smoothing every node will have to be determined empirically.

4.6.4 Neural Network Training

Figure 4-6 shows the training result of the neural network on a 1000-node RRT* tree with no smoothing. The decreasing training and validation losses suggest that the neural network is able to approximate the policy from the RRT* tree.

Nevertheless, it is worth mentioning that the training loss is not completely representative of the neural network policy approximation. For instance, if the loss results from an incorrect prediction of a crucial mode switch, a single prediction error could break the entire trajectory. The true validity of the neural network policy can only

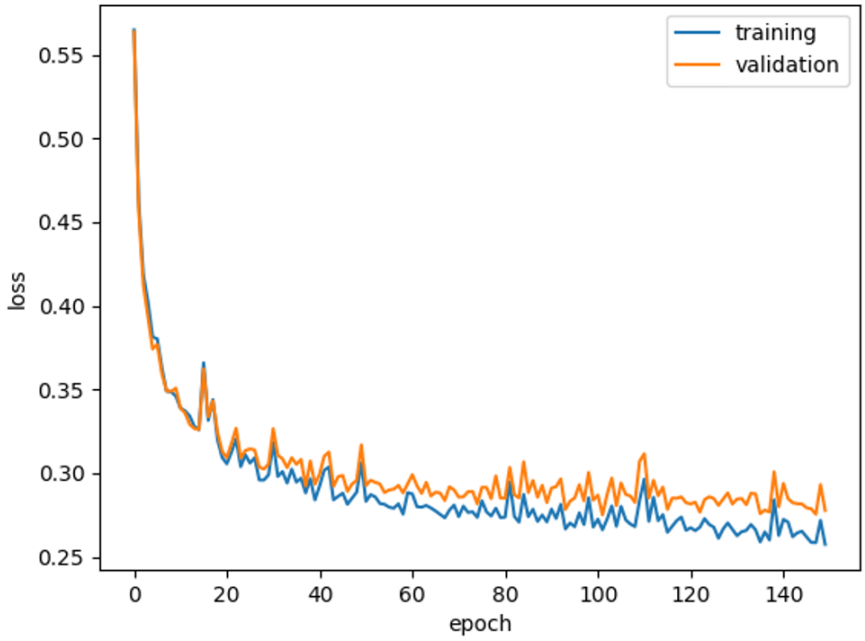


Figure 4-6: Neural network policy approximation on a 1000-node RRT* tree without smoothing. 1000 epochs were run with a batch size of 100.

be verified via simulation.

4.7 Chapter Summary and Future Work

In this chapter, a novel paradigm for generating a manipulation policy to a given target position is proposed. During the offline stage, a backward RRT* tree is computed to provide sparse motion plans for explored states throughout the state space. A neural net is then trained to approximate the policy obtained from RRT*. Online, the neural net should be queried with the system's current state and contact configuration to produce a target next-step state and contact configuration for a low-level controller to track. Through performing trajectory optimization during the extension and rewiring stages of RRT*, motion plans were successfully obtained from RRT*. A post-processing step via contact-explicit trajectory optimization smoothed the RRT* tree and visibly improved the RRT* plan. Preliminary testing of the neural network on RRT* trees suggested that extracting a policy from the RRT* plan is indeed

possible.

Moving forward, besides addressing the challenges described in Chapter 4.6, more investigation can be done on the neural network approximation, including trying different parameters and network architecture. In addition, the RRT* plan and the neural network policy can be validated through physics-based simulation and hardware experiments. Finally, the project can be expanded to include a perception component. Leveraging techniques such as keypoint representations [29] may yield a visuomotor policy with category-level generalization.

Chapter 5

Conclusions and Closing Remarks

Motion planning is a key component for achieving challenging behaviors with robots. Sampling-based planners are powerful candidates for motion planning in hybrid robotic systems as they can potentially circumvent the complexity due to having multiple dynamic modes. This thesis investigated the sampling-based planning stack through introducing "R3T" for sampling-based planning with reachable sets, algorithms for solving the nearest polytope problem, and a framework combining RRT* with neural network policy approximation for robot manipulation planning.

Results from this thesis can improve the understanding of hybrid system planning and robot manipulation. The primary purpose is to provide algorithmic frameworks that can be leveraged to achieve desired applications. In recent years, robots with hybrid dynamics such as quadrupeds have begun to emerge as commercial products. As more sophisticated applications with hybrid systems are developed, sampling-based planning combined with learning methods could provide a direction for developing complex motion plans efficiently.

