EFFICIENT AND ASYMPTOTICALLY OPTIMAL KINODYNAMIC MOTION PLANNING

By

ZAKARY LITTLEFIELD

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Kostas E. Bekris

And approved by

_____

_____

_____

_____

New Brunswick, New Jersey

May 2020

ABSTRACT OF THE DISSERTATION

Efficient and Asymptotically Optimal Kinodynamic Motion Planning

by ZAKARY LITTLEFIELD

Dissertation Director:

Kostas E. Bekris

This dissertation explores properties of motion planners that build tree data structures in a robot's state space. Sampling-based tree planners are especially useful for planning for systems with significant dynamics, due to the inherent forward search that is performed. This is in contrast to roadmap planners that require a steering local planner in order to make a graph containing multiple possible paths. This dissertation explores a family of motion planners for systems with significant dynamics, where a steering local planner may be computationally expensive or may not exist. These planners focus on providing practical path quality guarantees without prohibitive computational costs. These planners can be considered successors of each other, in that each subsequent algorithm addresses some drawback of its predecessor. The first algorithm, `Sparse-RRT`, addresses a drawback of the `RRT` method by considering path quality during the tree construction process. `Sparse-RRT` is proven to be probabilistically complete under mild conditions for the first time here, albeit with a poor convergence rate. The second algorithm presented, `SST`, provides probabilistic completeness and asymptotic near-optimality properties that are provable, but at the cost of additional algorithmic overhead. `SST` is shown to improve the convergence rate compared to `Sparse-RRT`. The third algorithm, `DIRT`, incorporates learned lessons from these two algorithms and

their shortcomings, incorporates task space heuristics to further improve runtime performance, and simplifies the parameters to more user-friendly ones. `DIRT` is also shown to be probabilistically complete and asymptotically near-optimal. Application areas explored using this family of algorithms include evaluation of distance functions for planning in belief space, manipulation in cluttered environments, and locomotion planning for an icosahedral tensegrity-based rover prototype that requires a physics engine to simulate its motions.

Acknowledgements

My first acknowledgment is a huge thank you to all of the other graduate students and postdocs that I have had the pleasure of working with over the course of my graduate studies, and in some cases, even before then. Going through a graduate program can seem like a boot camp of sorts and everyone that goes through that process together are always able to comiserate with one another or celebrate each other's successes. This thank you is attributed to Ilias Apostolopoulos, Athanasios Krontiris, Yanbo Li, James Marble, Ryan Luna, Andrew Kimmel, Andrew Dobson, Rahul Shome, Justin Cardoza, Colin Rennie, Shaojun Zhu, Yunxiao Shan, Zacharias Psarakis, Hristiyan Kourtev, Nick Stiffler, Chaitanya Mitash, Shuai Han, Wei Tang, Aravind Sivaramakrishnan, Rui Wang, Kun Wang, David Surovik, and Avishai Sintov. An additional thank you goes out to Andrew Dobson and Andrew Kimmel. As roommates for most of my graduate studies, it is amazing that we were able to see each other so often at work and at home, and still end up being friends at the end of it. I want to thank you both for providing a welcome space to live and being some of my closest friends.

I have had the opportunity to work with many collaborators, and this work would not be possible without them. Thanks go out to Vytas SunSpiral, Jonathan Bruce, Ken Caluwaerts, Massimo Vespignani, Liam Pedersen, and Terry Fong at NASA Ames who were all highly accommodating during my many visits to the Mountain View area. These people also had a hand in guiding me to stay in robotics, and also provided additional support via the NASA Space Technology Research Fellowship which provided a large portion of my funding. Thank you to Dimitri Klimenko and Hanna Kurniawati for providing enlightening discussions on handling uncertainty in motion planning, and being a friendly face at conferences. Thank you to Weifu Wang for providing a unique perspective on how to approach problems. Thank you to Michal Kleinbort, Kiril Solovey, and Dan Halperin for working with me on interesting problems

in motion planning.

I feel that a special thank you needs to be extended to my advisor, Kostas Bekris. While he and I would have our disagreements, we both knew there were no personal grudges. Having spent a lot of time working with Kostas, I got to know him fairly well as a work colleague, but also as a mentor. Kostas has an infectious drive, optimism, and attitude toward research that keeps the people he works with moving forward. I will be forever grateful to him for giving me a chance to work with him and learn from him.

Thank you to my family, who have been wildly supportive, even when I moved across the country from them to continue studying. You have helped to get me to this point, and I constantly am working to reward your faith in me.

Thanks to all those at UNR and Rutgers who have provided indirect support to me during my post-secondary education, whether it be clerical, administrative, advisory, or whatever. Every bit of help you have provided is not forgotten and I am immensely appreciative to all of you. Finally, thank you to my committee members, Abdeslam Boularias, Jingjin Yu, and Timothy Barfoot, for your time and energy in assisting the preparation and review of this dissertation.

This dissertation is a compilation of several previous publications and original material. Existing citations compiled here include [45, 46, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 94]

Dedication

This work is dedicated to my late grandfather, LeRoy Fixen. While he will never read this dissertation, I know he would have loved to hear about it since he was always proud of my work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In words, the motion planning problem is to determine the inputs a robot needs to move from a start location to a goal location while avoiding collisions with its environment or itself. This high level task can vary depending on the robotic platform, but the motion planning algorithm employed will usually be determining some higher level inputs to that system that "drives" the system to its goal. In its most general context, this problem has been shown to be PSPACE-HARD [88].

One of the main contributors to this hardness result is the so called "curse of dimensionality." The idea is that as the dimensionality of the motion planning problem increases, the resulting algorithmic process that solves that problem gets exponentially more difficult to perform. Of note, a robot's state dimensionality is not limited to the three-dimensional space of the world, but is instead inherent to how the robot is built and configured. Another contributor to the difficulty of the motion planning problem is robot dynamics. The dynamics of the robot introduce additional constraints that must be respected in order to return a feasible motion plan. The consideration of both kinematic constraints such as obstacles and these dynamic constraints constitute the kinodynamic motion planning problem [20]. Examples of kinodynamic planning problems involve domains where the environment can cause drift (flying and underwater robots), or if the robot is subject to its own inertia when controlled. The complexity of these dynamics can even introduce runtime issues in a motion planning algorithm if the dynamical model is highly complex.

1.1 Sampling-based Planners

One class of motion planning algorithms has consistently shown the ability to address the dimensionality problem outlined above, namely the sampling-based planner. These methods aim to approximate the connectivity of a robot's configuration space with a graph structure. One such method is the Probabilistic Roadmap Method (`PRM`) [43]. This method builds an undirected graph in configuration space aimed at answering multiple start-goal queries. By providing multiple different candidate paths through the configuration space, it is possible to provide results for each of these requests, and do so by preprocessing the environment, requiring just a graph search to satisfy the query.

Another type of sampling-based planner instead builds a tree structure in the search space, instead of a multi-connected graph. These methods gain a memory benefit from not maintaining all possible edge connections in the graph, but are now relegated to only working for a smaller subset of queries, usually where the start state is given beforehand. In one sense, the roadmap method above is multiple-query, and tree methods, such as the Rapidly-exploring Random Tree (`RRT`) [49] and Expansive-Space Tree (`EST`) [34] are single query, and, in general, must be reconstructed for each new start-goal query, except in specialized cases.

In reality, using multi-connected graphs or trees both have benefits and drawbacks. Roadmaps have the benefit of being preprocessed, meaning a bulk of the runtime is offline, and therefore has less impact during query time, and the size of the graph constructed can be a bottleneck. This also assumes that the environment is mostly static, otherwise the approximation of the free configuration space is inaccurate at query time. For tree-based methods, there is minimal preprocessing, therefore most of its execution is online, but there is no graph search to perform. Tree-based methods are also amenable to more dynamic environments, since there is no preprocessing to invalidate.

One of the most important differences between these two method types is in one of the underlying primitive operations needed for edge generation. For the roadmap-based

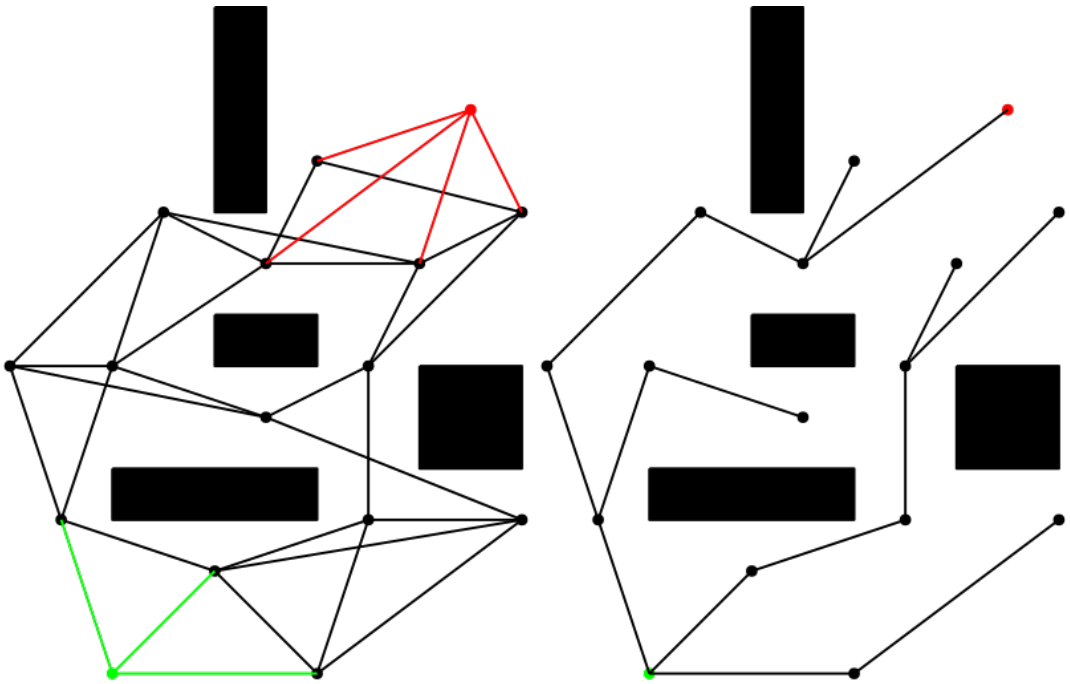

Figure 1.1: Construction of a roadmap and a tree. For roadmaps, an offline process generates a graph approximating the connectivity of the configuration space. Then, start and goal points are connected to the roadmap at query time. For tree planners, a search is started from the state state and extended until the goal is reached.

methods, these edges are constructed node to node, with the aid of a boundary value problem (`BVP`) solver. Given two configurations in the space, this primitive connects the two states with an optimal local plan without considering obstacles. This primitive is needed to maintain the multiple connections between nodes and their neighbors. For tree-based planners, while this `BVP` solver is sometimes utilized, it is not required. This critical observation becomes especially important in the context of kinodynamic planning. Kinodynamic planning is usually conducted in a state space, which usually consists of derivative terms from the configuration space (i.e. velocities, accelerations, jerks, etc.) As such, it becomes more difficult to perform these locally optimal connections. In some cases, it may even be impossible or impractical, which means we have to rely on forward propagation instead of a `BVP` solver, which only requires the initial condition and simulates a control input forward to reach a new state.

1.2 Properties of Sampling-based Planners

1.2.1 On the Completeness of Sampling-based Planners

Even though sampling-based methods can work effectively in high-dimensional configuration and state spaces, this comes at the cost of completeness. Instead, these algorithms have the property of probabilistic completeness:

Definition 1 *(Probabilistic Completeness) The probability of finding a solution to the motion planning query provided, if a solution exists, approaches one as the number of samples tends toward infinity [42, 49].*

Of note, this property has no guarantee on reporting that no solution exists, but can only guarantee that if a solution exists, the algorithm will find that solution. Most, if not all, sampling-based motion planners have this property, or some variation of it, such as resolution completeness [76].

1.2.2 On the Optimality of Sampling-based Planners

One major point of note is that while probabilistic completeness is a guarantee about returning a solution if it exists, this property does not provide a guarantee about how optimized that solution is. In fact, in the case of `RRT`, there is a proof that the common base version of `RRT` is provably sub-optimal [40]. For similar reasons to the completeness property being impossible to show, it is also not going to be possible to prove that a sampling-based algorithm is optimal. Therefore, a different property is considered, that of asymptotic optimality.

Asymptotic optimality is a similar property to probabilistic completeness, just with a different end result.

Definition 2 *(Asymptotic Optimality) The property that as the number of samples taken in the algorithm tends toward infinity, the probability that the cost of a solution returned by that algorithm is the optimal tends toward one [40].*

This property does not explicitly guarantee the solution quality is good at any finite iteration, but in practice, an anytime behavior is observed, where over time, solutions tend to improve as more samples are generated. This anytime behavior is usually observed in reasonable runtimes of sampling-based algorithms with this property, and it is usually not necessary to exhaust a large amount of time to generate a better solution. The asymptotic optimality property can also be relaxed into an asymptotic near-optimality property, where the guaranteed solution cost is not necessarily the optimal cost, but bounded by some function of the optimal cost.

A carefully constructed version of PRM, PRM* was analyzed and was determined to have this asymptotic optimality property [40]. In that same work, an asymptotically optimal algorithm called RRT* was also introduced that aimed to build a tree structure just like RRT. These two algorithms have become the basis for many different variants, just as PRM and RRT were previously (e.g. [1, 24, 35, 37, 55, 83, 108]) . Once again, there is a common assumption that the asymptotic optimality property is inherited in each variant.

1.3 Directions for Improving Practical Performance

In practice, there is no single implemented sampling-based algorithm that will solve all potential use cases. It is often the case that many different variant methods will be proposed that address different issues for different problems. For instance, there are may different PRM variants (e.g. [19, 26, 32, 39, 43, 70, 93, 109]). Likewise, there are many variants of RRT (e.g. [17, 21, 38, 50, 71, 112, 113]). It is commonly assumed that these modifications do not cause issues with probabilistic completeness, but that is not always the case. An example related to a particular common configuration of the RRT algorithm can be found in [51]. A particular problem domain example was provided that was not handled by this common RRT variant. It is not always the case that modifications remove properties however. Certain modifications to the base algorithmic framework can introduce more useful properties.

1.4 Sampling-based Planning: Kinodynamic Case

When applying sampling-based motion planners to the kinodynamic problem domain, there are additional considerations that need to be made compared to kinematic domains, which relate to the robot's dynamic constraints. For example, a canonical RRT variant constructs a new edge with the following steps:

- Randomly sample a state.
- Find the closest existing node in the search tree.
- Extend an edge toward the random sample, making a direct connection if within a distance threshold.

That third step becomes more problematic for dynamical systems. The task of steering toward a given state is the boundary value problem (BVP), which was discussed in the context of roadmap methods, and are also appearing in certain tree-based variants. In practice, a solver for the BVP needs to be provided, or a motion planning method that does not require such a solver needs to be selected.

Luckily, there is a simple solution to this problem, where instead of steering toward a state, an input to the system can be sampled at random, and then integrated forward in time, thus making forward progress. In this way, edges can be introduced into a tree search, but without the need for a BVP solver. However, this means that roadmap methods cannot make use of this forward propagation strategy easily.

1.4.1 Efforts towards Asymptotic Optimality

As discussed previously, the PRM* and RRT* methods are proven to be asymptotically optimal sampling-based motion planners [40]. Both methods require this BVP solver in order to provide that property, however. PRM* is explicitly a roadmap method, so it makes sense to have that requirement, but RRT* is building a tree structure. Moreover, RRT* is an algorithm that is a minimum spanning tree over an implicit roadmap, specifically that of the RRG algorithm [40]. Therefore, if planning is performed in a kinodynamic domain, and a variant of PRM* or RRT* is desired, a BVP solver is a needed component, either to maintain the multiple connections in the graph or to rewire the

tree. While this is not always a prohibitive requirement (e.g. [36, 55, 108]), for some challenging systems, it may be difficult or impractical to provide such a solver in all domains. On the other hand, a purely naïve strategy of randomly selecting nodes and randomly trying control inputs can be proven to be asymptotically optimal, but at an exponential computational cost [59]. This then leads to the central question of this dissertation:

> Is it possible to achieve asymptotic optimality (or asymptotic near-optimality) with a sampling-based planner that does not use a `BVP` solver at a reasonable computational cost?

1.4.2 Dissertation Contributions

This dissertation addresses this gap in sampling-based methods for kinodynamic planning. Tree sampling-based methods that do not require a `BVP` solver but can also provide asymptotic properties regarding path quality are presented. In summary, this dissertation presents:

- A preliminary algorithm, `Sparse-RRT`, which combines an existing `RRT` modification with a pruning strategy to be an effective motion planning method. `Sparse-RRT` is shown to be effective at finding and improving solution trajectories for kinodynamic problems. Unique to this dissertation, is a proof of probabilistic completeness for `Sparse-RRT`, and arguments about the convergence rate of the algorithm [65].
- With the motivation of improving the convergence rate, a second algorithm is presented, `SST`, which modifies the pruning strategy of `Sparse-RRT` by making the pruning regions stable, addressing an analytical problem with `Sparse-RRT`. This modification does not negatively affect runtime performance, and is still shown to be probabilistically complete. `SST` also is shown to improve the convergence rate for finding solutions relative to `Sparse-RRT`[58, 59].
- Following trends from the community, task-space heuristic information is introduced into this family of algorithms. In addition to this, the `DIRT` algorithm removes

the need for hand-tuned pruning and selection parameters needed by `Sparse-RRT` and `SST`, and replaces them with a unified selection and pruning mechanism, the dominance informed regions [60, 61].

- Given this family of algorithms, several application areas are also discussed, including planning under uncertainty, manipulation in clutter, and planning for a highly complex tensegrity-based robot [45, 64, 66, 67, 68].

This family of algorithms presented all derive inspiration from `RRT`, with useful fundamental changes in key components. Not only are these modifications able to provide path quality guarantees, explicit care is taken to improve the runtime efficiency of these algorithms, since the lack of a `BVP` solver can result in worse runtime performance. Chapter 2 introduces formal definitions and assumptions these sampling-based algorithms use. Chapter 3 discusses the `Sparse-RRT` algorithm, which has good performance in practice, but has a poor convergence rate, even when the algorithm is probabilistically complete. This convergence issue is then addressed with an updated algorithm, `SST`, in Chapter 4. Given this algorithm which has good computational properties, a case study on how the domain of planning under uncertainty can make use of `SST` is explored in Chapter 5. This application is aimed at evaluating different distance function choices for belief-space planning. Next, in order to further improve the performance for kinodynamic planning using a sampling-based method, and to address some shortcomings related to parameters in `SST`, the `DIRT` method is discussed in Chapter 6. Finally in Chapter 7, two challenging planning domains are discussed. The first is the problem of manipulation in clutter, which can be tackled with an application of `DIRT` that leverages the manipulator's different task spaces representations. Second, a challenging robotic platform based on the principle of tensegrity is discussed, along with an exploration of how the `DIRT` algorithm is able to effectively plan motions for that platform.

Chapter 2

Foundations on Sampling-based Kinodynamic Planning

Sampling-based motion planning methods work by creating samples that reside in the space where the robot can move. This requires a parameterization of that space, and there are different parameterizations depending on the use case. Typically, sampling-based planners aim to work in the configuration space, $\mathbb{C}$ [69]. This is the space where a full specification of the robot has been defined. For example, for a multi-link arm robot, the sequence of joint angles would provide the location of each component of that arm all the way down to its end-effector. The workspace, $\mathbb{W}$, is the world that the robot is affecting or working in. In the arm example, that would be the three dimensional world we observe and where the end-effector may try to manipulate objects. The workspace is where we usually define obstacles, since that is the space where they are exist, and then the robot itself has obstacles.

When moving to a planning domain of kinodynamic problems, it is common to extend from a configuration space to a state space, $\mathbb{X}$. This space is usually defined as the configuration space, $\mathbb{C}$, along with some of those terms' higher order derivatives, i.e. velocities, accelerations, jerks, etc. These extra terms are the extra dimensionality that adds additional complexity to the planning problem. In addition to the higher dimensionality, the dynamic constraints have to be considered as well.

Definition 3 *(Robot Dynamics) A robot's dynamics are assumed to satisfy a differential equation of the following form:*

$$\dot{x}(t) = f(x, u), \quad x \in \mathbb{X}, \quad u \in \mathbb{U} \tag{2.1}$$

where t is a time parameter and $\mathbb{U}$ is the space of control inputs to the robot.

In other words, there is a function that defines how the robot will evolve over time, given a state x and a control input u. The sequence of controls inputs to this function is denoted as a plan, and the resulting sequence of states generated by such a plan is called a trajectory.

Definition 4 *(Plan) A plan υ is a function $\upsilon(t) : [0, t_\upsilon] \rightarrow \mathbb{U}$, where t_υ is its duration. The space of all plans is defined as Υ. In this dissertation, Υ is composed of piecewise constant control sequences of length w.*

Definition 5 *(Trajectory) A trajectory π is a function $\pi(t) : [0, t_\pi] \rightarrow \mathbb{X}$, where t_π is its duration. A trajectory π is generated by starting at a given state $\pi(0) \in \mathbb{X}$, and by following the sequence of inputs provided by a plan, υ, and is subject to Equation 2.1. Note that $t_\pi = t_\upsilon$ if π is generated using υ.*

Recall that one of the objectives of a motion planning algorithm is to avoid obstacles. This necessitates defining a collision-free portion of the state space, $\mathbb{X}_f$, and an obstacle subset of the state space $\mathbb{X}_{obs}$. Mathematically, these two subspaces are complements of one another, $\mathbb{X}_f = \mathbb{X} \backslash \mathbb{X}_{obs}$. So the goal of a motion planner is to find a plan that results in a trajectory that completely resides in the collision-free part of the state space, while also getting from a desired start point to a desired goal, then optimizing the trajectory with respect to some cost function as a secondary objective. Of note, since the lack of `BVP`-solver implies the inability to get to a specific state easily, the goal needs to be defined as a region with some tolerance, which is commonly defined by a center point and a small radius.

Definition 6 *(Kinodynamic Motion Planning Problem) Given a robot that is constrained by dynamics of the form in Equation 2.1, a collision-free subset of that robot's state space, $\mathbb{X}_f$, an initial state, $x_\circ \in \mathbb{X}_f$, a goal region $\mathbb{X}_G \subset \mathbb{X}_f$, and the robot's control space, $\mathbb{U}$, find a plan υ that results in a trajectory π that satisfies $\pi(0) = x_\circ$, $\pi(t_\pi) \in \mathbb{X}_G$, and $\pi(t) \in \mathbb{X}_f$ for all $t \in [0, t_\pi]$.*

Note that by definition, the trajectory π is subject to the kinodynamic constraints of Equation 2.1.

2.1 Problem Definition and Notation

Recall during Section 1.2.1 that sampling-based planners are not able to provide a traditional completeness guarantee, but instead have to rely on an asymptotic property, probabilistic completeness:

Definition 7 *(Probabilistic Completeness) Let* Π_n^{ALG} *denote the set of trajectories discovered by an algorithm ALG at iteration* n*. Algorithm ALG is probabilistically complete, if for any motion planning problem* ($\mathbb{X}_f$, $x_\circ$, $\mathbb{X}_G$, $\mathbb{U}$) *the following holds:*

$$\liminf_{n\to\infty} \mathbb{P}(\ \exists\ \pi \in \Pi_n^{ALG} : \pi \text{ solution to } (\mathbb{X}_f, x_\circ, \mathbb{X}_G, \mathbb{U})) = 1.$$

Stated another way, as the number of planner iterations tends toward infinity, the probability that the motion planner's data structure contains a trajectory that solves the motion planning problem goes to one. This probabilistic completeness property does not make any guarantee about the quality of the trajectory that solves the motion planning problem. In order to compare trajectories to one another, a cost function needs to be defined.

Definition 8 *(Cost Function) A cost function, denoted by* **cost***, assigns a strictly positive numerical value to a trajectory, i.e.* **cost***:* $\Pi \to \mathbb{R}^+$ *and results in a cost value denoted by* g*. The notation* g *is commonly used in search-based planners such as* `A*`*.*

A cost function attempts to encode the desired behavior of the robot. Common cost function choices include distance traveled, time to execute the trajectory, or energy usage. Particular assumptions on this cost function are discussed in Assumption 3. The aim of a motion planner sensitive to the cost of its output trajectory is to minimize this cost. But once again, sampling-based planners do not provide a traditional optimality guarantee, but instead may provide an asymptotic optimality guarantee.

Definition 9 *(Asymptotic Optimality) Let* g^* *denote the optimal trajectory cost for a motion planning problem* ($\mathbb{X}_f$, $x_\circ$, $\mathbb{X}_G$, $\mathbb{U}$)*. Let* Π_n^{ALG} *denote the set of trajectories*

discovered by an algorithm ALG at iteration n. Let Y_n denote a random variable that represents the minimum cost value among trajectories in Π_n^{ALG}. An algorithm, ALG, is asymptotically optimal if for all independent runs, $\mathbb{P}(\left\{ \limsup_{n\to\infty} Y_n = g^* \right\}) = 1$

Another useful property that sampling-based planners can provide is asymptotic near-optimality. In this definition, instead of guaranteeing that an algorithm will find a true optimum almost surely, the cost of the best trajectory found by the algorithm is bounded.

Definition 10 *(Asymptotic Near-Optimality) Let g^* denote the optimal trajectory cost for a motion planning problem ($\mathbb{X}_f$, $x_\circ$, $\mathbb{X}_G$, $\mathbb{U}$). Let Π_n^{ALG} denote the set of trajectories discovered by an algorithm ALG at iteration n. Let Y_n denote a random variable that represents the minimum cost value among trajectories in Π_n^{ALG}. ALG is asymptotically near-optimal if for all independent runs:*

$$\mathbb{P}(\left\{ \limsup_{n\to\infty} Y_n \leq Bound(g^*) \right\}) = 1$$

where $Bound : \mathbb{R} \to \mathbb{R}$ is a function of the optimum cost, where $Bound(g^*) \geq g^*$.

Notice that the bound on cost returned by an algorithm is a function of the true optimum. In addition, there are usually other parameters of an algorithm that will influence this bounding function, and those are discussed as they arise.

2.2 Common Modules in Sampling-based Planners

Although sampling-based planners come in may different variations, there are a common set of tools that most of these algorithms employ. These common functions and modules have made experimentation and implementation simple. This section formalizes these modules to clarify what their expected inputs and outputs are, what variations are commonly considered, and what drawbacks are introduced by using the module.

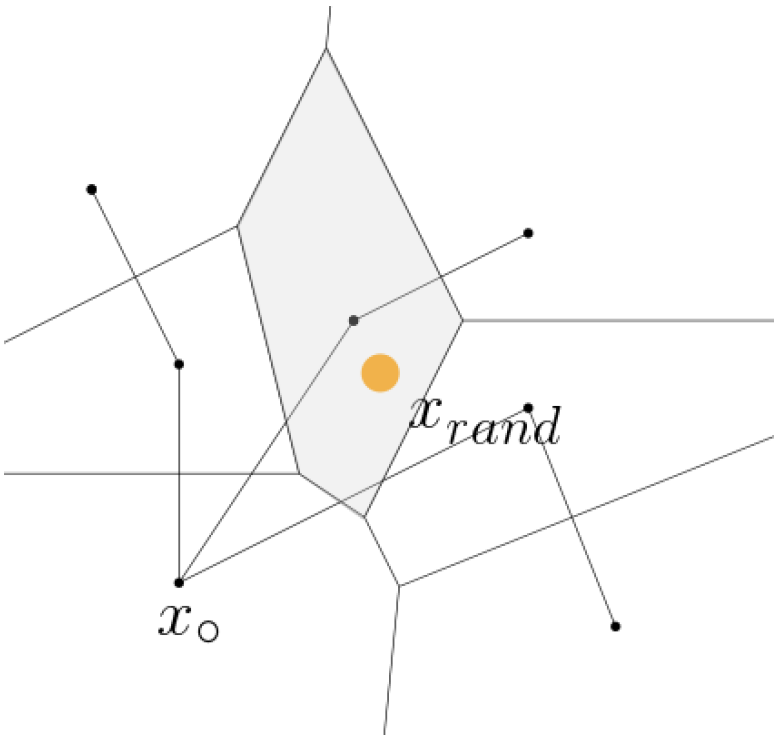

Figure 2.1: Voronoi bias example. The probability of selecting a node is proportional to the volume of the node's Voronoi region.

2.2.1 Sampling Strategy

One could argue that this is the most fundamental choice for a sampling-based planner; how are samples going to be drawn from the target space? The de facto answer is to sample uniformly at random. As an example, in the operation of `RRT`, the first step is to sample a state x from $\mathbb{X}$. This is then used to guide the rest of that algorithm iteration. By sampling from $\mathbb{X}$ uniformly at random, a natural exploration property arises, namely the Voronoi bias. Another common sampling strategy is to sample uniformly at random some percentage of the time, and then all other times provide a sample from the goal region $\mathbb{X}_G$. This introduces a bit of exploitation into the algorithm to try to get to the goal faster. Overall, there are a wide variety of strategies for sampling, but they all follow a similar signature:

Definition 11 *(Sampler) A sampler is a function* $\texttt{Sample} : \mathbb{S} \to s$ *where* $s \in \mathbb{S}$ *and* $\mathbb{S}$ *is some generic bounded space.*

A sampling strategy can also be used to only find samples in $\mathbb{X}_f$ or even to sample

controls from $\mathbb{U}$ to compose plans. One example of a specialization of this primitive samples in smaller subset of the state space biased based on previous solutions [24, 25].

2.2.2 Collision Checker

A collision checker is how a sampling-based planner generally can determine if a state is in $\mathbb{X}_f$ or $\mathbb{X}_{obs}$. The most common method for determining if a state is collision-free or not relies on computational geometry. Given a configuration of a robot, all the linkages and physical components of the robot can be modeled with polyhedra, usually composed of triangle faces. The same thing is done with obstacles in the environment. Then, there are fast algorithms for determining if one set of polyhedra collides with another set, and there are useful off-the-shelf libraries for this purpose [97, 16, 56, 78]. Then, we can define the collision checker as follows:

Definition 12 *(Collision Checker) A collision checker is a boolean function* $\texttt{Colliding} : \mathbb{C} \to \{0, 1\}$ *where if the function returns 0, that configuration is in* $\mathbb{X}_f$*, or if the function returns 1, then that configuration lies in* $\mathbb{X}_{obs}$*. This function can take a trajectory as well to return if any configuration on that trajectory is in collision or not.*

Note that only the configuration of the robot (and its geometry) are needed for a collision checker. None of the higher-order terms of the robot's state space are included in this primitive. A generalization of the configuration space collision checker to the state space can be seen as inevitable collision states [22]. In that framework, an inevitable collision state is a state where no matter what action the robot takes, a collision will occur in the future, even if the geometry is not in collision now. Ideally, when planning for a dynamical system, these are states we want to detect early and avoid, but these states are computationally intensive to compute. In contrast, configuration space collision checking is cheap, and be further accelerated with hardware solutions like GPUs and FPGAs [75, 77, 79] or with fast broadphase methods that can rule out possible collisions ahead of time.

2.2.3 Distance Function

One of the most heavily utilized components of sampling-based motion planners is the distance function.

Definition 13 *(Distance Function) A distance function,* **dist***, is a function that determines how far away two states are from one another.*

$$\textbf{dist} : (\mathbb{X}, \mathbb{X}) \to \mathbb{R}_{\geq 0}$$

A commonly used distance function is the weighted L2 norm, or weighted Euclidean distance. It is not always the case however, that all state spaces can be accurately measured with a Euclidean distance, especially those spaces which contain rotational dimensions. Although these spaces can be locally Euclidean, it is useful to remember what limitations a distance function may have on a particular problem domain.

In the context of motion planning, a distance function is taking the role of a "cost-to-go" estimator. The function **dist** is providing an estimate on what the cost of traveling between two states would be, since that is what the true distance between those states would be. As an example, for a car that needs to parallel park, the Euclidean distance between the start and end states of parallel parking is small (just the lateral shift of the vehicle), but the true distance would make use of a curved path that gets the vehicle into the parking space.

2.2.4 Nearest Neighbor Data Structure

A fundamental operation of sampling-based planners is to find what states are nearby a query state, given a distance function **dist**. In `RRT`, the first step after sampling a random state is to find what node in the existing search tree is closest to the random sample. In asymptotically optimal variants `RRT`* and `PRM`*, the set of k closest nodes to a state also needs to be answered (or alternatively, get all nodes within a radial region around the query point). A brute force implementation of these two methods would perform a linear search over all existing nodes in the data structure and compute the

distances between the query state and all of those nodes, an $\mathcal{O}(n)$ operation. Since this operation needs to be performed every iteration of the planner, these nearest neighbor queries end up dominating the runtime complexity of sampling-based algorithms.

In practice, there are off-the-shelf implementations of algorithms that can answer these nearest neighbor queries in much less time, usually $\mathcal{O}(\log(n))$ time where n is the number of nodes stored. These algorithms try to exploit whatever state space the robot has to perform hierarchical searches in that space efficiently [4, 7, 74]. However, these exploitations depend on the distance function in use, and if the distance function does not have the right structure (usually Euclidean), then the runtime improvements may not be observed. It can also be challenging to build these structures online while maintaining efficient lookup times.

A nearest neighbor structure for sampling-based motion planning needs to provide the following operations:

- $\texttt{Add}(q, \texttt{NN})$ - Inserts a state into the data structure (denoted here as $\texttt{NN}$)
- $\texttt{Nearest}(q) : \mathbb{X} \rightarrow \texttt{NN}$ - Given a query state q, return the closest state in the data structure.
- $\texttt{Near}(q, r) : \mathbb{X}, \mathbb{R}_{\geq 0} \rightarrow \texttt{NN}, \{x | x \in \texttt{NN}, \textbf{dist}(x, q) <= r\}$ - Return all states in $\texttt{NN}$ that are at most r distance from the query state.
- $\texttt{K} - \texttt{Near}(q, k) : \mathbb{X}, \mathbb{Z}_{>0} \rightarrow \texttt{NN}$ - Return the closest k states in $\texttt{NN}$ to the query state.

Algorithm 1: $\texttt{Graph-Add}(q, \texttt{NN})$

```
1 NN ← NN ∪ {q};
2 K_near ← K − Near(q, k ∝ log(|NN|));
3 foreach x ∈ K_near do
4 |   q.neighbors ← q.neighbors ∪ {x};
5 |   x.neighbors ← x.neighbors ∪ {q};
```

Some algorithms may need to make use of a remove operation ($\texttt{Remove}$), and this can invalidate certain methods for nearest neighbor structures. This is related to the hierarchical search strategy that assumes no removals occur. By removing nodes, additional modifications need to be added to the nearest neighbor structure, which adds runtime

costs. One way to accomplish the goal of fast removal with minimal data structure overhead takes inspiration from roadmap sampling-based methods like PRM*. Then, nearest neighbor queries become graph searches on this "roadmap" of nodes that maintain neighboring nodes when additions or removals happen.

Algorithm 2: `Graph-Nearest`(q)

$V_{rand} \leftarrow$ `Sample_Random_Vertices(NN)` `// Samples` $\sqrt{(}|$`NN`$|)$ `nodes at random.`
$v_{min} \leftarrow \arg\min_{v \in V_{rand}} \mathbf{dist}(v, q)$;
repeat
  $Nodes \leftarrow$ `Neighbors`$(v_{min}) \cup \{v_{min}\}$;
  $v_{min} \leftarrow \arg\min_{v \in Nodes} \mathbf{dist}(v, q)$;
until v_{min} *unchanged*;
return v_{min};

Algorithm 3: `Graph-K` $-$ `Near`(q, k)

$v_{min} \leftarrow$ `NN.Nearest`(v);
$K_{near} \leftarrow \{v_{min}\}$;
repeat
  $Nodes \leftarrow$ `Neighbors`(K_{near}) ; `// All neighbors of nodes in this set.`
  $K_{near} \leftarrow K_{near} \cup Nodes$;
  $K_{near} \leftarrow \arg\min_{(} k) v \in K_{near} \mathbf{dist}(v, q)$;
until K_{near} *unchanged*;
return K_{near};

Algorithm 4: `Graph-Remove`(q)

foreach $n \in q.neighbors$ **do**
  $n.neighbors \leftarrow n.neighbors \setminus \{q\}$;
foreach $n \in v.neighbors$ **do**
  $n.neighbors \leftarrow n.neighbors \setminus \{v\}$;
`NN` $\leftarrow$ `NN` $\setminus \{q\}$;

Algorithms that implement this graph-based nearest neighbor structure come from Section 4.4 in [59], and are detailed in Algorithms 1,2,3, and 4. The radial version of Algorithm 3 just changes the set update to be dependent on a radius instead of a parameter k.

2.2.5 Local Planner or Forward Propagation

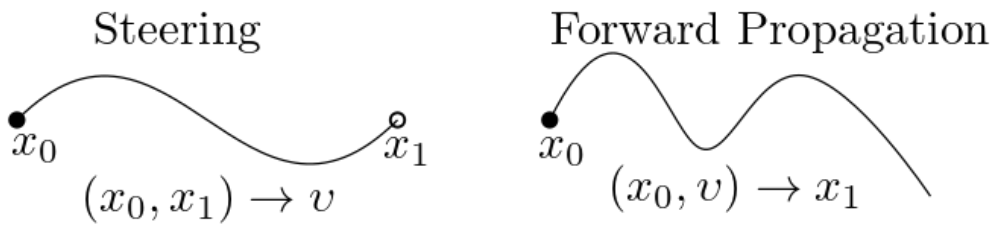

Figure 2.2: Local planner examples, steering (on the left) and forward propagation (right).

A local planner is how a sampling-based planner connects states together. In roadmap methods like `PRM`, a local planner provides the edges that connect states with trajectories that traverse between those two states, and in tree methods like `RRT`, the local planner can drive the system toward a goal point. The local planner's job is to generate the edges that grow the roadmap or tree, and actually accomplish the task of getting from start to goal.

Definition 14 *(Local Planner) A baseline local planner function can be defined as:*

$$local\_plan : \mathbb{X} \rightarrow \Upsilon, \Pi$$

where given a start state, generate a plan and a trajectory from that start state.

The reason that this definition is a baseline and not the canonical definition is because this function is a linchpin into how a planner operates, and is heavily coupled to the motion planning method. Examples of local planners are illustrated in Figure 2.2. As discussed above, a roadmap method needs to make use of a local planner that takes two states and provides a trajectory that connects those two states. A tree-based planner may make use of this "steering" local planner, but may also want to cutoff the trajectory after a certain distance. On the other hand, a tree-based planner may want to randomly sample a plan and use the system dynamics to generate a trajectory by forward simulating those control inputs, and foregoing any steering behavior. This is the so-called "forward propagation" primitive that is usually the fallback strategy

when the "steering" local planner is not available. This difference between "steering" and "forward propagation" local planners is one of the driving observations for this dissertation, and the effects of the difference between these two primitives is further discussed in the following chapters.

2.3 Baseline Kinodynamic Planner: RRT

The RRT algorithm [57] can be summarized into two main operations: a selection process and an expansion process. The selection is performed by randomly sampling a state in the state space, and then finding the closest node in the search tree to that random sample, using the nearest neighbor data structure. Then, an edge is generating that attempts to make progress toward the random sample. If that edge is collision-free, then that edge is added to the tree. This algorithm is outlined in Algorithm 5.

Algorithm 5: RRT-ForwardProp($\mathbb{X}$, $\mathbb{U}$, $x_\circ$, $\mathbb{X}_G$, T_{prop}, N)

```
1  G = {V ← {x_o}, E ← ∅};
2  NN.Add(x_o);
3  for N iterations do
4      x_selected ← Voronoi_Selection(V, X);
5      π_new ← MonteCarlo-Prop(x_selected, U, T_prop);
6      if not Colliding(π_new) then
7          V ← V ∪ {π_new(t)};
8          E ← E ∪ {π_new};
9          NN.Add(π_new(t));
10 return G(V, E);
```

This version of RRT in Algorithm 5 is commonly referred to as RRT-ForwardProp [57]. RRT-ForwardProp makes use of the random control sampling strategy of MonteCarlo-Prop to enable RRT to plan for kinodynamic systems. This is contrast to a version of RRT that tries to extend an edge toward the randomly sampled state from the selection process, which is commonly referred to as just RRT, or in some contexts RRT-Connect (conflicting with a bidirectional variant of RRT with the same name) [49].

The selection process of RRT has been labeled as Voronoi_Selection and the formal definition is in Algorithm 6. It is called Voronoi_Selection due to the implicit Voronoi

bias that this particular selection process exhibits. This bias creates an effect where nodes in RRT that are at the periphery of the explored space have higher likelihood to be selected for expansion at a given iteration. This creates an exploration effect in the search space.

Algorithm 6: Voronoi_Selection($\mathbb{V}, \mathbb{X}$)

```
1 x_sample ←Sample(𝕏);
2 return NN.Nearest(x_sample);
```

The expansion process for RRT-ForwardProp, called MonteCarlo-Prop, is formalized in Algorithm 7. Simply, a time duration is sampled, up to a max duration of T_{prop}, then a plan is sampled at random. Finally, a trajectory is generated by using the given system dynamics.

Algorithm 7: MonteCarlo-Prop(x, $\mathbb{U}$, T_{prop})

$t \leftarrow$ Sample($[0, T_{prop}]$);
$\upsilon \leftarrow$Sample(Υ);
return $\pi \leftarrow \int_0^t f(\pi(t), \upsilon(t))\, dt,\ where\ \pi(0) = x$;

2.3.1 Properties of RRT-ForwardProp

Formally, RRT-ForwardProp has the probabilistic completeness property [46]. The proof for probabilistic completeness was recently re-confirmed and clarified to determine the assumptions needed for probabilistic completeness. Arguably, the most effective practical property that RRT-ForwardProp has relates to the Voronoi bias previously discussed. This bias is especially effective at allowing the algorithm to explore the reachable space quickly.

A good search strategy should be balancing an exploration-exploitation trade-off. RRT-ForwardProp, with no other modifications, is a quintessential exploratory planner, with very little exploitation to find the goal more quickly. A method to introduce some exploitation is to perform goal-biasing in the state sampling step.

Another form of exploitation that a planner can exhibit relates to the quality of the resulting plans returned. RRT-ForwardProp has not been shown to have provable

path quality guarantees. In practice, `RRT-ForwardProp` can find better solutions over time given enough time executing, but this behavior is not exhibited in most practical scenarios.

These goals of exploration and exploitation (both finding solutions quickly and optimizing the solution found) are instrumental in a practical motion planning algorithm, and the canonical `RRT` method is not equipped to accomplish all of these goals. The `Sparse-RRT` algorithm aims to address the improving path quality goal that `RRT` was not able to provide on its own, while not sacrificing the exploration properties that make `RRT` successful.

Chapter 3

Sparse-RRT: Practical Kinodynamic Planning

After the discussion of RRT, what properties it has, benefits of the approach, and what can be improved, this section describes two modifications of RRT that make up the Sparse-RRT algorithm. Sparse-RRT is the baseline for the subsequent algorithms in Chapters 4 and 6. After presenting practical benefits of the Sparse-RRT approach, an analytical framework for proving probabilistic completeness is shown, and how it can be used to prove that Sparse-RRT is probabilistically complete. Finally, discussion on the weaknesses of this proof strategy is presented, as it relates to determining how quickly solutions can be found.

3.1 Algorithmic Description of Sparse-RRT

One of the reasons why RRT is unsuccessful at providing high quality paths arises because there is no step in the algorithm that considers path costs of trajectories in the tree, but instead only focuses on providing quick exploration of the free space. One example method to incorporate path costs is to select nodes biased by their path cost [104]. In this way, every expansion that is performed is biased toward the higher quality trajectories in the tree so far. Sparse-RRT uses this type of strategy (inspired by [104]) to improve path quality over time.

Algorithm 8 describes the BestNear selection procedure for Sparse-RRT. It begins the same as Voronoi_Selection with selecting a random sample from the state space. Then, a radial query to the nearest neighbor data structure is performed, with a given parameter δ_{BN}. This can return a set of nodes from the tree, or it can return nothing (which is common in the first iterations of the planner). If no nodes are within the δ_{BN} radius, then this reverts to the Voronoi_Selection algorithm. If there are nodes in the

Algorithm 8: `BestNear_Selection`$(\mathbb{V}, \mathbb{X}, \delta_{BN})$

$x_{sample} \leftarrow$`Sample`$(\mathbb{X})$;
$X_{near} \leftarrow$ `NN.Near`$(x_{sample}, \delta_{BN})$;
if $X_{near} = \emptyset$ **then**
    return `NN.Nearest`(x_{sample});
else
    return $\underset{x \in X_{near}}{\arg\min}$ **cost**(x);

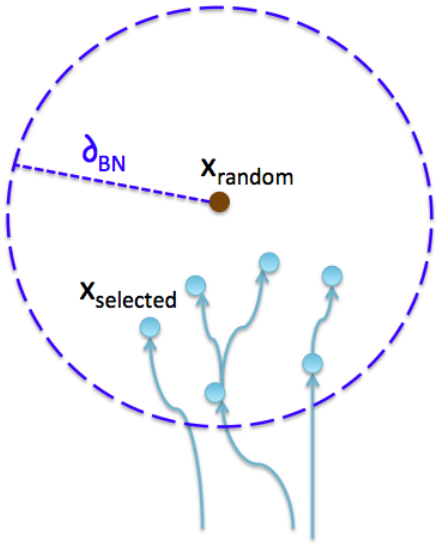

Figure 3.1: Illustration of the `BestNear` operation.