Bibliography

[1] Farhad Bayat, Tor Arne Johansen, and Ali Akbar Jalali. Using hash tables to manage the time-storage complexity in a point location problem: Application to explicit model predictive control. *Automatica*, 47(3):571 – 577, 2011.

[2] Alberto Bemporad, Francesco Borrelli, and Manfred Morari. Piecewise linear optimal controllers for hybrid systems. In *American Control Conference, 2000. Proceedings of the 2000*, volume 2, pages 1190–1194. IEEE, 2000.

[3] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[4] Michael S Branicky, Michael M Curtiss, Joshua A Levine, and Stuart B Morgan. Rrts for nonlinear, discrete, and hybrid planning and control. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, volume 1, pages 657–663. IEEE, 2003.

[5] Stéphane Caron, Quang-Cuong Pham, and Yoshihiko Nakamura. Completeness of randomized kinodynamic planners with state-based steering. *Robotics and Autonomous Systems*, 89:85 – 94, 2017.

[6] Nikhil Chavan-Dafle, Rachel Holladay, and Alberto Rodriguez. In-hand manipulation via motion cones. *arXiv preprint arXiv:1810.00219*, 2018.

[7] F. J. Christophersen, M. Kvasnica, C. N. Jones, and M. Morari. Efficient evaluation of piecewise control laws defined over a large number of polyhedra. In *2007 European Control Conference (ECC)*, pages 2360–2367, July 2007.

[8] Robin Deits, Twan Koolen, and Russ Tedrake. Lvis: Learning from value function intervals for contact-aware robot controllers. *2019 International Conference on Robotics and Automation (ICRA)*, pages 7762–7768, 2018.

[9] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *Ieee access*, 2:56–77, 2014.

[10] Tom Erez and Emanuel Todorov. Trajectory optimization for domains with contacts using inverse dynamics. *IEEE International Conference on Intelligent Robots and Systems*, pages 4914–4919, 2012.

[11] Roland Geraerts and Mark H Overmars. Reachability analysis of sampling based planners. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 404–410. IEEE, 2005.

[12] Gustavo Goretkin, Alejandro Perez, Robert Platt, and George Konidaris. Optimal sampling-based planning for linear-quadratic kinodynamic systems. In *2013 IEEE International Conference on Robotics and Automation*, pages 2429–2436. IEEE, 2013.

[13] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1984.

[14] Francois Robert Hogan and Alberto Rodriguez. Feedback Control of the Pusher-Slider System: A Story of Hybrid and Underactuated Contact Dynamics. pages 1–16, 2016.

[15] Piet Houthuys. Box sort, a multidimensional binary sorting method for rectangular boxes, used for quick range searching. *The Visual Computer*, 3(4):236–249, Dec 1987.

[16] Léonard Jaillet, Judy Hoffman, Jur Van den Berg, Pieter Abbeel, Josep M Porta, and Ken Goldberg. Eg-rrt: Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2646–2652. IEEE, 2011.

[17] S. Karaman and E. Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. In *The International Journal of Robotics Research*, volume 30, pages 846–894. IEEE, 2011.

[18] Sertac Karaman and Emilio Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *49th IEEE conference on decision and control (CDC)*, pages 7681–7687. IEEE, 2010.

[19] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[20] C. C. Kemp, A. Edsinger, and E. Torres-Jara. Challenges for robot manipulation in human environments [grand challenges of robotics]. *IEEE Robotics Automation Magazine*, 14(1):20–29, March 2007.

[21] M. Kleinbort, K. Solovey, Z. Littlefield, K. E. Bekris, and D. Halperin. Probabilistic completeness of rrt for geometric and kinodynamic planning with forward propagation. *IEEE Robotics and Automation Letters*, 4(2):x–xvi, April 2019.

[22] Oliver Kroemer, Scott Niekum, and George Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms, 2019.

[23] James J. Kuffner and Steven M. La Valle. RRT-connect: an efficient approach to single-query path planning. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2000.

[24] Tobias Kunz and Mike Stilman. Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete. In *Springer Tracts in Advanced Robotics*, 2015.