X_{near} set, then the node with the minimum trajectory cost from the root node (using the overloaded notation of **cost**(x) to represent this cost). To optimize this operation, the nearest neighbor query can be optimized to return the closest node if there are no nodes the δ_{BN} radius. In this way, there is no need to have two independent queries to the nearest neighbor structure.

3.1.1 Computational Efficiency of Using `BestNear`

In `RRT`, the dominant computational cost comes from the nearest neighbor queries in the selection step. All of the other operations in `RRT` have a relatively constant runtime complexity since they do not have to scale with the number of nodes in the tree. The nearest neighbor queries, therefore, influence the practicality of using a motion planning algorithm based on `RRT`. The `Nearest` function is generally the most efficient query that a nearest neighbor structure can perform (beyond the basic addition and removal operations). This is followed by the `K-Near` and `Near` operations. While these functions all generally have the same runtime complexity, $\mathcal{O}(\log(n)$, in practice the queries that return multiple results take more time to process. Therefore, the `BestNear` selection modification results in a slower algorithm, but the resulting trajectories are improved relative to the ones in `RRT`.

3.1.2 Achieving Computational Efficiency Via The Pruning Primitive

The second modification that `Sparse-RRT` employs is a pruning mechanism to remove unnecessary nodes from the search tree. By removing these nodes, they will never be selected for expansion again, and allow for more selections to be focused on the promising nodes in the tree for path quality.

`Sparse-RRT` makes use of Algorithm 9 to introduce a sparsity criterion. Whenever a new trajectory is introduced, the end state of that trajectory is compared with other trajectory end states within a radius, δ_s. If the new end state has a worse path cost than another node in the tree, the new state is removed from the tree. On the other hand, if the new node has better path cost, i.e. represents a better trajectory from the root node, the previously existing node is removed. However, this pruning operation should

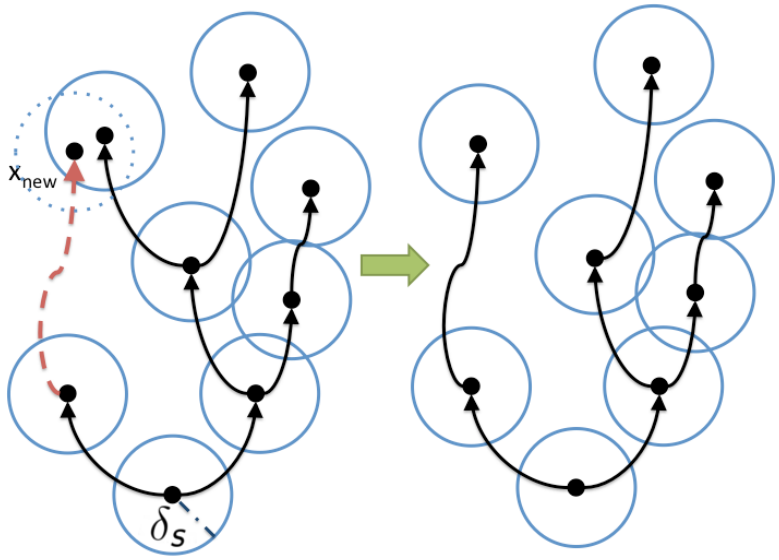

Figure 3.2: Illustration of the pruning operation of `Sparse-RRT`.

Algorithm 9: `Prune`(π_{new}, G, δ_s)

$X_{near} \leftarrow$ `NN.Near`$(\pi_{new}(t), \delta_s)$;
if **cost**$(\pi_{new}(t)) \geq \underset{x \in X_{near} \setminus \pi_{new}(t)}{\arg\min}$ **cost**(x) **then**
  $\mathbb{V}_{active} \leftarrow \mathbb{V}_{active} \setminus \{\pi_{new}(t)\}$;
  $\mathbb{E} \leftarrow \mathbb{E} \setminus \{\pi_{new}\}$;
  `NN.Remove`$(\pi_{new}(t))$;
else
  for $x_i \in X_{near} \setminus \pi_{new}(t)$ **do**
    $\mathbb{V}_{active} \leftarrow \mathbb{V}_{active} \setminus \{x_i\}$;
    `NN.Remove`(x_i);
    $\mathbb{V}_{inactive} \leftarrow \mathbb{V}_{inactive} \cup \{x_i\}$;
    $x_{del} \leftarrow x_i$;
    while `IsLeaf`(x_{del}) **and** $x_{del} \in \mathbb{V}_{inactive}$ **do**
      $x_{next} \leftarrow$`Parent`(x_{del});
      $\mathbb{V}_{inactive} \leftarrow \mathbb{V}_{inactive} \setminus \{x_{del}\}$;
      $\mathbb{E} \leftarrow \mathbb{E} \setminus \{\pi_{del}\}$;
      $x_{del} \leftarrow x_{next}$;

not remove a node's children, potentially removing an entire subtree. Those children nodes may be providing connectivity to a part of the state space that is difficult to reach. For this reason, when a node is pruned, it is removed from the nearest neighbor structure, which means that it will never be selected for expansion or prune other nodes. The nodes that are included in the nearest neighbor structure are represented by $\mathbb{V}_{active}$ and nodes that are only retained for path connectivity are in $\mathbb{V}_{inactive}$. As an optimization, if a node is in $\mathbb{V}_{inactive}$ and does not have children nodes, that node can be safely removed from the search tree, as well as an parent nodes that would have no children once its child is removed.

With the introduction of this pruning procedure, there are now two different nearest neighbor queries conducted in `Sparse-RRT`, one for selection and one for the pruning step. If the number of nodes that `Sparse-RRT` maintains was the same as `RRT`, there would be a significant runtime cost to performing these extra operations. But in reality, the number of nodes that `Sparse-RRT` maintains is significantly smaller than `RRT`, therefore, the runtime cost is actually smaller. In fact, the pruning operation creates the effect of maintaining a finite number of nodes in bounded state spaces, which means that there is an upper bound on the iteration cost of `Sparse-RRT`. For completeness, the algorithm of `Sparse-RRT` with the highlighted changes relative to `RRT` is presented in Algorithm 10.

Algorithm 10: `Sparse-RRT`($\mathbb{X}$, $\mathbb{U}$, $x_\circ$, $\mathbb{X}_G$, T_{prop}, $\underline{\delta_{BN}, \delta_s,}$ N)

$G = \{\underline{\mathbb{V}_{\textbf{active}} \leftarrow \{x_\circ\}}, \underline{\mathbb{V}_{\textbf{inactive}} \leftarrow \emptyset}, \mathbb{E} \leftarrow \emptyset\}$;
`NN.Add`($x_\circ$);
for N *iterations* **do**
  $x_{selected} \leftarrow \underline{\texttt{BestNear-Selection}(\mathbb{V}_{active}, \mathbb{X}, \delta_{BN})}$;
  $\pi_{new} \leftarrow$ `MonteCarlo-Prop`($x_{selected}$, $\mathbb{U}$, T_{prop});
  if *not* `Colliding`(π_{new}) **then**
    $\underline{\mathbb{V}_{\textbf{active}} \leftarrow \mathbb{V}_{\textbf{active}} \cup \{\pi_{new}(t)\}}$;
    $\mathbb{E} \leftarrow \mathbb{E} \cup \{\pi_{new}\}$;
    `NN.Add`($\pi_{new}(t)$);
    $\underline{\texttt{Prune}(\pi_{new}, \mathbf{G}, \delta_s)}$;
return $G(\mathbb{V}_{active}, \mathbb{V}_{inactive}, \mathbb{E})$;

In practice, there are some minor modifications that can be made to this algorithm.

First, the pruning condition for the newly generated trajectory can be checked before it is added to the search tree, or before collision checking. If the new trajectory is going to be pruned anyway, there is no reason to have to waste the computational effort for adding the node to the nearest neighbor structure, or collision check the trajectory. This type of trade-off can take into account what the expected computational bottleneck for a particular problem will be, and put the cheapest computation first. In this way, an iteration can "short circuit" early if the candidate edge would be removed.

3.2 Experimental Performance of `Sparse-RRT`

As previously discussed, `Sparse-RRT` can perform iterations much faster than `RRT` can when the number of iterations gets larger. This is despite the extra nearest neighbor queries that `Sparse-RRT` employs for the pruning operation. Data supporting this claim is provided in Table 3.1, where both the `BestNear` and `Prune` methods are introduced alone, combined together into the `Sparse-RRT` algorithm, and compared with a close variant [37]. The number of iterations when using the `Prune` procedure is at worst proportional to that of `RRT` in these cases. Note that the resulting path costs returned by methods using either `BestNear` and/or `Prune` are improved.

Another important consideration when evaluating motion planning techniques is to examine the behavior of an algorithm over time. This type of examination provides evidence that a motion planner can improve solutions over time, or can find solutions quickly. Figures 3.3 and 3.4 examine these components of these motion planners. The combination of the `BestNear` selection procedure along with the pruning operation (called `Drain` in the figure and in the reference material [65]) allows for solution cost improvement over time.

3.3 Analysis

Practical performance is important to consider when deciding what motion planning algorithm to use for an application. However, it is important to consider what provable properties an algorithm has as well, since those properties examine the core components

System	Algorithm	Iterations	Nodes	Path Cost (s)
Double Integrator	RRT	299749	299750	7.86
	RRT with Prune	511137	35979	6.67
	RRT with BestNear	269244	269245	6.87
	Sparse-RRT	508187	35495	6.6675
	RRT* w/ Shooting	66438	66439	7.76
Simple Pendulum	RRT	521528	521529	4.785
	RRT with Prune	761252	39215	2.5275
	RRT with BestNear	160856	160857	2.6825
	Sparse-RRT	668167	38699.5	2.385
	RRT* w/ Shooting	310289	310290	6.3825
Two-Link Acrobot	RRT	191081	154482	8.775
	RRT with Prune	301122	24451	5.22
	RRT with BestNear	184367	148824	5.8475
	Sparse-RRT	295923	23862	4.245
	RRT* w/ Shooting	313806	313807	6.67
Second-order Car	RRT	114200	83981	76.22
	RRT with Prune	129721	3672.5	32.995
	RRT with BestNear	119090	76062	38.175
	Sparse-RRT	135741.5	2967	35.655
	RRT* w/ Shooting	N/A	N/A	N/A

Table 3.1: Planner Performances for Variants of Sparse-RRT and A Benchmark After 10 Minutes (Provided from [65])

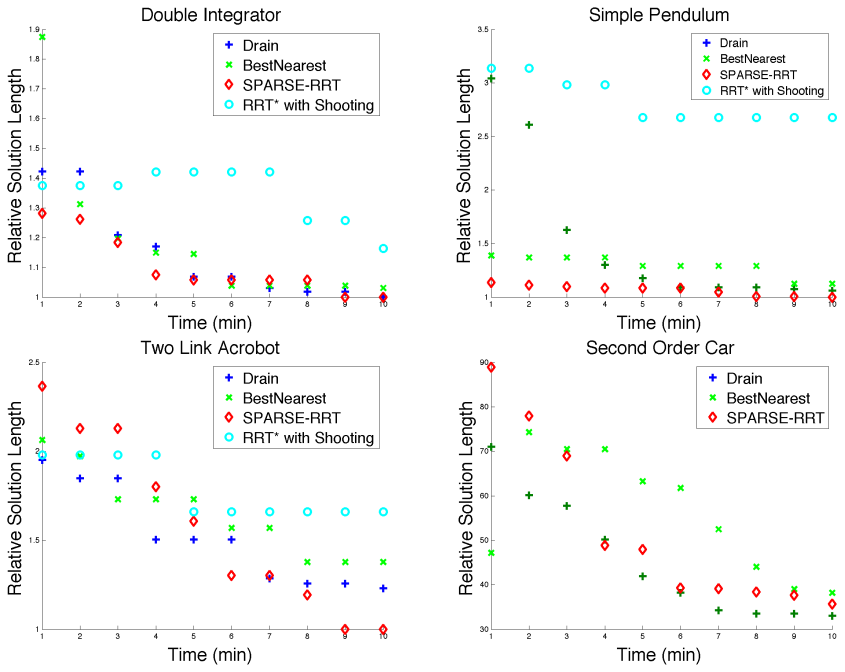

Figure 3.3: Comparison of Improvement Rates related to Sparse-RRT. The plots display the relative path cost over time. The path costs are relative to the best path found. Reported in [65].

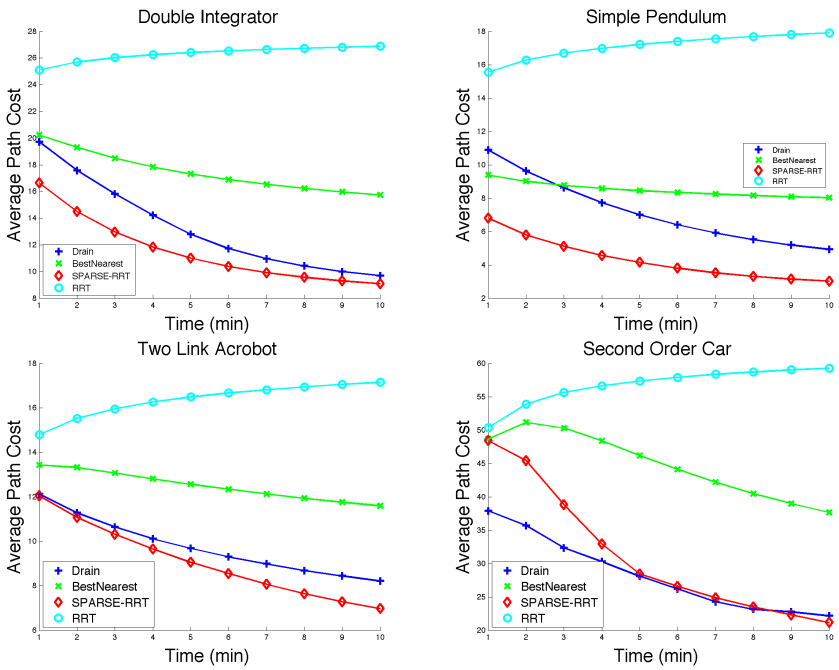

Figure 3.4: Comparison of average path cost of all nodes over time. Results from [65].

of the algorithm and provide confidence in its operations.

At a high level, the proof strategy for the family of planners discussed in this dissertation uses a Markov chain absorption argument (Theorem 11.3 [27]). The idea is to construct a Markov chain that represents the construction of a solution trajectory that a motion planner builds over time. Then, each state on the Markov chain represents progress toward building on this hypothetical solution trajectory. Each of these Markov states has a probability of transiting to the next state, which represents generating an edge that "observes" the solution trajectory. If the probability of transiting from each state to the next is strictly positive, it is guaranteed to "absorb" into the last state in the chain, otherwise known as the sink state, given infinite iterations. If the sink state is guaranteed to be reached, then it is guaranteed that the motion planner will generate a solution trajectory to the given problem. To summarize, the steps to this proof strategy are:

- Assume the existence of a solution trajectory with desired characteristics.
- Construct a covering ball sequence that discretizes this trajectory into trajectory segments.

- Create a Markov chain that mirrors this covering ball sequence.
- Prove that the transition probabilities in the Markov chain are strictly positive (i.e. provide a lower bound that is non-zero).

Now, each of these steps is discussed in detail with the needed lemmas and theorems to prove each step.

3.3.1 Assumptions on Solution Trajectory Existence

Beginning with the solution trajectory that will be observed, certain properties of that trajectory are needed. These definitions are adapted from the analysis of SST[58, 59]. First, there needs to be some clearance from obstacles all along this trajectory.

Definition 15 *(Obstacle Clearance) The obstacle clearance ϵ of a trajectory π is the minimum distance from obstacles over all states in π, i.e., $\epsilon = \inf_{t\in[0,t_\pi],x_o\in\mathbb{X}_{obs}}$ $\mathbf{dist}(\pi(t), x_o)$.*

Assumption 1 *The system dynamics from Equation 2.1 that generate a trajectory need to satisfy the following properties:*

- Chow's *condition [14] of Small-time Locally Accessible (*STLA*) systems [12]: For* STLA *systems, it is true that the reachable set of states $A(x, \leq T) \subset V$ from any state x in time less than or equal to T without exiting a neighborhood $V \subset \mathbb{X}$ of x, and for any such V, has the same dimensionality as $\mathbb{X}$.*
- *Bounded second derivative: $|\ddot{x}(t)| \leq M_2 \in R^+$.*
- Lipschitz continuous *for both of its arguments, i.e., $\exists\ K_u > 0$ and $\exists\ K_x > 0$:*

$$||f(x_0, u_0) - f(x_0, u_1)|| \leq K_u||u_0 - u_1||,$$

$$||f(x_0, u_0) - f(x_1, u_0)|| \leq K_x||x_0 - x_1||.$$

During the construction of the Markov chain that represents building up a solution trajectory, the notion of trajectories that are δ-similar is used:

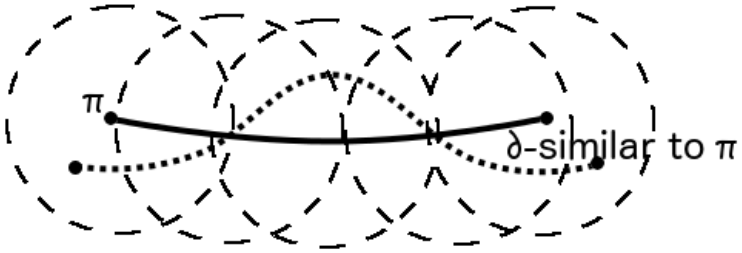

Figure 3.5: Illustration of a pair of δ-similar trajectories.

Definition 16 *(δ-Similar Trajectories) Trajectories π, π' are δ-similar if for a continuous, nondecreasing scaling function $\sigma : [0, t_\pi] \rightarrow [0, t_{\pi'}]$, it is true that $\pi'(\sigma(t)) \in \mathcal{B}_\delta(\pi(t))$. See Figure 3.5 for illustration.*

This idea of a δ-similar trajectories is needed to relate the hypothetical solution trajectory to the one that is actually constructed by the search tree. The main idea is to guarantee that there is a positive probability that an algorithm can generate such a δ-similar trajectory. First, this δ-similar trajectory needs to exist.

Lemma 1 *Let there be a trajectory π for a system satisfying Assumption 1. Then there exists a positive value δ_0 called the* **dynamic clearance***, such that: $\forall\ \delta \in (0, \delta_0]$, $\forall\ x'_0 \in \mathcal{B}_\delta(\pi(0))$, and $\forall\ x'_1 \in \mathcal{B}_\delta(\pi(t_\pi))$, there exists a trajectory π', so that: (i) $\pi'(0) = x'_0$ and $\pi'(t_{\pi'}) = x'_1$; (ii) π and π' are δ-similar trajectories.*

Proof: This property is very closely related to other assumptions of sampling-based planning proofs. Examples of this assumptions and concepts are "attraction sequences" [57], "homotopic in δ-interior of $\mathbb{X}_f$" [40, 41], and "linking sequence in ϵ-good free spaces" [33]. Informally speaking, *Chow's* condition implies that the *Ball Box* theorem holds. It also implies that the manifold $\mathbb{X}_f$ is *regular* and *involutory* [12]. A real-analytic control-affine system is small-time locally accessible ($STLA$), if and only if the *distribution* satisfies *Chow's* condition [102]. Assume every state on the optimal trajectory is a *regular point*. Then, the *sub-Riemannian ball* up to a small constant radius t_ϵ contains a weighted box of the same dimension of the state space and it is oriented according to vector fields of the *Lie brackets*. The bases are real analytical. Therefore

there exists an open neighborhood at each point x such that the bases evaluated at a different point x' converge to the bases at x as x' approaches x. Then, the weighted boxes centered by two sufficiently close states have a non-empty intersection. It implies that a hyper ball of some positive radius δ_0 can be fitted into this intersection region. Overall, there are two sufficiently close hyper-ball regions on the optimal trajectory such that between any point x in one ball and any point in the other ball there exists a horizontal curve and the length of the curve is less or equal to the radius t_ϵ of the *sub-Riemannian ball*. Then concatenating all hyper balls along a specified trajectory, results in the generation of δ-similar trajectories. ■

A kinodynamic motion planner in this context needs to consider both obstacle clearance and dynamic clearance. Trajectories that satisfy both clearance constraints are notated as being δ-robust.

Definition 17 *(δ-Robust Trajectories) A trajectory π for a dynamical system following Eq. 2.1 is called δ-robust if both its obstacle clearance ϵ and its dynamic clearance δ_0 are greater than δ.*

As an additional assumption on the kinodynamic motion planning problem, it is assumed that one such trajectory exists, and that the plans constructed by the motion planning algorithm can be used to generate such a trajectory.

Assumption 2 *For a motion planning problem, there exists a δ-robust trajectory π generated by a plan $\upsilon \in \Upsilon$.*

This assumption in practice is not very restrictive, since most real world environments have some clearance in them. Nevertheless, it is an important restriction to be made aware of.

3.3.2 Constructing the Markov Chain: The Covering Ball Sequence

Although path cost is not necessarily important in the context of probabilistic completeness, for the convenience of the forthcoming asymptotic near-optimality/optimality

proofs, assumptions on **cost** are introduced into the covering ball sequence construction method.

Assumption 3 *The cost function* $\mathbf{cost}(\pi)$ *of a trajectory is assumed to be* Lipschitz continuous. *Specifically,* $\exists\ K_c > 0$*:*

$$|\mathbf{cost}(\pi_0) - \mathbf{cost}(\pi_1)| \leq K_c \cdot sup_{\forall t}\{||\pi_0(t) - \pi_1(t)||\},$$

for all π_1, π_2 *with the same start state. Consider two trajectories* π_1, π_2 *such that their concatenation is* $\pi_1|\pi_2$ *(i.e., following trajectory* π_2 *after trajectory* π_1*),* **cost** *satisfies:*

- $\mathbf{cost}(\pi_1|\pi_2) = \mathbf{cost}(\pi_1) + \mathbf{cost}(\pi_2)$ *(additivity)*
- $\mathbf{cost}(\pi_1) \leq \mathbf{cost}(\pi_1|\pi_2)$ *(monotonicity)*
- $\forall\ t_2 > t_1 \geq 0$, $\exists M_c > 0$, $t_2 - t_1 \leq M_c \cdot |\mathbf{cost}(\pi(t_2)) - \mathbf{cost}(\pi(t_1))|$ *(non-degeneracy)*

With the assumptions laid out, the definition of the covering ball sequence is as follows:

Definition 18 *(Covering Ball Sequence) Given a trajectory* $\pi(t)$*:* $[0, t_\pi] \to \mathbb{X}_f$*, robust clearance* $\delta \in R^+$*, and a cost value* $g_\Delta > 0$*, the set of covering balls* $\mathbb{B}(\pi(t),\ \delta,\ g_\Delta)$ *is defined as a set of* $M + 1$ *hyper-balls:* $\{\mathcal{B}_\delta(x_0),\ \mathcal{B}_\delta(x_1),\ ...,\ \mathcal{B}_\delta(x_M)\}$ *of radius* δ*, where* x_i *are defined such that* $\mathbf{cost}(x_i \to x_{i+1}) = g_\Delta$ *for* $i = 0, 1, ..., M - 1$*. Illustration can be found in Figure 3.6.*

With this definition, and given an optimal solution trajectory, a theoretical covering ball sequence can be constructed. Note that this trajectory may not be found by the motion planning algorithm, and is instead a hypothetical trajectory that the motion planner might build during its operation, and is guaranteed to exist by Assumption 2.

Now that the covering ball sequence is constructed, the Markov chain that relates how a motion planner can build a trajectory that "observes" this covering ball sequence can be constructed as well. Let each Markov state $q_i \in \mathbb{Q}$ denote that a trajectory built by a motion planning algorithm has reached each covering ball of the sequence, i.e. q_i

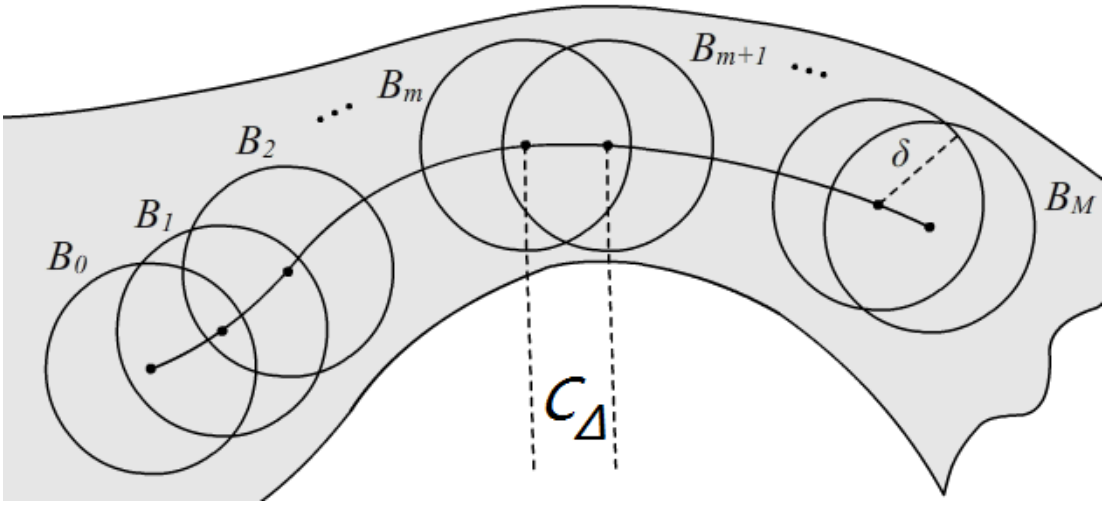

Figure 3.6: Illustration of a covering ball sequence around a reference trajectory, delimited by a small change in cost.

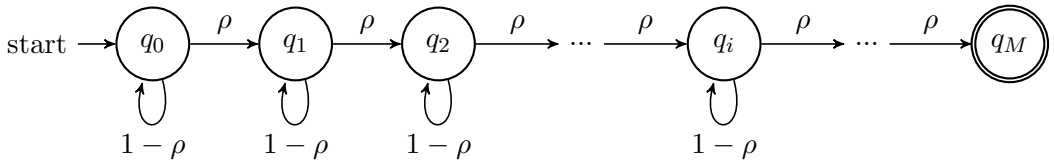

Figure 3.7: Markov chain illustrating the construction of a trajectory with `RRT-BestNear`.

is the state where a trajectory has reached $\mathcal{B}_\delta(x_i)$. Thus, the state q_M would represent the state where the motion planner has constructed a trajectory that solves the motion planning problem.

Next comes the transition probabilities on this Markov chain. Transitioning from state q_i to q_{i+1} represents the motion planner constructing a new trajectory from the robot's state $x'_i \in \mathcal{B}_\delta(x_i)$ to a new state $x'_{i+1} \in \mathcal{B}_\delta(x_{i+1})$. Let's denote the probability of this transition occurring as ρ. Then, the Markov chain can advance to the next state with probability ρ, and stays in the same state with probability $1-\rho$. If ρ is guaranteed to be strictly positive, i.e. $\rho > 0$, then the absorption theorem guarantees reaching the final Markov state (Theorem 11.3 [27]). An illustration of this Markov chain can be found in Figure 3.7.

3.3.3 Probability of Markov State Transition

For the sampling-based planners being discussed here, there are two major factors that influence the transition probabilities in the Markov chain:

- Will the node in the tree that lies in $\mathcal{B}_\delta(x_i)$ be selected this iteration?
- If that node is selected, will a trajectory to $\mathcal{B}_\delta(x_{i+1})$ be generated?

For the selection question, the algorithm `BestNear` needs to guarantee that the selection probability is positive. Let this selection probability from the `BestNear` algorithm be called γ_{BN}. For the expansion question, it is necessary to prove that the `MonteCarlo-Prop` procedure is guaranteed to generate a δ-similar trajectory to the reference trajectory with positive probability. Let γ_δ denote this probability. Then, the probability ρ is the product of these two events:

$$\rho = \gamma_{BN} * \gamma_\delta$$

.

Let's begin with the calculation of γ_{BN}.

Lemma 2 *Assuming uniform sampling in the* `Sample` *function of* `BestNear`*, and that* $\delta_{BN} < \delta$ *(the robust clearance constant), if* $\exists\ x$ *s.t.* $x \in \mathcal{B}_{\delta_{BN}}(x_i^*)$ *at iteration* n*, then the probability that* `BestNear` *selects for propagation a node* $x' \in \mathcal{B}_\delta(x_i^*)$ *can be lower bounded by a positive constant* γ_{BN} *for every* $n' > n$*.*

Proof: Consider the case that a random sample x_{rand} is placed at the intersection of a small ball of radius $\theta = \delta - \delta_{BN}$, and of a δ_{BN}-radius ball centered at a state $y_i \in \mathcal{B}_{\delta_{BN}}(x_i)$ that was generated during an iteration of the motion planner. In other words, if $x_{rand} \in \mathcal{B}_\theta(x_i) \cap \mathcal{B}_{\delta_{BN}}(y_i)$, then y_i will always be considered by `BestNear` because y_i will always be within δ_{BN} distance of a random sample there. The small ball is defined so that the δ_{BN} ball of x_{rand} can only reach states in $\mathcal{B}_\delta(x_i)$. It is also required that x_{rand} is in the δ_{BN}-radius ball centered at y_i, so that at least one node in $\mathcal{B}_\delta(x_i)$ is guaranteed to be returned. Thus, the probability the algorithm selects for propagation a node $x' \in \mathcal{B}_\delta(x_i^*)$ can be lower bounded by the following expression:

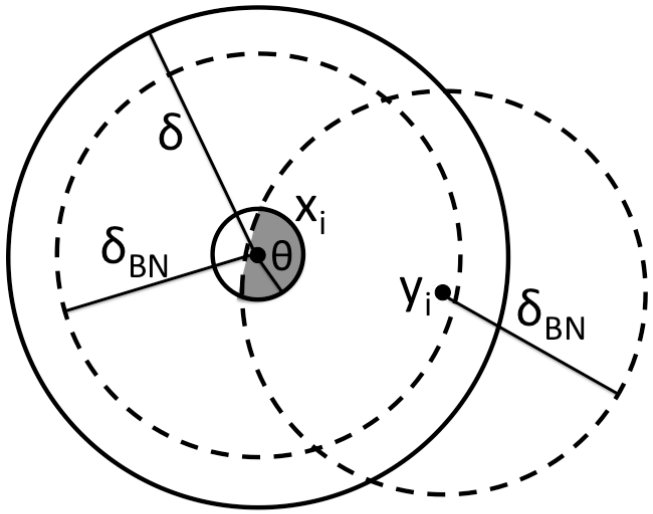

Figure 3.8: Possible area needed to guarantee selection of a node using `BestNear`.

$$\gamma_{BN} = \frac{\mu(\mathcal{B}_\theta(x_i) \cap \mathcal{B}_{\delta_{BN}}(x'))}{\mu(\mathbb{X})} > 0$$

where μ denotes the Lebesque measure of the region. ■

With a lower bound on the probability of selecting a node, next to consider is the lower bound on the `MonteCarlo-Prop` procedure at generating a δ-similar trajectory. First, consider the case where two trajectories share the same start state.

Theorem 1 *For two trajectories π, π' and any time horizon $T \geq 0$, so that $\pi(0) = \pi'(0) = x_0$ and $\Delta u = \sup_t(||\upsilon(t) - \upsilon'(t)||)$:*

$$||\pi'(T) - \pi(T)|| < K_u \cdot T \cdot e^{K_x \cdot T} \cdot \Delta u.$$

Proof: Given Assumption 1, for any two states x_0, x_1 and two controls u_0, u_1:

$$||f(x_0, u_0) - f(x_0, u_1)|| \leq K_u||u_0 - u_1|| \qquad ||f(x_0, u_1) - f(x_1, u_1)|| \leq K_x||x_0 - x_1||.$$

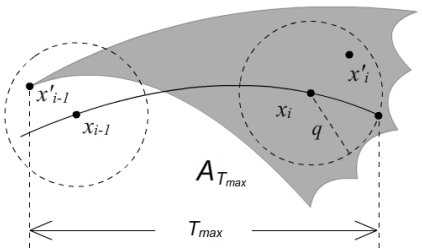

Figure 3.9: An illustration of the local reachability set for x'_{i-1}.

By summing these two inequalities:

$$||f(x_0,u_0)-f(x_0,u_1)||+||f(x_0,u_1)-f(x_1,u_1)|| \leq K_u||u_0-u_1||+K_x||x_0-x_1||. \quad (3.1)$$

Given the Euclidean distance, the following inequality is true:

$$||f(x_0u_0)-f(x_1,u_1)|| \leq ||f(x_0,u_0)-f(x_0,u_1)||+||f(x_0,u_1)-f(x_1,u_1)||.$$

By joining this with (3.1):

$$||f(x_0,u_0)-f(x_1,u_1)|| \leq K_u||u_0-u_1||+K_x||x_0-x_1||. \quad (3.2)$$

Now, divide $[0,T]$ into n segments with equal length Δt. Approximating the value of a trajectory $\pi(T)$ using Euler's Method, there is a sequence of states $\{x_0,x_1,...,x_n\}$. Let u_i denote $\upsilon(i\Delta t)$ corresponding to the control applied at each state.

$$x_i = f(x_{i-1},u_{i-1})\Delta t + x_{(i-1)}.$$

For two trajectories π and π' such that $\pi(0)=\pi'(0)=x_0$, $\upsilon(t)$ and $\upsilon'(t)$ are the corresponding plans. Then:

$$x_n = x_{n-1}+f(x_{n-1},u_{n-1})\Delta t x'_n = x'_{n-1}+f(x'_{n-1},u'_{n-1})\Delta t.$$

Then:

$$|x_n-x'_n|| \leq ||x_{n-1}-x'_{n-1}||+||f(x_{n-1},u_{n-1})-f(x'_{n-1},u'_{n-1})||\Delta t. \quad (3.3)$$

Using (3.2) and (3.3):

$$||x_n - x'_n|| \leq ||x_{n-1} - x'_{n-1}|| + (K_u||u_{n-1} - u'_{n-1}|| + K_x||x_{n-1} - x'_{n-1}||)\Delta t,$$

$$||x_n - x'_n|| \leq K_u\Delta t||u_{n-1} - u'_{n-1}|| + (1 + K_x\Delta t)||x_{n-1} - x'_{n-1}||. \tag{3.4}$$

By repeatedly reusing (3.4) to expand $||x_{n-1} - x'_{n-1}||$:

$$\begin{aligned} ||x_n - x'_n|| \leq (1 + K_x\Delta t)^n||x_0 - x'_0||+ \\ K_u\Delta t||u_{n-1} - u'_{n-1}||+ \\ (1 + K_x\Delta t)K_u\Delta t||u_{n-2} - u'_{n-2}||+ \\ \cdots + \\ (1 + K_x\Delta t)^{n-1}K_u\Delta t||u_0 - u'_0||. \end{aligned}$$

Since $x_0 = x'_0$, and $\Delta u = max_{i=0}^{n-1}(||u_i - u'_i||$:

$$||x_n - x'_n|| \leq K_u\Delta t\sum_{i=0}^{n-1}(1 + K_x\Delta t)^i\Delta u.$$

Since $n\Delta t = T$:

$$||x_n - x'_n|| \leq K_uT\frac{1}{n}\sum_{i=0}^{n-1}(1 + \frac{K_xT}{n})^i\Delta u$$

Due to the fact that $1 < (1 + \frac{\alpha}{n})^i < e^\alpha$, where $1 \leq i \leq n$ and $\alpha > 0$:

$$||x_n - x'_n|| < K_uTn\frac{1}{n}e^{K_xT}\Delta u \Rightarrow ||x_n - x'_n|| < K_uTe^{K_xT}\Delta u.$$

Given Assumption 1, Euler's method converges to the solution of the `Initial Value Problem`. Then:

$$||\pi(T) - \pi'(T)|| = \lim_{n\to\infty}||x_n - x'_n||, \text{ where } n\Delta t = T.$$

Therefore:$||\pi(T) - \pi'(T)|| < K_uTe^{K_xT}\Delta u$ ■

Theorem 1 provides a worst case upper bound on the error between two trajectories

that start at the same state. This type of construction is useful for the first edge that `MonteCarlo-Prop` will generate from the root node of the tree, but by itself is not general enough to handle the subsequent edges. It does, however, enable the task of lower bounding the probability that `MonteCarlo-Prop` can get to the next covering ball.

Theorem 2 *Given a trajectory π of duration t_π, the success probability for* `MonteCarlo-Prop` *to generate a δ-similar trajectory π' to π when called from an input state $\pi'(0) \in \mathcal{B}_\delta(\pi(0))$ and for a propagation duration $t_{\pi'} = T_{prop} > t_\pi$ is lower bounded by a positive value $\gamma_\delta > 0$.*

Proof: Consider that the start of trajectory π is $\pi(0) = x_{i-1}$ (from the covering ball sequence), while its end is $\pi(t_\pi) = x_i$. Similarly for π': $\pi'(0) = x'_{i-1}$ and $\pi'(t_{\pi'}) = x'_i$. From Lemma 1 regarding the existence of dynamic clearance we have the following: regardless of where x'_{i-1} is located inside $\mathcal{B}_\delta(x_{i-1})$, there must exist a δ-similar trajectory π' to π starting at x'_{i-1} and ending at x'_i. Therefore, if the reachable set of nodes $A_{T_{prop}}$ from x'_{i-1} is considered, it must be true that $\mathcal{B}_\delta(x_i) \subseteq A_{T_{prop}}$.

In other words, $A_{T_{prop}}$ has the same dimensionality d as the state space (Assumption 1). The goal is to determine a probability γ_δ that trajectory π' will have an endpoint in $\mathcal{B}_\delta(\pi(t_\pi))$.

Given a $\lambda \in (0, 1)$, construct a ball region $b = \mathcal{B}_{\lambda\delta}(x_b)$, such that the center state $x_b \in \pi(t)$ and $b \subset \mathcal{B}_\delta(x_i)$. Let Λ_δ denote the union of all such b regions. Clearly,

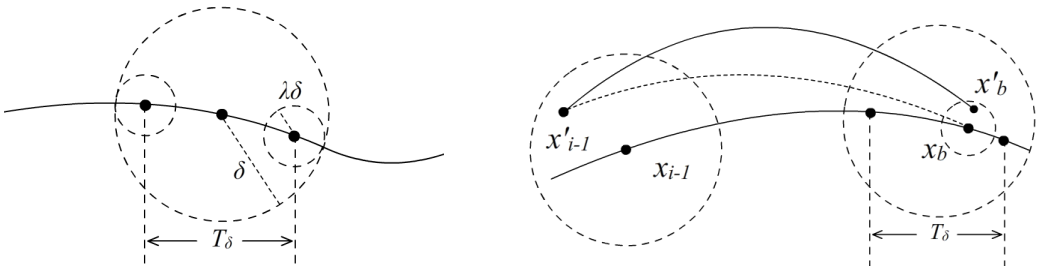

Figure 3.10: (left) A constructed segment of trajectory π of duration T_δ. (right) The dotted curve illustrates the hypothetical trajectory used as reference, and the solid curve above it illustrates one possible edge that is created by `MonteCarlo-Prop`.

all of x_b form a segment of trajectory $\pi(t)$. Let T_δ denote the time duration of this trajectory segment. For any state x_b, there must exist a δ-similar to π trajectory $\pi_b = \pi(x'_{i-1} \rightarrow x_b)$, due to Lemma 1.

Recall that `MonteCarlo-Prop` samples a duration for integration, and then, samples a plan in Υ. The probability to sample a duration t_{π_b} for π_b so that it reaches the region $\Lambda\delta$ is T_δ / T_{prop}.

Since the trajectory segment exists, it corresponds to a plan $\upsilon_m \in \Upsilon$. `MonteCarlo-Prop` only needs to sample a plan υ'_m, such that it is close to υ_m and results in a δ-similar trajectory. Then Theorem 1 guarantees that `MonteCarlo-Prop` can generate trajectory $\pi'_b = \pi(x'_{i-1} \rightarrow x'_b)$, which has bounded "spatial difference" from $\pi(x'_{i-1} \rightarrow x_b)$. And both of them have exactly the same duration of t_{π_b}. More formally, given the "spatial difference" $\lambda\delta$, if `MonteCarlo-Prop` samples a control vector υ'_m such that:

$$||\upsilon'_m - \upsilon_m|| \leq \frac{\lambda\delta}{K_u \cdot T_{prop} \cdot e^{K_x \cdot T_{prop}}} \qquad \Rightarrow \qquad ||x_b - x'_b|| < \lambda\delta.$$

Therefore, starting from state x'_{i-1}, with propagation parameter T_{prop}, `MonteCarlo-Prop` generates a δ-similar trajectory $\pi(x'_{i-1} \rightarrow x'_b)$ to $\pi(x_{i-1} \rightarrow x_i)$ with probability lower bounded by:

$$\gamma_\delta = \frac{T_\delta}{T_{prop}} \cdot \frac{\zeta \cdot (\frac{\lambda\delta}{K_u \cdot T_{prop} \cdot e^{K_x \cdot T_{prop}}})^w}{\mu(\Upsilon)} > 0.$$

where ζ is a unit hyperball in the parameter space for Υ.

■

Now that there are lower bounds on the probability for selection (γ_{BN}) and for propagating from one covering ball to the next (γ_δ), `RRT-BestNear`(`Sparse-RRT` without the `Prune` operation) has been shown to be probabilistically complete via the Markov chain absorption theorem. The next section discusses how this proof framework has complications when used for the full `Sparse-RRT` algorithm.

3.3.4 Probabilistic Completeness of `Sparse-RRT`: Complications

In practice, `Sparse-RRT` appears to exhibit good behavior when applied to different motion planning problems. In an effort to prove that `RRT-BestNear` is probabilistically complete, a Markov chain was constructed that represents the events of generating δ-similar trajectories to a hypothetical optimal one (although, this was assuming that only δ_{BN} was a parameter, see Figure 3.7). One characteristic of that Markov chain was that there were no backtracking edges on that chain. Every transition was to either stay in the same state, or to move forward to the next state. When the `Prune` procedure is introduced in `Sparse-RRT` this is no longer the case. Now, at every Markov state q_i, there is an additional possible transition to all $q_j, j < i$, and is illustrated in Figure 3.11. In addition, these probabilities are constantly changing, since these

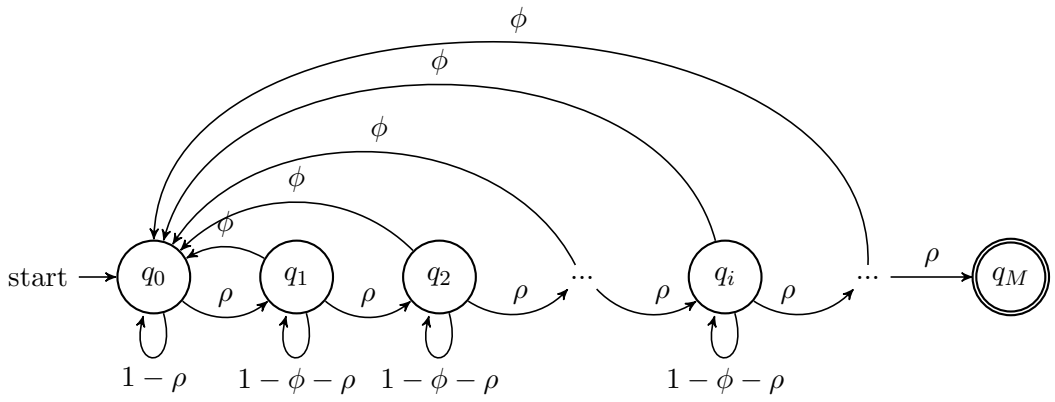

Figure 3.11: Markov chain illustrating the construction of a trajectory with `Sparse-RRT`.

probabilities of pruning depend heavily on the history of operations `Sparse-RRT` has already performed, but assume that these probabilities are upper bounded by ϕ. This assumption is reasonable since a node must be selected to generate an edge that would remove nodes on the covering ball sequence. This selection process relies on random sampling, so there is always some positive probability that this node is selected. This implies an upper bound of $\phi = 1 - \gamma_{BN}$ assuming that if the desired node is not selected, then the worst case node is selected and an edge that would remove progress is generated with probability one. Since these backward transitions are the worst case

scenario for finding a full solution trajectory, it is appropriate to consider the worst case probability for these transitions, where the transition results in arriving at the initial Markov state in the chain.

Even in this case, the Markov chain is still absorbing, and thus `Sparse-RRT` is still probabilistically complete. To clarify this result, below is a restatement of Theorem 11.3 used for proving that a homogeneous Markov chain will reach its sink state [27].

Theorem 3 *For an absorbing Markov chain, written in transition matrix form*

$$P = \left(\begin{array}{c|c} Q & R \\ \hline \mathbf{0} & \mathbf{I} \end{array}\right)$$

where the i,j element in this matrix holds the transition probability for going from state q_i to q_j. Q holds all of the transitions that are not absorbing states (transient), and R is the matrix of probabilities from transient states to sink states. The Markov chain is guaranteed to reach the sink state with probability one if and only if

$$Q^n \to 0 \; as \; n \to \infty$$

3.3.5 Convergence Rate for Finding Solutions with `RRT-BestNear` and `Sparse-RRT`

While both `RRT-BestNear` and `Sparse-RRT` are probabilistically complete, both algorithms are not equally able to find solutions. Consider another element from the analysis of absorbing Markov chains called the fundamental matrix. Using the fundamental matrix, it is possible to determine the expected number of iterations needed to reach the absorbing state. The following theorem is a restatement of Theorems 11.4 and 11.5 from [27].