[25] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[26] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[27] Sergey Levine and Vladlen Koltun. Guided policy search. In *30th International Conference on Machine Learning, ICML 2013*, 2013.

[28] Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. Sampling-based contact-rich motion control. In *ACM Transactions on Graphics (TOG)*, volume 29, page 128. ACM, 2010.

[29] Lucas Manuelli, Wei Gao, Peter R. Florence, and Russ Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *ArXiv*, abs/1903.06684, 2019.

[30] Tobia Marcucci, Robin Deits, Marco Gabiccini, Antonio Biechi, and Russ Tedrake. Approximate hybrid model predictive control for multi-contact push recovery in complex environments. In *Humanoid Robotics (Humanoids), 2017 IEEE-RAS 17th International Conference on*, pages 31–38. IEEE, 2017.

[31] Huseyin Onur Mete and Zelda B. Zabinsky. Pattern hit-and-run for sampling efficiently on polytopes. *Operations Research Letters*, 40(1):6 – 11, 2012.

[32] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand, 2019.

[33] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. Hybrid systems: From verification to falsification. In *International Conference on Computer Aided Verification*, pages 463–476. Springer, 2007.

[34] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *International Journal of Robotics Research*, 33(1):69–81, 2014.

[35] M. H. Raibert. Hopping in legged systems — modeling and simulation for the two-dimensional one-legged case. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14(3):451–463, May 1984.

[36] Sadra Sadraddini and Russ Tedrake. Linear encodings for polytope containment problems. *arXiv preprint arXiv:1903.05214*, 2019.

[37] Sadra Sadraddini and Russ Tedrake. Linear encodings for polytope containment problems. *arXiv preprint arXiv:1903.05214*, 2019.

[38] Sadra Sadraddini and Russ Tedrake. Sampling-based polytopic trees for approximate optimal control of piecewise affine systems. pages 7690–7696, 05 2019.

[39] Sadra Sadraddini and Russ Tedrake. Sampling-based polytopic trees for approximate optimal control of piecewise affine systems. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7690–7696. IEEE, 2019.

[40] Alexander Shkolnik, Michael Levashov, Ian R Manchester, and Russ Tedrake. Bounding on rough terrain with the littledog robot. *The International Journal of Robotics Research*, 30(2):192–215, 2011.

[41] Alexander Shkolnik, Matthew Walter, and Russ Tedrake. Reachability-guided sampling for planning under differential constraints. In *2009 IEEE International Conference on Robotics and Automation*, pages 2859–2865. IEEE, 2009.

[42] Jack Snoeyink. Handbook of discrete and computational geometry. chapter Point Location, pages 559–574. CRC Press, Inc., Boca Raton, FL, USA, 1997.

[43] Kiril Solovey, Lucas Janson, Edward Schmerling, Emilio Frazzoli, and Marco Pavone. Revisiting the asymptotic optimality of rrt*, 2019.

[44] David E Stewart. Rigid-body dynamics with friction and impact. *SIAM review*, 42(1):3–39, 2000.

[45] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[46] Russ Tedrake. *Applied optimal control for dynamically stable legged locomotion*. PhD thesis, Massachusetts Institute of Technology, 2004.

[47] Russ Tedrake, Ian R Manchester, Mark Tobenkin, and John W Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010.

[48] P. Tondel, T.A. Johansen, and A. Bemporad. Evaluation of piecewise affine control via binary search tree. *Automatica*, 39(5):945 – 950, 2003.

[49] Andrés Klee Valenzuela. Mixed-integer convex optimization for planning aggressive motions of legged robots over rough terrain. 2016.

[50] Dustin J Webb and Jur Van Den Berg. Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 IEEE International Conference on Robotics and Automation*, pages 5054–5061. IEEE, 2013.

[51] Albert Wu, Sadra Sadraddini, and Russ Tedrake. The nearest polytope problem: Algorithms and application to controlling hybrid systems. To appear in the proceedings of *2020 American Control Conference*, 2020.

[52] Albert Wu, Sadra Sadraddini, and Russ Tedrake. R3t: Rapidly-exploring random reachability set trees. To appear in the proceedings of *2020 IEEE International Conference on Robotics and Automation*, 2020.

[53] G.M. Ziegler. *Lectures on Polytopes*. Graduate texts in mathematics. Springer-Verlag, 1995.