Theorem 4 *For an absorbing Markov chain, written in transition matrix form*

$$P = \left(\begin{array}{c|c} Q & R \\ \hline \mathbf{0} & \mathbf{I} \end{array}\right)$$

The fundamental matrix $N = (\mathbf{I} - Q)^{-1}$ *which represents the expected number of times that the Markov chain is in each state, starting at each other state. Then, the number to iterations to reach the absorbing state from each other state can be calculated as*

$$\mathbf{E}(iterations) = N \cdot \mathbf{1}$$

where $\mathbf{1}$ *is column vector of ones.*

Using the fundamental matrix, and computing the expected number of iterations, it is possible to characterize the convergence rate of each algorithm to finding solutions. Consider the Markov chain representing `RRT-BestNear`(Figure 3.7). The transient matrix for this chain is

$$Q_{BN} = \begin{pmatrix} 1-\rho & \rho & 0 & \dots & 0 \\ 0 & 1-\rho & \rho & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1-\rho & \rho \\ 0 & \dots & \dots & 0 & 1-\rho \end{pmatrix}$$

and is an MxM matrix. The values $1 - \rho$ comprise the main diagonal of the matrix and then the forward transitions are the adjacent diagonal comprised of the ρ values. Given Q_{BN}, the fundamental matrix is

$$N_{BN} = (\mathbf{I} - Q_{BN})^{-1} = \begin{pmatrix} \frac{1}{\rho} & \frac{1}{\rho} & \frac{1}{\rho} & \dots & \frac{1}{\rho} \\ 0 & \frac{1}{\rho} & \frac{1}{\rho} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \frac{1}{\rho} \\ \vdots & \ddots & \ddots & \frac{1}{\rho} & \frac{1}{\rho} \\ 0 & \dots & \dots & 0 & \frac{1}{\rho} \end{pmatrix}$$

Then, to determine the expected number of iterations for `RRT-BestNear`, multiply a column vector of ones with the fundamental matrix N_{BN} and examining the first entry in the resulting matrix, since that first entry corresponds to the number of iterations

needed to get to the absorbing state from the initial state.

$$N_{BN} * \mathbf{1} = \begin{pmatrix} \frac{M}{\rho} \\ \vdots \end{pmatrix}$$

With this result, `RRT-BestNear` requires an expected number of iterations equal to $\frac{M}{\rho}$. This matches the result obtained if the same analysis was performed as Bernoulli trials with events of probability ρ. Now, let's contrast this result with the same result for `Sparse-RRT`.

Recall the Markov chain for `Sparse-RRT`(Figure 3.11). The transient matrix is defined by

$$Q_{\texttt{Sparse-RRT}} = \begin{pmatrix} 1-\rho & \rho & 0 & \dots & 0 \\ \phi & 1-\rho-\phi & \rho & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & 1-\rho-\phi & \rho \\ \phi & 0 & \dots & 0 & 1-\rho-\phi \end{pmatrix}$$

and the fundamental matrix (only focusing on the top row for clarity) is

$$N_{\texttt{Sparse-RRT}} = (\mathbf{I} - Q_{\texttt{Sparse-RRT}})^{-1} = \begin{pmatrix} \frac{(\rho+\phi)^{M-1}}{\rho^M} & \frac{(\rho+\phi)^{M-2}}{\rho^{M-1}} & \dots & \frac{(\rho+\phi)}{\rho^2} & \frac{1}{\rho} \\ & & \vdots & & \\ & & \vdots & & \end{pmatrix}.$$

Then, the expected number of iterations needed to reach the absorbing state from the initial state is

$$\sum_{i=1}^{M} \frac{(\rho+\phi)^{i-1}}{\rho^i} = \frac{(\frac{\rho+\phi}{\rho})^M - 1}{\phi}.$$

The expected number of iterations for the `Sparse-RRT` algorithm is exponential w.r.t. the number of Markov states, M. In contrast, the expected number of iterations for `RRT-BestNear` is linear w.r.t M. Granted, this is a worst case analysis, so the empirical performance is likely to not follow these expectations, but under the right conditions, `Sparse-RRT` may have difficulty finding a solution.

Chapter 4

SST: Sampling-based Kinodynamic Planner with Guarantees

While `Sparse-RRT` has good practical performance, proving probabilistic completeness is difficult due to the pruning operation. The main problem arises from the shifting nodes resulting from repeated node removals. These removals in the tree can remove the stability of trajectories in certain regions of the state space, particularly the theoretical optimum that is used in analysis. In order to maintain as much of the computational benefit as possible from `Sparse-RRT`, but make the proof of probabilistic completeness possible with the desired convergence rate, the pruning operation needs to be modified.

4.1 Algorithmic Description of `Stable Sparse-RRT`

The primary drawback when proving the convergence rate of `Sparse-RRT` is that the regions that can prune other nodes shift along with the nodes in the search tree, causing subtle shifts in the locations of nodes, thereby creating instability in the tree. It is then theoretically possible that after the right sequence of events, there will not be nodes in a local area where there once were. This instability can also make the algorithm a bit volatile in some adversarial cases. This undesired behavior may not be highly likely, but remedying this potential outcome is necessary for a more effective algorithmic strategy.

Instead of having the regions that are "claimed" by a node be represented by the node itself, and subject to moving along with the node, when a node reaches an unexplored part of the state space, a "witness" node can be placed where that tree node is. This witness node does not change its location over time, and then watches for any new tree nodes that enter its region, and maintains the invariant that only one node can be the "representative" in that witness's region. Another interpretation of this is that a

flagpole is placed by the first node that reaches a region of the state space. Then, only one tree node can be represented by that flag at any given time.

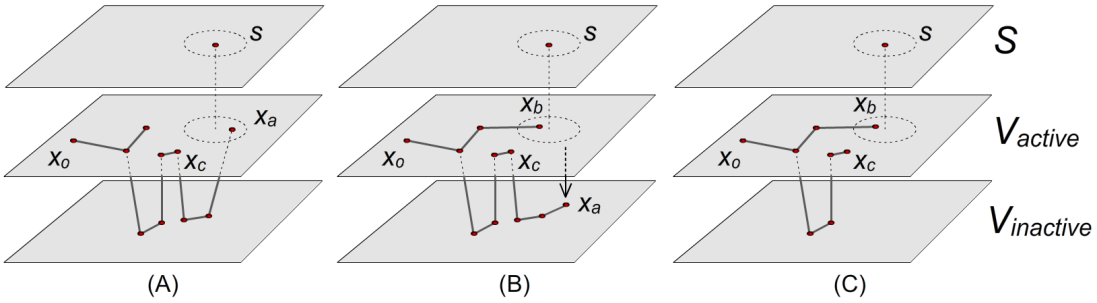

Figure 4.1: Example illustrating the witness set, S, in `SST`, the active set of nodes that have not been pruned, $\mathbb{V}_{active}$, and nodes that are removed from the nearest neighbor structure, but needed for connectivity, $\mathbb{V}_{inactive}$.

Algorithm 11: `Stable Sparse-RRT`($\mathbb{X}$, $\mathbb{U}$, $x_\circ$, $\mathbb{X}_G$, T_{prop}, δ_{BN}, δ_s, N)

$G = \{\mathbb{V}_{active} \leftarrow \{x_\circ\}, \mathbb{V}_{inactive} \leftarrow \emptyset, \mathbb{E} \leftarrow \emptyset\}$;
$s_0 \leftarrow x_\circ$, $s_0.rep = x_\circ$, $S \leftarrow \{s_0\}$;
`NN.Add`($x_\circ$);
`NN`$_S$`.Add`(s_0);
for N *iterations* **do**
  $x_{selected} \leftarrow$ `BestNear-Selection`($\mathbb{V}_{active}, \mathbb{X}, \delta_{BN}$);
  $\pi_{new} \leftarrow$ `MonteCarlo-Prop`($x_{selected}$, $\mathbb{U}$, T_{prop});
  $s_{new} \leftarrow$`FindWitness`($\pi_{new}(t), G, S, \delta_s$);
  if *not* `Colliding`(π_{new}) **and** `BetterRepresentative`($\pi_{new}(t), s_{new}$) **then**
    $\mathbb{V}_{active} \leftarrow \mathbb{V}_{active} \cup \{\pi_{new}(t)\}$;
    $\mathbb{E} \leftarrow \mathbb{E} \cup \{\pi_{new}\}$;
    `NN.Add`($\pi_{new}(t)$);
    `SST-Prune`($s_{new}.rep$,G);
    $s_{new}.rep \leftarrow \pi_{new}(t)$
return $G(\mathbb{V}_{active}, \mathbb{V}_{inactive}, \mathbb{E})$;

Algorithm 11 provides this stability modification to the `Sparse-RRT` algorithm, thus making `Stable Sparse-RRT(SST)`. `SST` makes use of a second set of nodes, S, that represent the witnesses. An illustration of this witness set relative to tree nodes is in Figure 4.1. This witness set is used in the pruning step of the algorithm, just as in `Sparse-RRT`, but each witness is decoupled from the tree. Each iteration selects a node for expansion with the `BestNear-Selection` algorithm (Algorithm 8), expands from that state using `MonteCarlo-Prop`(Algorithm 7), collision checks the edge, and

Algorithm 12: `FindWitness`$(x_{new}, G, S, \delta_s)$

$s_{new} \leftarrow \texttt{NN}_S.\texttt{Nearest}(\pi_{new}(t))$;
if $\textbf{dist}(x_{new}, s_{new}) > \delta_s$ **then**
    $S \leftarrow S \cup \{x_{new}\}$;
    $s_{new} \leftarrow x_{new}$;
    $s_{new}.rep \leftarrow NULL$;
    $\texttt{NN}_S.\texttt{Add}(s_{new})$;
return s_{new}

Algorithm 13: `BetterRepresentative`(x_{new}, s_{new})

$x_{peer} \leftarrow s_{new}.rep$;
if $x_{peer} == NULL$ **or cost**(x_{new}) $<$ **cost**(x_{peer}) **then**
    return True;
return False;

Algorithm 14: `SST-Prune`(x, G)

if x *is not* $NULL$ **then**
    $\mathbb{V}_{active} \leftarrow \mathbb{V}_{active} \setminus \{x\}$;
    $\texttt{NN}.\texttt{Remove}(x)$;
    $\mathbb{V}_{inactive} \leftarrow \mathbb{V}_{inactive} \cup \{x\}$;
    $x_{del} \leftarrow x$;
    while $\texttt{IsLeaf}(x_{del})$ **and** $x_{del} \in \mathbb{V}_{inactive}$ **do**
        $x_{next} \leftarrow \texttt{Parent}(x_{del})$;
        $\mathbb{V}_{inactive} \leftarrow \mathbb{V}_{inactive} \setminus \{x_{del}\}$;
        $\mathbb{E} \leftarrow \mathbb{E} \setminus \{\pi_{del}\}$;
        $x_{del} \leftarrow x_{next}$;

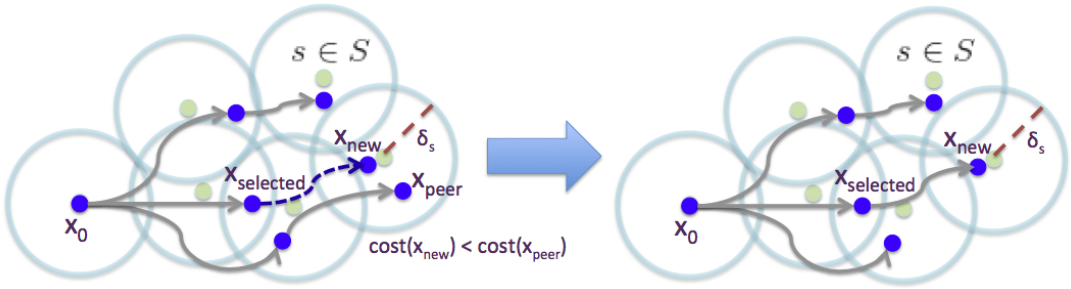

Figure 4.2: When adding a new edge that ends in an existing witness region, the worse cost node is moved into the $\mathbb{V}_{inactive}$ set. Note that the witness region does not shift when the new node is added, which is where the `Stable` attribute is introduced.

finally prunes unneeded nodes (now using Algorithms 12, 13, and 14). There is a slight modification where a node is never added if that new tree node does not end up replacing an existing representative node, or does not reach new areas of the state space. In the cases where a new region of the state space is explored, a new witness node is placed at the same state the tree node is, and that witness will exist for the lifetime of the algorithm (see Figure 4.3 for an example).

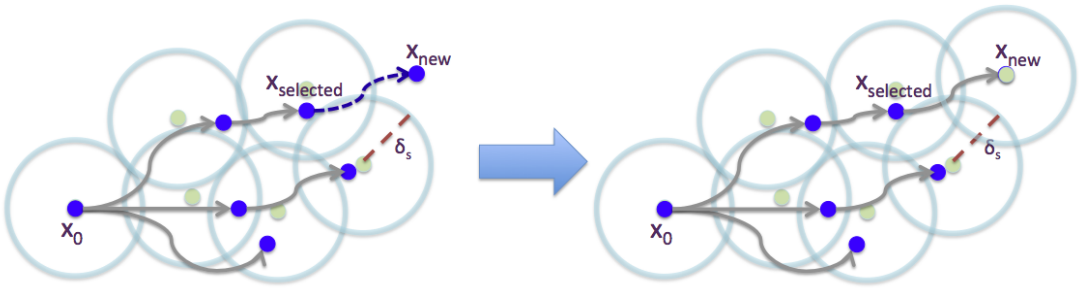

Figure 4.3: When a new edge does not end in any existing witness region, a new one is created.

4.2 Experimental Performance of `SST`

Note that `SST` now makes use of two nearest neighbor data structures, one for the tree nodes and one for the witness nodes. Now that the witness nodes used for pruning are decoupled from the tree nodes, this is needed to keep the selection operation and pruning operation correct. While this does impose an additional computational cost for storage, since this aims to reduce the number of nodes that need to be stored, it is

			δ_s				
			0.2	0.4	0.6	0.8	1.0
δ_{BN}	1.0	Initial Solution Time	0.1105	0.1190	0.0603	0.0451	0.0548
		Initial Solution Cost (s)	3.4201	3.2445	3.2155	3.0468	2.7695
		Final Solution Cost (s)	1.7782	1.7851	1.7916	1.8229	1.8627
	1.2	Initial Solution Time	0.1296	0.0915	0.0999	0.0593	0.0635
		Initial Solution Cost (s)	3.2891	3.2614	3.2105	2.9554	2.7371
		Final Solution Cost (s)	1.7798	1.7797	1.7973	1.8273	1.8723
	1.4	Initial Solution Time	0.1516	0.0938	0.0670	0.0498	0.0567
		Initial Solution Cost (s)	3.2248	3.1364	2.9795	2.8908	2.7365
		Final Solution Cost (s)	1.7866	1.7833	1.7988	1.8193	1.8621
	1.6	Initial Solution Time	0.1687	0.0961	0.0671	0.0545	0.0595
		Initial Solution Cost (s)	3.0865	3.0506	2.9523	2.8334	2.7185
		Final Solution Cost (s)	1.7890	1.7829	1.7987	1.8232	1.8853
	1.8	Initial Solution Time	0.2095	0.0906	0.0679	0.0724	0.0601
		Initial Solution Cost (s)	3.0641	3.1027	2.8082	2.6549	2.7493
		Final Solution Cost (s)	1.7949	1.7852	1.7971	1.8416	1.8846

Table 4.1: A comparison of different parameter choices in `SST` for a 2D point with 60 seconds of execution time. (From [59])

still a better investment in space than keeping all nodes like in `RRT` or `RRT-BestNear`. The time complexity is still the same as `Sparse-RRT` however. Practically, `SST` still provides good runtime performance in both finding good solutions and in keeping the computational cost low.

Table 4.1 shows practical performance for different values of the parameters for selection radius (δ_{BN}) and pruning radius (δ_s). Examining some of the trends that this data exhibits, the pruning radius δ_s has a relationship with how quickly a solution is found and how good that initial solution is. As the pruning radius gets larger, solutions are found faster and of better path quality. The amount of improvement in the solution trajectories is reduced when the pruning radius is high however. For larger pruning radii, `SST` is greedily removing edges that while they are locally suboptimal, may provide a better solution path to the goal.

Looking at the selection radius (δ_{BN}), there are also interesting effects to observe. In general, as the selection radius increases, the time to finding the first solution also increases. This makes sense intuitively since larger selection radii will result in processing more nodes each iteration. This extra cost allows `SST` to find higher quality

solutions when the selection radius is larger. By selecting values of δ_{BN} and δ_s, a user can tailor the implementation to the particular problem domain and the computational budget for that task.

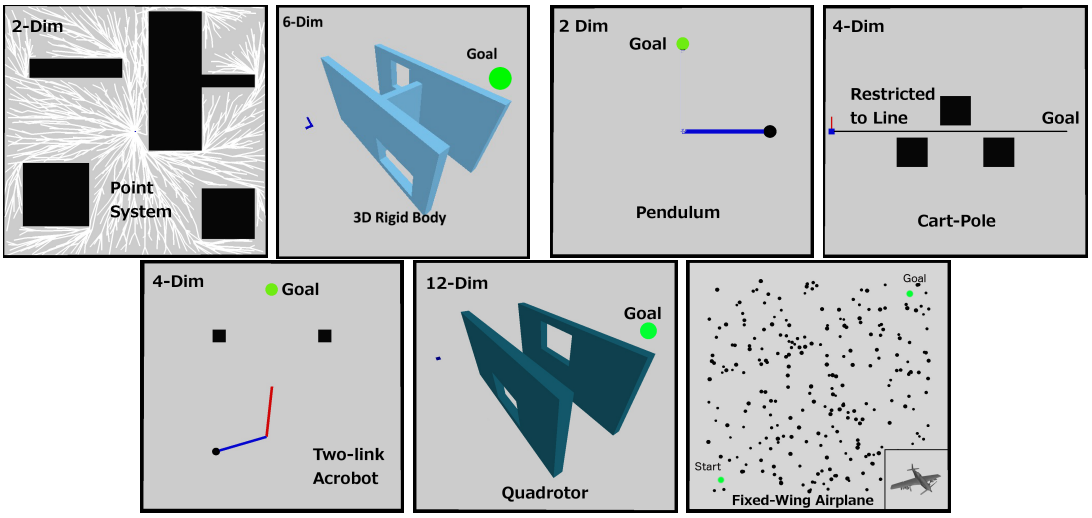

Figure 4.4: The different benchmarks used to evaluate SST. Each experiment is averaged over 50 runs of each algorithm, and results are reported from [59]

System	Parameters	Distance Function	δ_s	δ_{BN}
Kinematic Point	2 Dim. State, 2 Dim. Control	Euclidean Distance	.5	1
3D Rigid Body	6 Dim. State, 6 Dim. Control	Euclidean Distance	2	4
Simple Pendulum	2 Dim. State, 1 Dim. Control, No Damping	Euclidean Distance	.2	.3
Two-Link Acrobot [98]	4 Dim. State, 1 Dim. Control,	Euclidean Distance	.5	1
Cart-Pole [80]	4 Dim. State, 1 Dim. Control,	Euclidean Distance	1	2
Quadrotor [2]	12 Dim. State, 4 Dim. Control,	Distance in $\mathbb{S}E3$	3	5
Fixed-Wing Airplane [81]	9 Dim. State, 3 Dim. Control,	Euclidean Distance in $\mathbb{R}^3$	2	6

Table 4.2: The experimental setup used to evaluate SST. Parameters are available in the corresponding references.

Getting into comparisons against baseline planners, several different robot models are considered. The following experimental evaluations were originally presented in a previous publication [59].

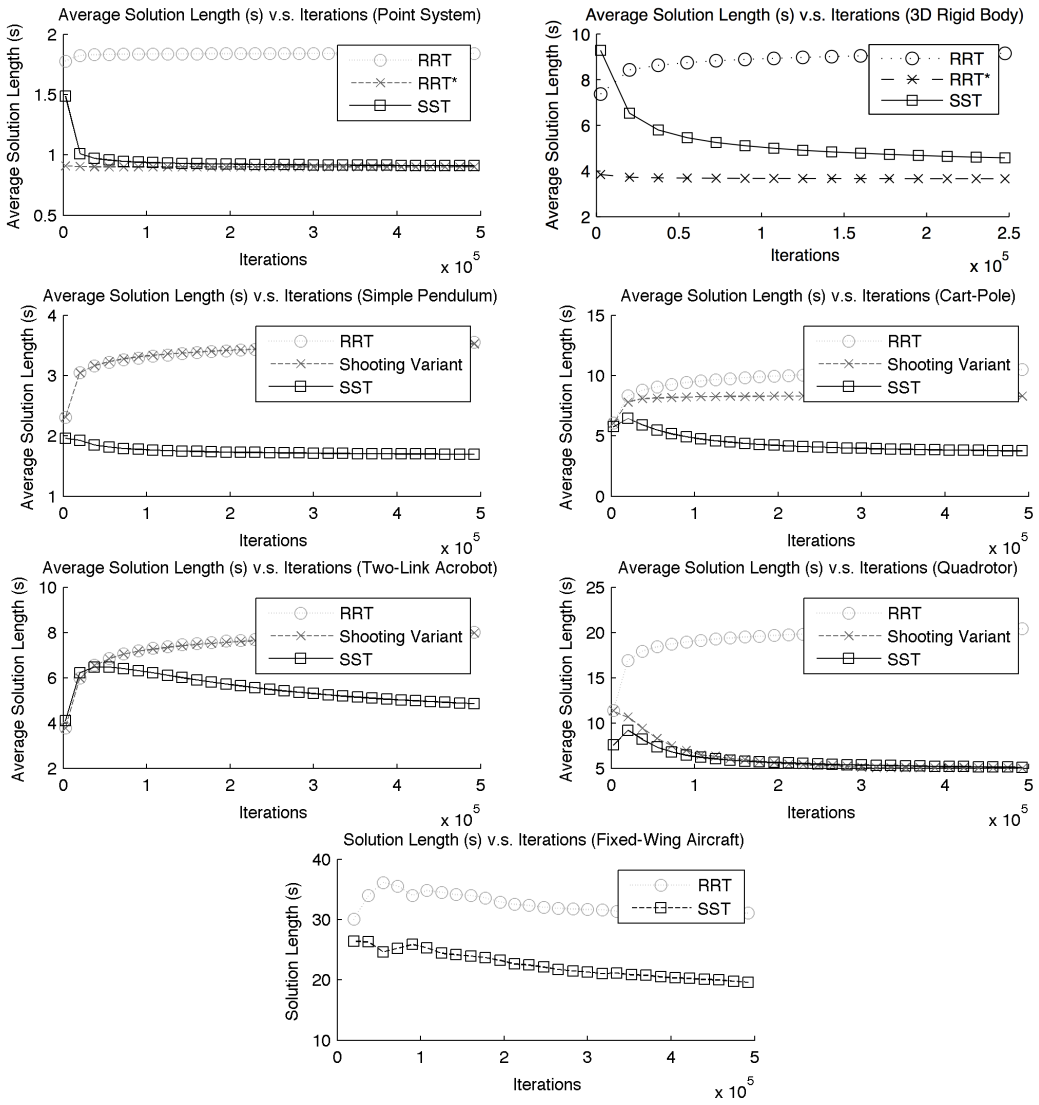

Figure 4.5: The average cost to each node in the tree for each algorithm (RRT, RRT* or the shooting approach, and SST). Results from [59]

Kinematic Point. A simple system for a baseline comparison. The state space is 2D (x, y), the control space is 2D (v, θ), and the dynamics are:

$$\dot{x} = v\cos(\theta) \qquad \dot{y} = v\sin(\theta).$$

3D Rigid Body. A free-flying rigid body. The state space is 6D $(x, y, z, \alpha, \beta, \gamma)$ signifying the space of SE(3) and the control space is 6D $(\dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma})$ representing

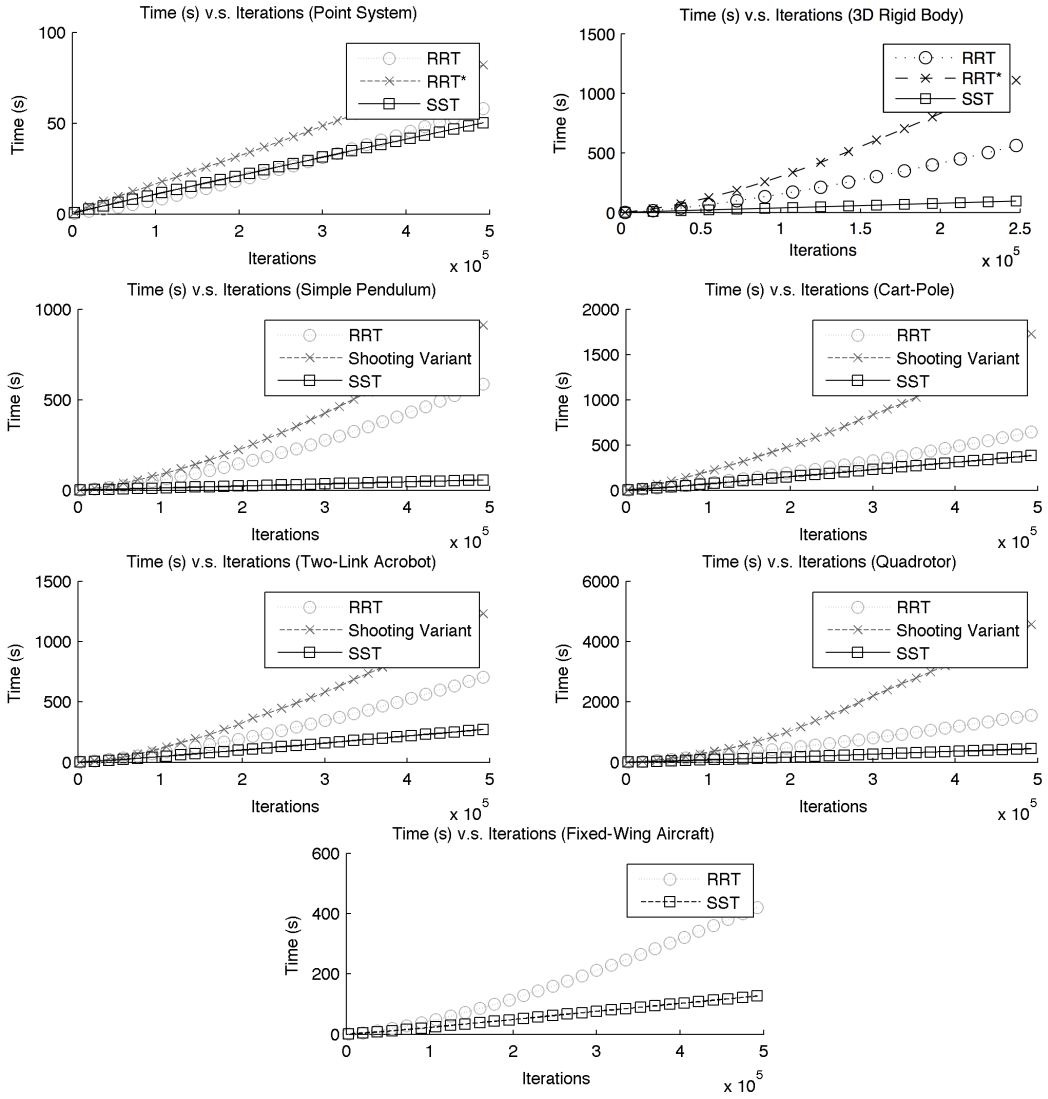

Figure 4.6: The time for execution for each algorithm (RRT, RRT* or the shooting approach, and SST). Results from [59]

the velocities of these degrees of freedom.

Simple Pendulum. A pendulum system typical in control literature. The state space is 2D $(\theta, \dot{\theta})$, the control space is 1D (τ), and the dynamics are:

$$\ddot{\theta} = \frac{(\tau - mgl * \cos(\theta) * 0.5) * 3}{ml^2}.$$

where $m = 1$ and $l = 1$.

Cart-Pole. Another typical control system where a block mass on a track has to balance a pendulum. The state space is 4D $(x, \theta, \dot{x}, \dot{\theta})$ and the control space is 1D (f)

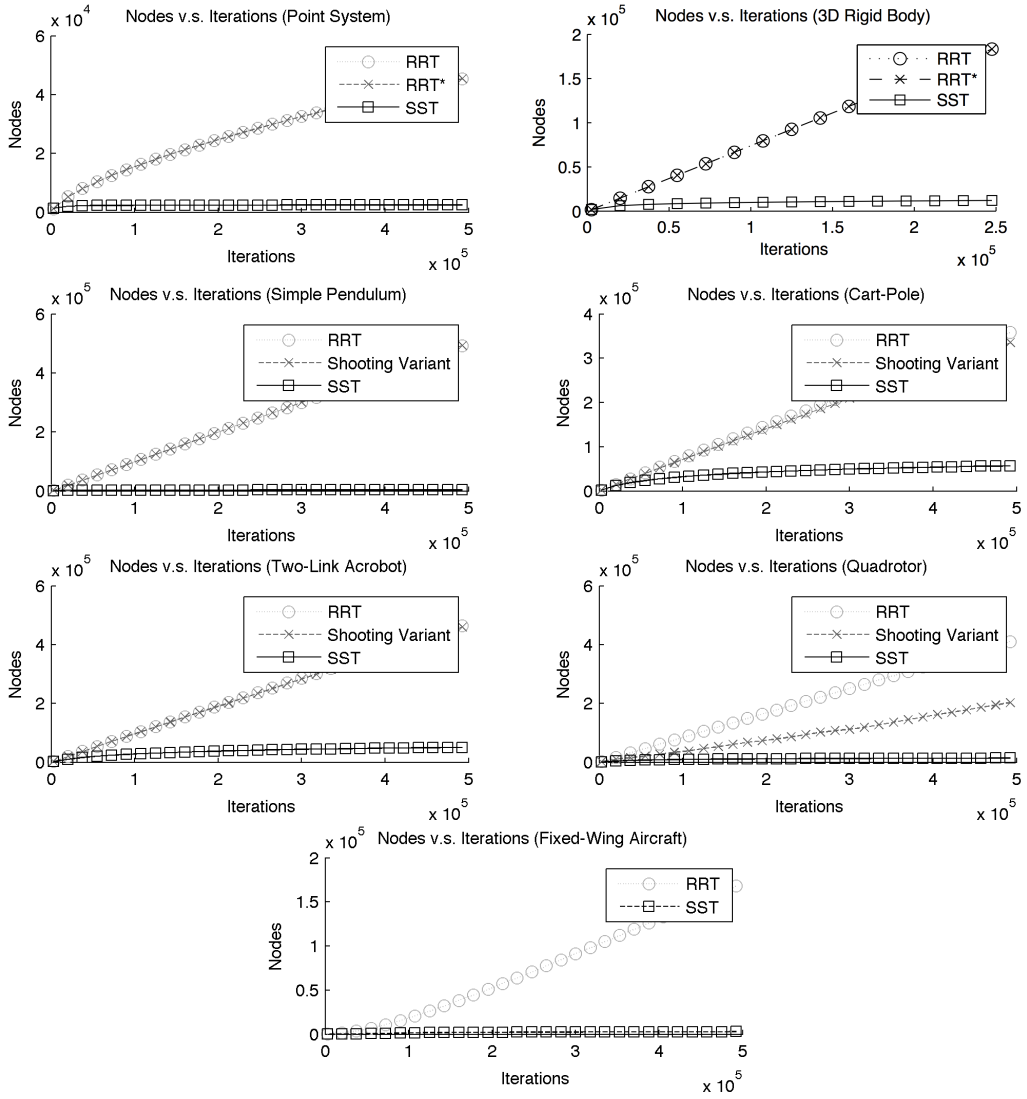

Figure 4.7: The number of nodes stored in each algorithm (RRT, RRT* or the shooting approach, and SST). Results from [59]

which is the force on the block mass. The dynamics are from [80].

Two-link Acrobot. The two-link acrobot model with a passive root joint. The state space is 4D $(\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2})$ and the control space is 1D (τ) which is the torque on the active joint. The dynamics are from [98].

Fixed-wing airplane. An airplane flying among cylinders. The state space is 9D $(x, y, z, v, \alpha, \beta, \theta, \omega, \tau)$, the control space is 3D $(\tau_{des}, \alpha_{des}, \beta_{des})$, and the dynamics are from [81].

Quadrotor. A quadrotor flying through windows. The state space is 12D

$(x, y, z, \alpha, \beta, \gamma, \dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma})$, the control space is 4D (w_1, w_2, w_3, w_4) corresponding to the rotor torques, and the dynamics are from [2].

Figures 4.5,4.6,4.7 illustrate the performance of SST compared with RRT, and either RRT* if the system is simple enough that a steering function is available, or RRT* with the shooting method [37]. In all cases, SST is able to improve path costs over time, contrary to RRT, or even RRT* with shooting in some cases. Looking at the cost of each iteration (in terms of computation time), SST is able to outperform even RRT in terms of computational cost. This is even when SST is performing more nearest neighbor queries than RRT, and this is due to the pruning operation keeping the number of nodes low, as evidenced by the plots comparing number of nodes.

4.3 Analysis

Recall from Section 3.3.4 that Sparse-RRT had difficulty using the Markov chain absorption argument to prove probabilistic completeness. This was related to the pruning operation causing new transitions in the chain to be possible, and being unable to quantify what the transition probabilities of those edges are. Now that the pruning regions are stable in SST, these extra transitions do not need to be drawn, and instead, a new lower bound on the selection probability can be determined. This, in turn, creates a new transition probability for moving from Markov state q_i to q_{i+1}. With that, SST is probabilistically complete.

In order for this probability lower bound to be computed, assumptions on the parameters δ_{BN} and δ_s relative to the robust clearance δ need to be made.

Proposition 1 *The parameters δ_{BN} and δ_s need to satisfy the following relationship given a robust clearance δ:*

$$\delta_{BN} + 2 \cdot \delta_s < \delta$$

.

Lemma 3 *Let $\delta_c = \delta - \delta_{BN} - 2\delta_s$ (guaranteed by Proposition 1). If a state $x_{new} \in V_{active}$ is generated at iteration n so that $x \in \mathcal{B}_{\delta_c}(x_i^*)$, then for every iteration $n' \geq n$, there is*

a state $x' \in V_{active}$ *so that* $x' \in \mathcal{B}_{(\delta-\delta_{BN})}(x_i^*)$ *and* $\mathbf{cost}(x') \leq \mathbf{cost}(x)$.

Lemma 3 is a formalization on the witness invariant discussed in Section 4.1. As long as a state was generated in the δ_c ball that covers the optimal path, there will always be a state in the $\delta - \delta_{BN}$ ball covering the optimal path. This allows the Markov chain to not require backward transitions.

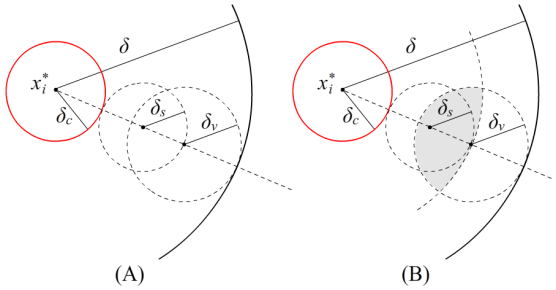

Figure 4.8: By sampling in the highlighted region (which exists given Proposition 1), it is always possible to select a node in the $\mathcal{B}_\delta$ around the optimal path.

Proof: Given x, a node generated by `SST`, then it is guaranteed that a witness point s is located near x. The witness point s can be located, in the worst case, at distance δ_s away from the boundary of $\mathcal{B}_{\delta_c}(x_i^*)$ if $x \in \mathcal{B}_{\delta_c}(x_i^*)$.

Note that x can be removed from $\mathbb{V}_{active}$ by `SST` in later iterations. In fact, x almost surely will be removed if $x \neq x_\circ$. It is possible that when x is removed, there could be no state in the ball $\mathcal{B}_{\delta_c}(x_i^*)$. Nevertheless, the witness sample s will not be deleted. A node x' representing s will always exist in $\mathbb{V}_{active}$ and x' will not leave the ball $\mathcal{B}_{\delta_s}(s)$. It is guaranteed by `SST` that the cost of the x' will never increase, i.e., $\mathbf{cost}(x') \leq \mathbf{cost}(x)$. In addition, x' has to exist inside $\mathcal{B}_{\delta-\delta_{BN}}(x_i^*) = \mathcal{B}_{\delta_c+2\delta_s}(x_i^*)$. ■

Now that it is guaranteed that once a node is in a covering ball of the optimal path, it is possible to quantify the probability of selecting that node.

Lemma 4 *Assuming uniform sampling in the* `Sample` *function of* `BestNear`*, if* $\exists\ x \in \mathbb{V}_{active}$ *so that* $x \in \mathcal{B}_{\delta_c}(x_i^*)$ *at iteration* n*, then the probability that* `BestNear` *selects for propagation a node* $x' \in \mathcal{B}_\delta(x_i^*)$ *can be lower bounded by a positive constant* $\gamma_{\texttt{SST}}$ *for every* $n' > n$.

Proof: `BestNear` performs uniform random sampling in $\mathbb{X}$ to generate x_{rand}, and then

examines the ball $\mathcal{B}_{\delta_{BN}}(x_{rand})$ to find the best path cost node. In order for a node in $\mathcal{B}_{\delta}(x_i^*)$ to be returned, the random sample needs to be in $\mathcal{B}_{\delta-\delta_{BN}}(x_i^*)$. If the sample is outside this ball, then a node not in $\mathcal{B}_{\delta}(x_i^*)$ can be considered, and therefore may be selected.

Next, consider the size of the intersection of $\mathcal{B}_{\delta-\delta_{BN}}(x_i^*)$ and a ball of radius δ_{BN} that is entirely enclosed in $\mathcal{B}_{\delta}(x_i^*)$. Let x_v denote the center of this ball. This intersection represents the area that a sample can be generated so as to return a state from ball $\mathcal{B}_{\delta-\delta_{BN}}(x_i^*)$. In the worst case, the center of this ball $\mathcal{B}_{\delta_{BN}}(x_v)$ could be on the border of $\mathcal{B}_{\delta-\delta_{BN}}(x_i^*)$. Then, the probability of sampling a state in this region can be computed as:

$$\gamma_{\texttt{SST}} = \frac{\mu(\mathcal{B}_{\delta-\delta_{BN}}(x_i^*) \cap \mathcal{B}_{\delta_{BN}}(x_v))}{\mu(\mathbb{X})}$$

. This is the smallest region that will guarantee selection of a node in $\mathcal{B}_{\delta}(x_i)$, and is illustrated in Figure 4.8. ■

With a lower bound of $\gamma_{\texttt{SST}}$, the transition probability in the Markov chain from state q_i to q_{i+1} is now $\rho = \gamma_{\texttt{SST}}\gamma_{\delta}$.

4.4 Asymptotic Near-Optimality of SST

Throughout this analysis, the hypothetical trajectory used for proving probabilistic completeness has been assumed to be the optimal trajectory with δ-robust clearance. Up until now, this has not be a requirement has not been explicitly used. Now, a bound on the cost of the path generated by SST will be examined, and this cost is relative to cost of the optimal trajectory g^*.

Theorem 5 SST *is asymptotically near-optimal*

Proof: Let $\pi(x'_{i-1} \rightarrow x_i)$ denote the δ-similar trajectory segment generated by SST where $x'_{i-1} \in \mathcal{B}_{\delta}(x^*_{i-1})$ of the optimal path and $x_i \in \mathcal{B}_{\delta_{BN}}(x_i^*)$. Theorem 2 guarantees the probability of generating this trajectory by MonteCarlo-Prop can be lower bounded as γ_{δ}. Then from the definition of δ-similar trajectories and *Lipschitz continuity* for

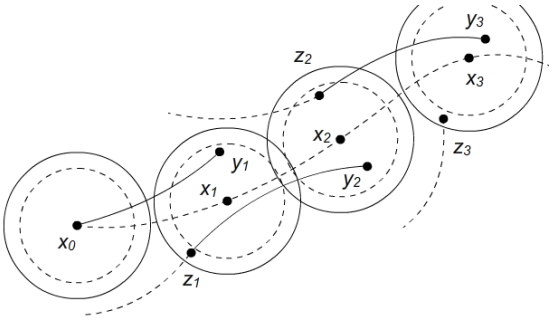

Figure 4.9: Along the optimal path, as long as a node enters the region around that path, there will always be a node there, either the original node or one with better path cost.

cost:

$$\mathbf{cost}(\pi(x'_{i-1} \to x_i)) \leq \mathbf{cost}(\pi(x^*_{i-1} \to x^*_i)) + K_c \cdot \delta. \tag{4.1}$$

Lemma 4 guarantees that when x_i exists in $\mathcal{B}_{\delta-\delta_{BN}}(x^*_i)$, then x'_i, returned by the `BestNear` function with probability $\gamma_{\texttt{SST}}$, must have equal or less cost, i.e., x'_i can be the same state as x_i or a different state with smaller or equal cost:

$$\mathbf{cost}(x'_i) \leq \mathbf{cost}(x_i). \tag{4.2}$$

Consider the second covering ball in the sequence, $\mathcal{B}_\delta(x^*_1)$. According to Equation 4.1 and Equation 4.2:

$$\mathbf{cost}(\pi(x_0 \to x'_1)) \leq \mathbf{cost}(\pi(x_0 \to x_1)) \leq \mathbf{cost}(\overline{x_0 \to x^*_1}) + K_c \cdot \delta$$

.

Assume this is true for k segments,

$$\mathbf{cost}(\pi(x_0 \to x'_k)) \leq \mathbf{cost}(\pi(x_0 \to x^*_k)) + k \cdot K_c \cdot \delta.$$

Then, the cost of the trajectory with $k+1$ segments is:

$$\begin{aligned}
\mathbf{cost}(\pi(x_0 \to x'_{k+1})) &\leq \mathbf{cost}(\pi(x_0 \to x_{k+1})) \\
&= \mathbf{cost}(\pi(x_0 \to x'_k)) + \mathbf{cost}(\pi(x'_k \to x_{k+1})) \\
&\leq \mathbf{cost}(\pi(x_0 \to x^*_k)) + kK_c\delta + \mathbf{cost}(\pi(x'_k \to x_{k+1})) \\
&\leq \mathbf{cost}(\pi(x_0 \to x^*_k)) + kK_c\delta + \mathbf{cost}(\pi(x^*_k \to x^*_{k+1})) + K_c\delta \\
&= \mathbf{cost}(\pi(x_0 \to x^*_{k+1})) + (k+1)K_c\delta.
\end{aligned}$$

By induction, this holds for all k. The index k is upper bounded by the number of covering balls M (which was derived by a cost differential g_Δ).

$$\mathbf{cost}(\pi(x_0 \to x'_M)) \leq \mathbf{cost}(\pi(x_0 \to x^*_M)) + M \cdot K_c \cdot \delta = (1 + \frac{K_c \cdot \delta}{g_\Delta}) \cdot g^*.$$

Therefore, `SST` is an asymptotically near-optimal algorithm with

$$Bound(g^*) = (1 + \frac{K_c \cdot \delta}{g_\Delta}) \cdot g^* \geq g^*$$

■

The bound on the path cost is dependent on the Lipschitz constant of the cost function, the robust clearance value, and the cost differential. The cost differential is mainly determined via the propagation time in `MonteCarlo-Prop`, and it essentially determines how many edges are needed to generate a trajectory to the goal region.

Chapter 5

Planning Under Uncertainty: A Case Study

Now that the SST algorithm has been presented, it can be used to study a particularly interesting problem domain, planning under uncertainty. This problem domain plans in a *belief space* where instead of states representing a robot's current position, they are now represented as probability distributions of where the robot could be. The source of such uncertainty can be anything from sensor noise making the state of the robot inaccurate, actuator error which results in unpredicted behavior, or imprecise environmental models. The content in this chapter is adapted from material from a previous paper [64].

5.1 Planning Under Uncertainty Preliminaries

There are many methods that aim to plan specifically in belief space [53, 85, 87, 106, 31, 5, 48, 107]. This is exciting because planning in belief space is doubly exponentially more difficult than planning in state space, because the space of possible probability distributions is large.

Sampling-based methods once again provide a useful tool to handling this problem, due to their inherent ability to work well with high dimensional spaces. As examples, some applications of sampling-based methods, along with a belief space abstraction of Gaussian probability distributions have been used previously [9, 71, 87]. One of the most important algorithmic choices in this domain however is what distance function to use. Many methods even rely heavily on this function for a variety of reasons, such as sampling or pruning [47, 53, 103].

The SST algorithm provides an opportunity to evaluate different distance function choices when planning in belief space. Since SST is not built to exploit any explicit

structure in a belief space planning problem, the main evaluation criteria is the distance function, of which SST makes heavy use. In addition, SST is only able to plan in belief space due to its reliance on forward propagation instead of steering. Steering requires knowledge about how the probability distributions can be constructed and generally requires those distributions to have certain parameterizations. SST is a good impartial testbed to evaluate different distance functions for belief space planning.

Commonly used distance functions, such as L1 and the Kullback-Leibler divergence (KL), make an important assumption that the underlying state space does not affect the distance measure. So, in the event that the probability distribution parameterization results in a finite region where probability mass is gathered, two distributions that don't ''overlap are considered to be infinitely far away from each other. A different distance function, the Wasserstein or Earth Mover's Distance (EMD) does make use of distance in state space, but has been sparingly used [52, 54].

In order to evaluate these distance functions, and to allow for SST to be applied to belief space planning, the type of problem considered here is a Non-Observable Markov Decision Process (NOMDP), or otherwise referred to as conformant planning. As the name suggests, this problem class does not have an observation model. In addition, when solving a Partially-Observable Markov Decision Process (POMDP), the result of the planning process is a policy, where in a NOMDP, the result is a nominal path, the same result as a planner operating in state space. This means that solving a NOMDP can focus on minimizing the cost of this nominal path, making SST a prime candidate for solving this type of problem.

5.2 Distance Functions in Belief Space

A distance in belief space is between two different belief distributions $b, b' \in \mathcal{B}(\mathbb{X})$ and aims to measure how far away these two distributions are from one another. Four different choices for distances in belief space are evaluated, two of which are commonly used for belief space planning, and the other two are not, but are used for computer vision or optimal transport problems.

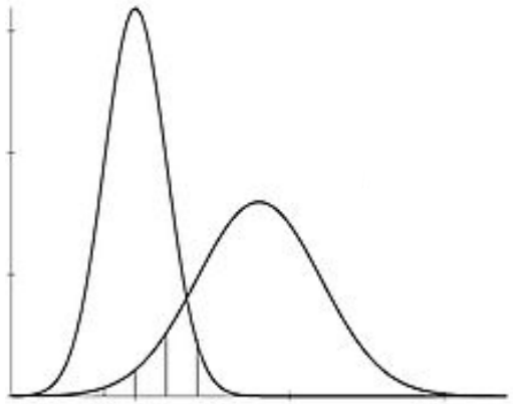

Figure 5.1: Illustration of the `L1` distance.

A commonly used distance function for belief space planning is the `L1` distance:

$$\mathbf{dist}_{\texttt{L1}}(b, b') = \int_{x \in \mathbb{X}} |b(x) - b'(x)| \, dx.$$

In practice, to compute this distance, the state space is discretized, which allows for this distance to be computed for each "bin" that was created and summing everything together. The accuracy of this distance is dependent on the resolution of the discretization.

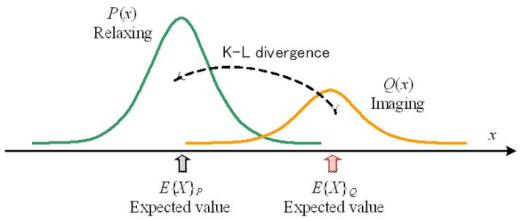

Figure 5.2: Illustration of the Kullback-Leibler (KL) divergence.

Another common distance function is the Kullback-Leibler divergence (`KL`), which in general measures the difference in information between two probability distributions:

$$\mathbf{dist}_{\texttt{KL}}(b, b') = \int_{x \in \mathbb{X}} b(x) \left(\ln b(x) - \ln b'(x) \right) \, dx. \tag{5.1}$$

while this distance does not generally consider the state space distance in its calculation,

when the underlying distributions are Gaussian, the state space distance appears in the difference of the means, i.e. when $b_1 = \mathcal{N}(\mu_1, \Sigma_1)$ and $b_2 = \mathcal{N}(\mu_2, \Sigma_2)$, then

$$\mathbf{dist}_{\mathtt{KL}}(b_1, b_2) = \frac{1}{2}\left((\mu_2 - \mu_1)^T \Sigma_1^{-1} (\mu_2 - \mu_1) + tr\left(\Sigma_2^{-1}\Sigma_1\right) + \ln\frac{|\Sigma_2|}{|\Sigma_1|} - K\right), \quad (5.2)$$

where K is the dimension of the underlying state space. Of note, the KL divergence is not symmetric, while distance functions are defined to be symmetric. Instead, the distance between the distributions means can be used, i.e. $\frac{\mathbf{dist}_{\mathtt{KL}}(b, \frac{b+b'}{2}) + \mathbf{dist}_{\mathtt{KL}}(b', \frac{b+b'}{2})}{2}$. This is also known as the Jensen-Shannon divergence. Both the Jensen-Shannon divergence and an approximation using Gaussians are evaluated.

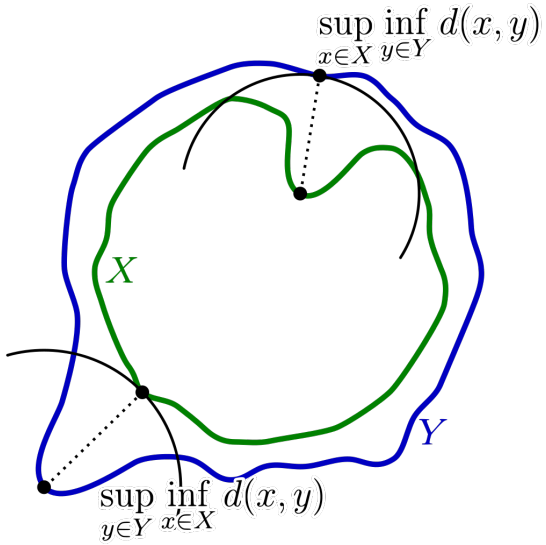

Figure 5.3: Illustration of the Hausdorff distance. Image attributed to [90].

Coming from the computer vision domain, the Hausdorff distance is also considered. This distance is between two sets and uses a max-min operation over those sets where they have nonzero probability (otherwise called their support). For probability distributions, this is defined as

$$\mathbf{dist}_H(b, b') = \max\{d_H(b, b'), d_H(b', b)\}$$

$$\text{where } d_H(b, b') = \max_{x \in support(b)} \left\{ \min_{x' \in support(b')} \{\mathbf{dist}(x, x')\} \right\},$$

and $support(b) = \{x \in \mathbb{X} \mid b(x) > 0\}$. Notice that in order to make this distance

symmetric, both directions need to be calculated.

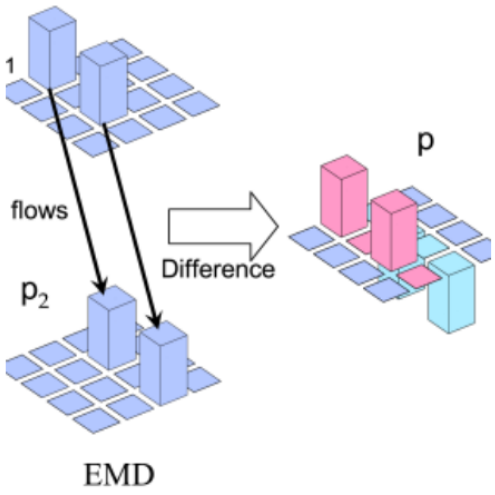

Figure 5.4: Illustration of the Earth Mover's Distance. Image from [92]

Finally, the Wasserstein distance is considered, more commonly known as the Earth Mover's Distance (`EMD`). This distance is analogous to computing the amount of work to move the probability mass from one distribution to another. The farther away the two piles are, the more work is necessary. Formally, this is defined by:

$$\begin{aligned} \mathbf{dist}_{\texttt{EMD}}(b, b') &= \inf_{f} \left\{ \int_{x \in \mathbb{X}} \int_{x' \in \mathbb{X}} \mathbf{dist}(x, x') f(x, x') \partial x \partial x' \;\middle|\; \right. \\ &\qquad \left. b = \int_{x'} f(x, x') dx' \;,\; b'(x') = \int_{x} f(x, x') dx \right\}, \end{aligned} \tag{5.3}$$

where f is a joint density function.

5.3 Experimental Performance

All of these previous results were presented in published works [64]. All distances are evaluated in the scenarios shown in Figure 5.5. The 2D rigid body (left) must move from the left to the right side. The car (left middle) must drive through one of 3 corridors. The fixed-wing airplane (middle right), must move to the opposite corner while avoiding cylindrical columns. The planar manipulator (right) must push the

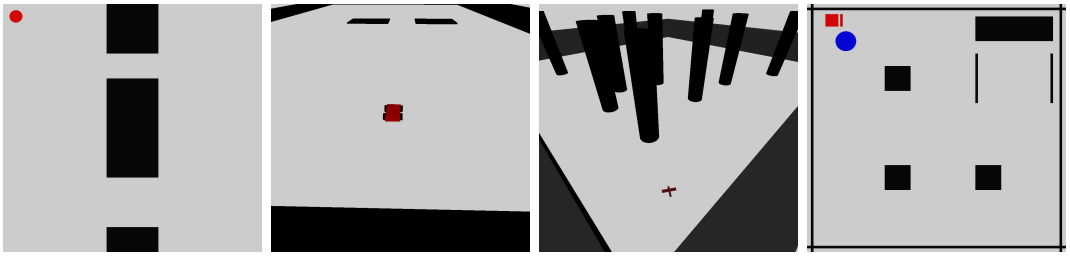

Figure 5.5: The scenarios for evaluating distances for planning in belief space (from [64]).

round object into the storage location at the top right. All scenarios produce non-Gaussian belief distributions due to the nonlinear dynamics, so these distributions are represented via a particle representation and are binned into cells in the state space. These binned cells and particle counts are then the inputs to the distance functions. The exception to this process is the KL-divergence that approximates the distribution as a Gaussian. The motion planning objective is to reach a goal region in state-space with at least 90% probability (90% of the particles are in the goal region). Valid trajectories have a collision probability of less than 20%, and any trajectory that would collide more than that is not added to the search tree.

2D Rigid Body. This introductory example is a 2D rigid body moving among two narrow corridors. Due to errors in actuation and requirement for collision avoidance, the robot can only move through the lower corridor. The state space is 2D (x, y) and the control space is also 2D (v, θ), where $v \in [0, 10]$ and $\theta \in [-\pi, \pi]$. The dynamics follow this model:

$$\dot{x} = \tilde{v}\cos(\tilde{\theta}), \quad \dot{y} = \tilde{v}\sin(\tilde{\theta}),$$

where $\tilde{v} = v + \mathcal{N}(0, 1)$ and $\tilde{\theta} = \theta + \mathcal{N}(0, 0.3)$. Numerical integration is performed using Euler integration.

2^{nd}-order Car. A four-wheeled vehicle with dynamics needs to reach a goal region, while ensuring low collision probability. The state space is 5D $(x, y, \theta, v, \omega)$, the control space is 2D $(a, \dot{\omega}) \in ([-1, 1], [-2, 0.2])$, actuation error is $(\mathcal{N}(0, 0.05), \mathcal{N}(0, 0.002))$, and the dynamics are:

$$\dot{x} = v\cos(\theta)\cos(\omega), \quad \dot{y} = v\sin(\theta)\cos(\omega),$$
$$\dot{\theta} = v\sin(\omega), \qquad \dot{v} = \tilde{a}.$$

Numerical integration is performed using Runge-Kutta order four (RK4). There are multiple feasible paths through each of the 3 corridors.

Fixed-wing airplane. An airplane flying among multiple cylinders. The state space is 9D $(x, y, z, v, \alpha, \beta, \theta, \omega, \tau)$, the control space is 3D ($\tau_{des} \in [4, 8]$, $\alpha_{des} \in [-1.4, 1.4]$, $\beta_{des} \in [-1, 1]$) and the dynamics are (from [81]):

$$\dot{x} = v\cos(\omega)\cos(\theta), \qquad \dot{y} = v\cos(\omega)\sin(\theta)), \qquad \dot{z} = v\sin(\omega),$$
$$\dot{v} = \tau * \cos(\beta) - C_d k v^2 - g\sin(\omega), \qquad \dot{\omega} = \cos(\alpha)(\frac{\tau\sin(\beta)}{v} + C_l k v) - g\frac{\cos(\omega)}{v},$$
$$\dot{\theta} = v\frac{\sin(\alpha)}{\cos(\omega)}(\frac{\tau\sin(\beta)}{v} + C_l k v), \qquad \dot{\tau} = \widetilde{\tau_{des}} - \tau \quad \dot{\alpha} = \widetilde{\alpha_{des}} - \alpha, \quad \dot{\beta} = \widetilde{\beta_{des}} - \beta,$$

where $\widetilde{\tau_{des}} = \tau_{des} + \mathcal{N}(0, 0.03)$, $\widetilde{\alpha_{des}} = \tau_{des} + \mathcal{N}(0, 0.01)$, and $\widetilde{\beta_{des}} = \beta_{des} + \mathcal{N}(0, 0.01)$. Numerical integration is performed using RK4. This problem has a state space that is generally larger than most planners in belief space can handle computationally. Leveraging sampling-based techniques with proper distance functions makes planning for the airplane model possible.

Non-prehensile manipulator. The task is to push an object to the goal. The state space is 5D $(x_{man}, y_{man}, x_{obj}, y_{obj}, \theta_{manip})$ and the control space is 2D (v, θ), where $v \in [0, 10]$ and $\theta \in [-\pi, \pi]$. The dynamics are:

$$\dot{x}_{man} = \tilde{v}\cos(\tilde{\theta}), \qquad \dot{y}_{man} = \tilde{v}\sin(\tilde{\theta}),$$

where $\tilde{v} = v + \mathcal{U}(-1, 1)$ and $\tilde{\theta} = \theta + \mathcal{U}(-0.3, 0.3)$. Numerical integration is performed using RK4. The object cannot be moved unless the manipulator moves in the direction of the object and pushes it, which implies that there is contact between the manipulator and the object. Once pushed, the object moves as if it is attached to the manipulator. Notice that the noise model used in this setup is a uniform distribution, meaning that

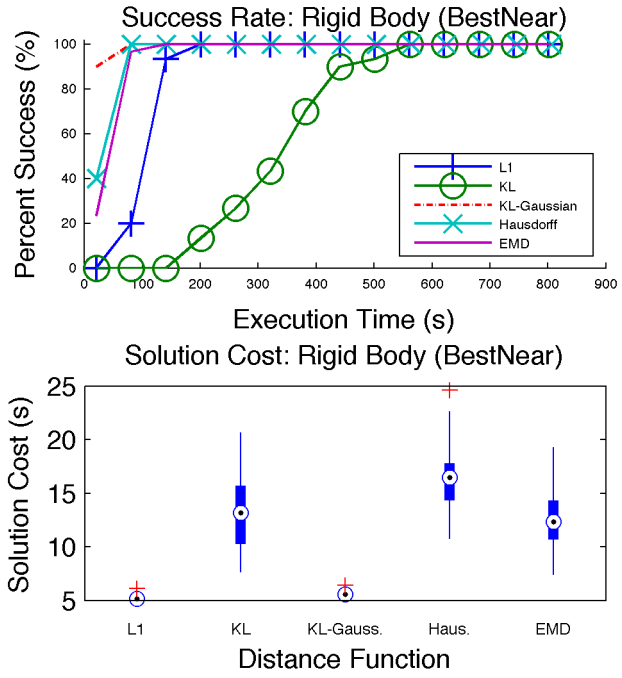

Figure 5.6: Distance comparison results for the 2D rigid body.

the resulting belief distributions are clearly non-Gaussian.

In order to evaluate each distance function in each domain, the success rate of SST in finding solutions and the quality of those solutions are compiled. Arguably, the rate of finding solutions is the most important metric here, since large computational cost of planning in belief space does not allow much extra computation time to optimizing an existing solution. Nevertheless, this quality data is also provided. All of these results are averaged over 30 independent SST runs.

In most scenarios, belief metrics that do consider the underlying state-space distance, such as EMD, Hausdorff, and KL-Gaussian, perform significantly better than those that do not. Such consideration is sufficient when the state space is small (as in Figure 5.6). As the size of the state space increases, this is no longer sufficient. The path costs achieved by the different metrics for successful runs are comparable, but there is a significant difference among the distance functions for when solutions are actually found.

In the spirit of the development of SST, it is useful to reexamine existing assumptions

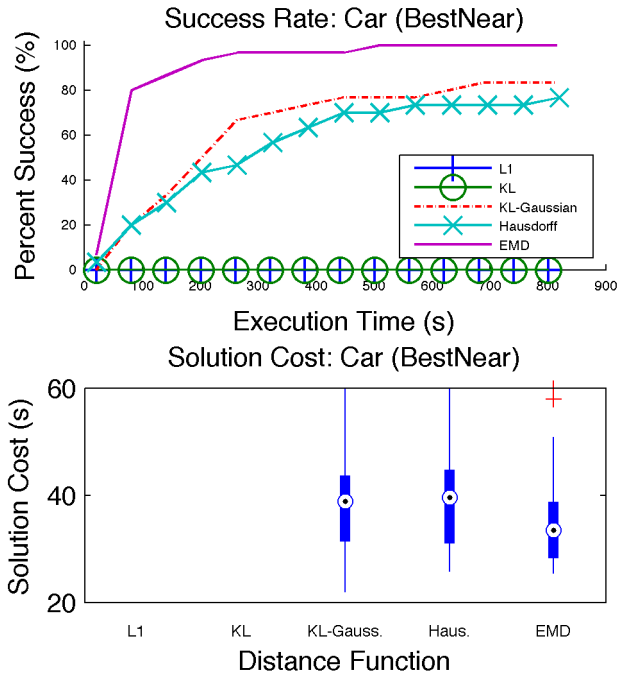

Figure 5.7: Distance comparison results for the car. L1 and KL failed to produce solutions.

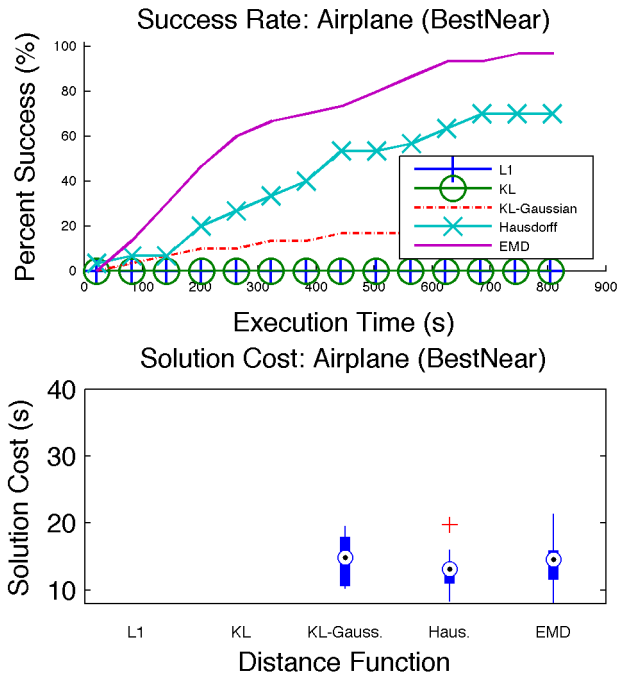

Figure 5.8: Distance comparison results for the airplane. L1 and KL failed to produce solutions.

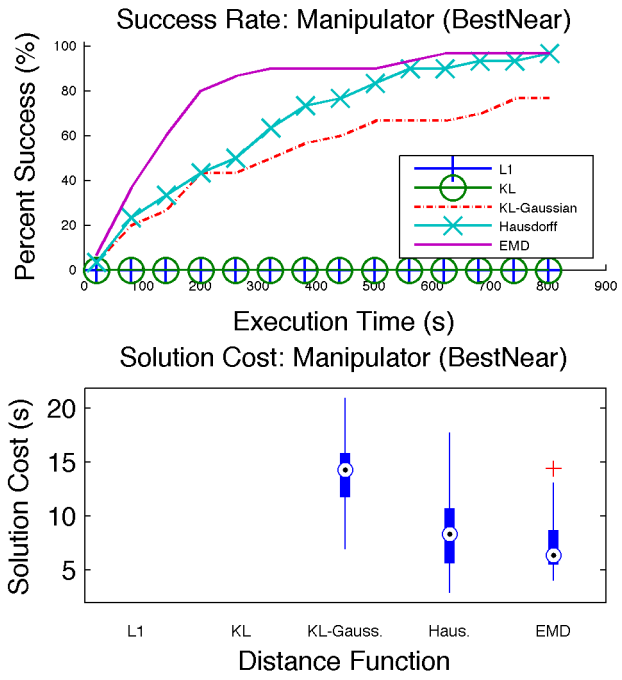

Figure 5.9: Distance comparison results for the manipulator. L1 and KL failed to produce solutions.

and methods to see if new perspectives can provide new conclusions. Just as reevaluating the `BestNear` primitive in the context of asymptotically-near optimal planners resulted in a performant motion planning algorithm with provable guarantees, questioning the distance function choice in belief space planning is a useful exercise. Knowing the advantages and disadvantages of a distance function choice is paramount to getting planners to work well in practice, in addition to the local planner choice.

Chapter 6

Informed Kinodynamic Planning with Guarantees

SST has good practical performance in some problem domains. SST does have some drawbacks however. First, there are the two user-defined parameters, δ_{BN} and δ_s, that need to be tailored to the particular environment, and are restricted by the duration of propagation. If the pruning radius is larger than the largest possible edge length, no edges can be added. If the selection radius is smaller than the pruning radius, the BestNear procedure will not be likely to examine multiple nodes in each selection step, negating its benefit of comparing multiple nodes.

Second, recall that the goal of Sparse-RRT and SST was to introduce more exploitation for path cost into RRT-based motion planners. The addition of this exploitation is at the cost of the pure Voronoi exploration bias of RRT. In addition to this cost, SST does not have any goal-finding exploitation to find solutions quickly.

Another source of poor goal-finding exploitation is inherent to using the MonteCarlo-Prop procedure for expanding nodes. Since the use of random controls does not directly take the system toward the goal, but instead just lets the flow of the system dynamics move the system, tt may take many iterations to attempt a control that avoids obstacles and moves toward the goal. Having a mechanism that splits the difference between MonteCarlo-Prop and the steering methods required by RRT* should help alleviate this issue.

6.1 Incorporating Task Space Heuristics

This section aims to provide more exploitation for finding goals into a sampling-based planner similar to SST. As discusses previously, a common and easy to implement exploitative modification is goal-biasing the selection primitive. The random sample

step in the selection primitive is replaced with "sampling" a state in the goal region with some probability. This makes selection biased toward nodes that are closer to the goal, thus increasing the likelihood that the goal will be reached in less iterations.

Another more effective method is to introduce some task space heuristic function. This is similar to search-based methods like A*, where in addition to the cost values, g, there are heuristic values, h. The cost value represents true cost that a trajectory has accumulated from its initial state, and the heuristic value represents the expected cost to get from the end of that trajectory to the goal region. If you are using the A* algorithm, the heuristic function, **H**, needs to be consistent and admissible. Formally, this is denoted as

$$\mathbf{H}(x \to \mathbb{X}_G) \leq \mathbf{cost}^*(x \to \mathbb{X}_G), \forall x \in \mathbb{X}_f$$

$$\mathbf{H}(\mathbb{X}_G \to \mathbb{X}_G) = 0$$

$$\mathbf{H}(x \to z) \leq \mathbf{H}(x \to y) + \mathbf{H}(y \to z)$$

For ease of notation, the heuristic function **H** with one argument has the implicit second argument of $\mathbb{X}_G$, since that that is the most common argument. It is also important to note how similar **H** is to the distance function **dist**. In fact, there are similar needs between both **dist** and **H**. Both functions want to estimate the cost between states as closely as possible. One strategy that is commonly used is to use a task space abstraction for this heuristic function. A task space, $\mathbb{T}$, is a transformation of the state space, $\mathbb{X}$, via an embedding function $\mathbb{F} : \mathbb{X} \to \mathbb{T}$. As long as the properties above are satisfied for a given cost function and task embedding, a heuristic function can make use of a distance function in task space instead of state space.

6.1.1 Prioritizing Selection with Heuristic Values

A* makes use of both cost and heuristic values to prioritize what nodes are expanded during its search process, and does so by putting unexpanded nodes into a priority

queue and sorting that queue by $f = g + h$, thus prioritizing lower f-values. In the context of sampling-based planners, since selection of a node needs to occur many times over the course of an algorithm's execution loop, using a priority queue may not be the right solution. The sampling-based selection process is one of the strengths of sampling-based planners (its in the moniker for the algorithm class). The strategy of using f-values for selection has already been applied in the RRT* family of algorithms [24, 25, 13].

Trying to apply this philosophy onto SST, and addressing the issue of hand-tuned δ_{BN} and δ_s, it is necessary to construct a different selection and pruning mechanism that makes use of both trajectory cost and heuristic cost. By removing the constraint of a user-provided parameter, additional flexibility is gained where every node can freely determine how good that node is. Nodes that have lower f-values should get higher priority than those with higher f-values. During a random selection process, this corresponds to having higher probability of selecting lower f-value nodes. Moreover, one way to accomplish this goal is to allow for the lower f-value nodes to have larger "regions of influence" where those nodes can be selected, whereas higher f-value nodes should have smaller regions. These regions that each node maintains are "dominance" regions that are informed by these f-values.

Each node maintains a dominance informed region (DIR) by looking at nearby nodes and determining the distance to the closest node that has a lower f-value. Then, at any given time, each node knows that it is the best node in a ball of that radius. Then, when a new node is added during a motion planner iteration, these dominance regions are updated to account for this new node, in addition to the new node calculating its dominance region.

Recall that the BestNear procedure looks at a region, examines all nodes in that region, and returns the one with the best path cost. An analogous operation that uses the dominance regions follows a process as follows:

- Randomly sample a state.
- Determine each dominance region that contains that random sample.
- If that set is empty, find the closest node to the random sample, and repeat this

Algorithm 15: DIR_Selection(G)

$x_{rand} \leftarrow$ `Sample`($\mathbb{X}$);
$X_{cand} \leftarrow \{x \in G | x_{rand} \in$ `DIR`$(x)\}$;
if $|X_{cand}| = 0$ **then**
    $x_{closest} \leftarrow$ `NN.Nearest`(x_{rand});
    $X_{cand} \leftarrow \{x \in G | x_{closest} \in$ `DIR`$(x)\}$;
return $x \in X_{cand}$ uniformly at random;

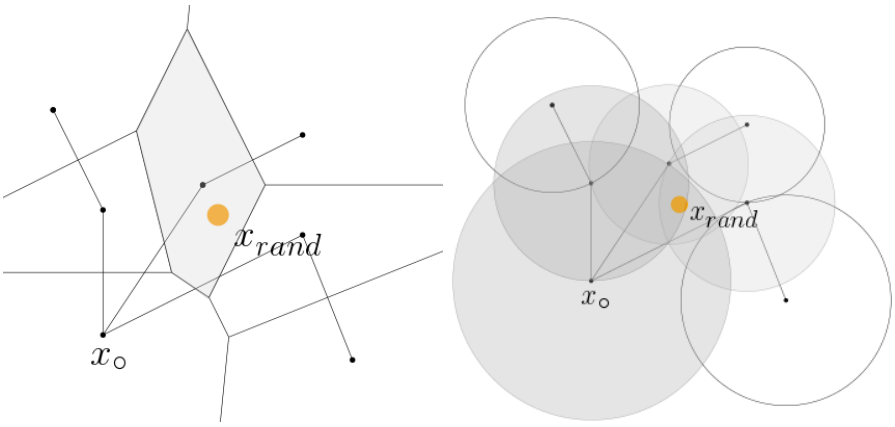

Figure 6.1: Comparison of how nodes are selected in `RRT` and `DIRT`. (left) An `RRT`-like, Voronoi-based selection would select the closest node to the random sample. (right) The `DIR`-based method selects equally among all nodes that contain the random sample in their `DIR`. The `DIR` radius is dependent on $\mathbf{cost}(x)+\mathbf{H}(x)$ values at each node relative to those nearby.

process using that node as the random sample.

- When the set of dominance regions is non-empty, uniformly at random select among those nodes.

The algorithm for this selection strategy is provided in Algorithm 15.

6.1.2 Updating Dominance Regions

As previously mentioned, whenever a new node is added into the search tree, all of the surrounding dominance regions need to be updated. Those existing nodes may not be as good as the new node that was added. Conversely, the existing nodes will influence the size of the dominance region of the new node. This process is formalized in Algorithm 16.

Ideally, there should also be a version of Algorithm 16 that can also prune away

Algorithm 16: Update_Dominance_Regions(G,$x_{selected}$,x_{new})

1 $X_{up} \leftarrow \{x \in G | \mathbf{dist}(x, x_{new}) \leq \mathbf{dist}(x_{selected}, x_{new})\}$;
2 $\forall x \in X_{up}$ s.t.
$\mathbf{cost}(x) + \mathbf{H}(x) > \mathbf{cost}(x_{new}) + \mathbf{H}(x_{new}), \texttt{DIR}(x) \leftarrow \mathcal{B}(x, \mathbf{dist}(x, x_{new}))$;
3 $\texttt{DIR}(x_{new}) \leftarrow \max_{x \in X_{up}} (\mathbf{dist}(x_{new}, x)$ s.t. $\mathbf{cost}(x) + \mathbf{H}(x) > \mathbf{cost}(x_{new}) + \mathbf{H}(x_{new})$;

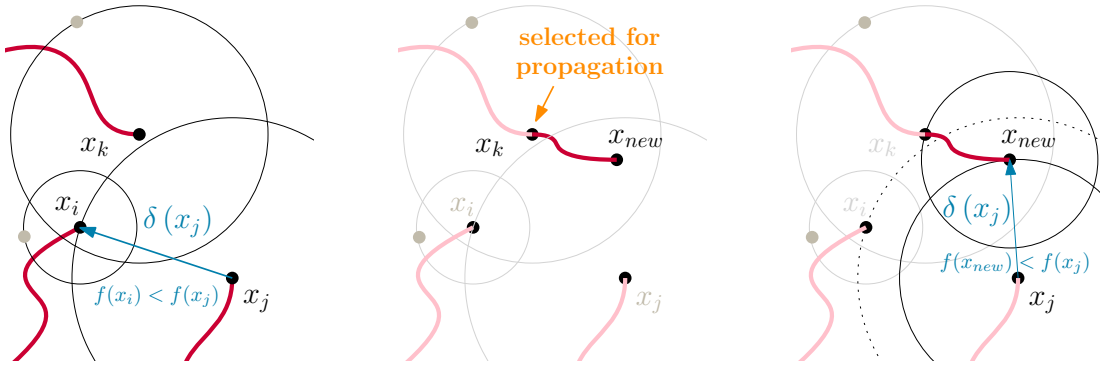

Figure 6.2: Computing a new DIR and updating others. A new node is added, which reduces the size of $\texttt{DIR}(x_j)$. Because the new node is not close enough to x_i, $\texttt{DIR}(x_i)$ is not effected. The dotted line denotes the old DIR, and the solid lines are the current DIR.

unneeded nodes. This way, an algorithm that uses dominance regions can gain the computational benefits from a smaller number of nodes. One intuitive method to accomplish this goal is to remove nodes that have dominance regions that are small enough to be completely subsumed by another dominance region, i.e. $\texttt{DIR}(x') \subset \texttt{DIR}(x)$. This version on updating dominance regions is provided in Algorithm 17.

6.2 Dominance Informed Region Trees (DIRT)

Now that a new parameterless selection and pruning scheme have been presented, the algorithm, Dominance Informed Region Trees (DIRT) can be presented. DIRT is an algorithm that replaces the somewhat unintuitive parameters of δ_{BN} and δ_s from SST with more human legible parameters of task-space heuristic functions and edge generation primitives.

Algorithm 18 contains the formal DIRT algorithm. The normal initialization procedure of inserting the start node is performed first. Then, one of the new portions of the DIRT algorithm is related to the saved node x_{save}. This addition allows for the

Algorithm 17: Update_DIR_Pruning(G, $x_{selected}$, x_{new})

Update_Dominance_Regions(G,$x_{selected}$,x_{new});
$X_d \leftarrow \{x \in G | \exists x', \texttt{DIR}(x) \subset \texttt{DIR}(x')\}$;
for $x \in X_d$ **do**
    $\mathbb{V}_{active} \leftarrow \mathbb{V}_{active} \setminus \{x\}$;
    $\texttt{NN.Remove}(x)$;
    $\mathbb{V}_{inactive} \leftarrow \mathbb{V}_{inactive} \cup \{x\}$;
    $x_{del} \leftarrow x$;
    while $\texttt{IsLeaf}(x_{del})$ **and** $x_{del} \in \mathbb{V}_{inactive}$ **do**
        $x_{next} \leftarrow \texttt{Parent}(x_{del})$;
        $\mathbb{V}_{inactive} \leftarrow \mathbb{V}_{inactive} \setminus \{x_{del}\}$;
        $\mathbb{E} \leftarrow \mathbb{E} \setminus \{\pi_{del}\}$;
        $x_{del} \leftarrow x_{next}$;
$G \leftarrow G \setminus X_d$;

Algorithm 18: DIRT($\mathbb{X}$, $\mathbb{U}$, $x_\circ$, $\mathbb{X}_G$, **H**, **BF**, T_{prop}, N)

$G = \{\mathbb{V}_{active} \leftarrow \{x_\circ\}, \mathbb{V}_{inactive} \leftarrow \emptyset, \mathbb{E} \leftarrow \emptyset\}$;
$\texttt{NN.Add}(x_\circ)$;
$x_{save} \leftarrow x_\circ$;
for N *iterations* **do**
    if $x_{save} \neq NULL$ **and** $\mathbf{H}(x_{save}) < \mathbf{H}(\texttt{Parent}(x_{save}))$ **then**
        $x_{selected} \leftarrow x_{save}$;
    else
        $x_{selected} \leftarrow$ DIR_Selection(G);
    if $E_{cand}(x_{selected}) = \emptyset$ **then** $E_{cand}(x_{selected}) \leftarrow \texttt{Blossom}(\ x_{selected}, \mathbb{U}, \mathbf{BF})$;
    while $E_{cand}(x_{selected}) \neq \emptyset$ **do**
        $\upsilon \leftarrow \arg\min_{\mathbf{H}(\pi(t))} E_{cand}(x_{selected})$;
        $E_{cand}(x_{selected}) \leftarrow E_{cand}(x_{selected}) \setminus \{\upsilon\}$;
        $\pi_{new} \leftarrow \int_0^t f(\pi(t), \upsilon(t))\, dt,\ where\ \pi(0) = x_{selected}$;
        if not $\texttt{Colliding}(\pi_{new})$ **then**
            $\mathbb{V}_{active} \leftarrow \mathbb{V}_{active} \cup \{\pi_{new}(t)\}$;
            $\mathbb{E} \leftarrow \mathbb{E} \cup \{\pi_{new}\}$;
            $\texttt{NN.Add}(\pi_{new}(t))$;
            Update_Dominance_Regions(G, $x_{selected}$, $\pi_{new}(t)$);
            $x_{save} \leftarrow \pi_{new}(t)$;
            if $\pi_{new}(t) \in G$ **and** $(\pi_{sol} = \emptyset$ **or** $\mathbf{cost}(x_{save}) < \mathbf{cost}(\pi_{sol}))$ **then**
                $\pi_{sol} \leftarrow \pi(x_\circ \rightarrow x_{save})$;
            break;
        else
            $x_{save} \leftarrow NULL$;
return π_{sol};

algorithm to be greedy when a node is first added. By selecting a newly added node that results in a smaller h value, in easy environments, in a small number of iterations, an initial solution can be found in just a few iterations. This process trusts the heuristic function, **H**, to judge when an edge is not making progress toward the goal, and reverts to the DIR_Selection process.

Algorithm 19: `Blossom`(x, $\mathbb{U}$, **BF**)

1 **if** x *expanded previously* **then** $t \leftarrow$ `Sample`$([0, T_{prop}])$;
2 $\upsilon \leftarrow$`Sample`(Υ);
3 **return** (υ,t) ;
4 **else return** $E_{cand} = \upsilon \in \Upsilon$ *s.t.* $|E_{cand}| =$ **BF** ;

The next new addition relates to the expansion process used here. When a node is selected, in an effort to maximize the likelihood that a useful edge will be added this iteration, a set of candidate edges are considered for each expansion step. This is similar to the Randomized `A`* algorithm, or `RRT-Blossom`[18, 38], where each node selection results in many edges added at once. The approach that `DIRT` takes is slightly different, in that only one edge can be added in an iteration. An unchecked edges are stored at that node and are examined when that node is selected again. The generation of this candidate edge set, is left as a user defined process, with a default as just a set of `MonteCarlo-Prop` calls. Finally, when the first edge set is exhausted, the algorithm reverts to a simple `MonteCarlo-Prop` procedure that only generates one edge. Algorithm 19 provides the `Blossom` procedure that switches between the edge generation process and the `MonteCarlo-Prop` process.

This means that each node can be considered to be in one of three propagation stages:

- If that node was generated along the negative gradient of the **H** function, it is prioritized.
- While there are user-defined maneuvers left to examine, try to add those edges.
- When all the options are exhausted, propagate random controls via `MonteCarlo-Prop`.

This progression of expansions when a node is selected over time reflects the priorities

of DIRT. The first priority is to find solutions quickly, then the priority becomes finding low cost trajectories.

6.3 Experimental Performance

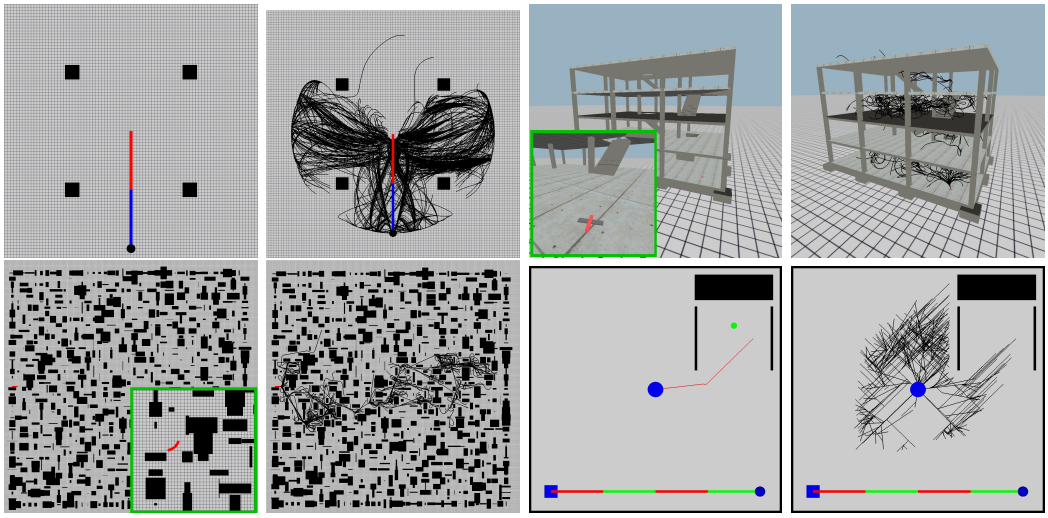

Figure 6.3: Test environments to evaluate DIRT: (Left to Right) 2-link acrobot, fixed-wing airplane in a building, car-trailer in a maze, 4-link planar manipulator pushing an object to the marked region. Results from [60].

The results here are originally from a previously published paper on DIRT[60]. The algorithms compared are Randomized A* [18], AO-RRT [28], and SST [59]. Randomized A* and SST have parameters that introduce pruning. SST has an additional parameter for a selection radius. The parameters were chosen to maximize the algorithm's effectiveness in each evaluation. Two versions of DIRT are included: i) the basic algorithm 18 and ii) a pruning version using Algorithm 17. This is denoted as DIRT-Prune. When Blossom generates a set of candidate edges, these edges are generated as equally spaced control values to span the control space of the system. Randomized A* only uses these generated candidate edges for expansion. These variants highlight the effectiveness of different modules of the proposed planner. Randomized A* uses a Blossom expansion strategy and shows the effects of not including randomized controls beyond greedy expansions. AO-RRT and SST have path quality guarantees but do not exhibit an informed nature. Methods that require an optimal local planner are not considered.

A set of scenarios involving dynamical systems are considered (Fig. 6.3):

a) a 2-link acrobot, which is passive-active [98]. The state space is 4 dims. The control is the torque for the second joint. The system is tasked with reaching an upright balancing position while avoiding obstacles, including self collisions. This system is low dimensional, but highly non-linear. The task space is just the 2 angles representing the point mass on the end of the links. The heuristic is the distance between the given state and the upright goal in task space.

b) A fixed-wing airplane moving through a building with tight stairwells to reach the top floor. The state space is 9 dimensions [81]. The task space is the x,y,z location of the fixed-wing airplane, and the heuristic is computed via L2 distances to waypoints in the stairwells.

c) A second order car-like vehicle with trailers and the task is moving among a maze of obstacles. The state space for this system is 8 dimensional, corresponding to the SE(2) configuration of the car, the steering angle, forward velocity, and three trailer angles, $\theta_1, \theta_2, \theta_3$, relative to the chassis attachment. The controls for this system are forward acceleration and steering angle velocity. The dynamics are:

$$\dot{x} = v\cos(\theta)\cos(\omega), \quad \dot{y} = v\sin(\theta)\cos(\omega),$$
$$\dot{\theta} = v\sin(\omega), \dot{v} = \tilde{a}.$$
$$\dot{\theta_1} = v\sin(\theta_1 - \theta), \dot{\theta_2} = v\sin(\theta_1 - \theta_2),$$
$$\dot{\theta_3} = v\sin(\theta_2 - \theta_3).$$

The task space is x, y location of the front vehicle and the heuristic is the L2 norm between the state and the goal.

d) A non-prehensile manipulation task, where an object must be pushed to a goal region. The object has inertia once pushed. The state space is 8 dimensional, corresponding to the 4 manipulator joint values and the object position and velocity in the 2D plane. The heuristic is the sum of a lower bounded time to reach the object with the end effector and a lower bounded time for the object to move at maximum velocity toward the goal, ignoring obstacles.

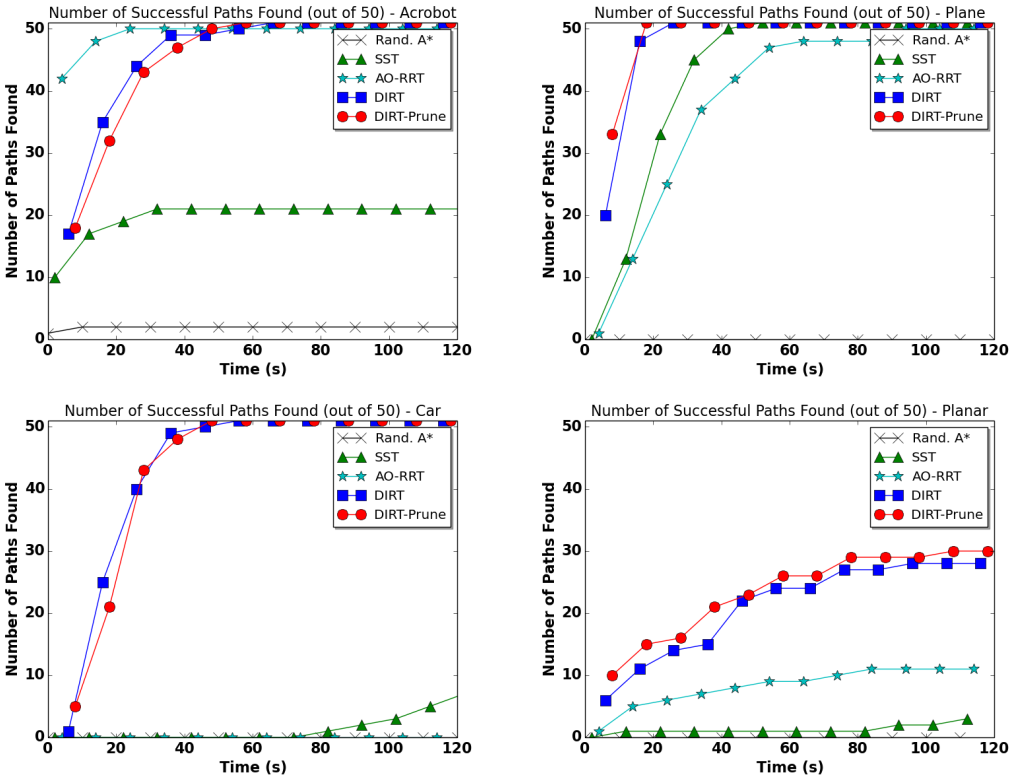

Figure 6.4: A sum of number of solutions found over time for each of the evaluated algorithm/system scenarios relating to DIRT. Each planner instance is ran 50 times to account for different random seeds. Results from [60].

Figures 6.4 and 6.5 (left) show the results on the 2-link acrobot. Many algorithms can find solutions quickly. Randomized A*, however, had difficulty finding solutions due to the pruning radius, which prohibits new nodes from being added in the highly non-linear space. SST gets around this by allowing new nodes to prune others only when path quality improves. SST does not find solutions for all instances within the time limit, unlike AO-RRT and DIRT. Both versions of DIRT find the best solutions and have lower variance.

Figures 6.4 and 6.5 (middle left) correspond to the fixed-wing airplane. Randomized A* has failed to find solutions and doesn't scale well. The DIRT methods effectively find solutions quickly with high solution quality. In these scenarios, there is only one possible corridor to the goal, which explains why both SST and DIRT find high quality solutions. DIRT finds them faster.

Figures 6.4 and 6.5 (middle right) are results for the car-trailer. This is the most

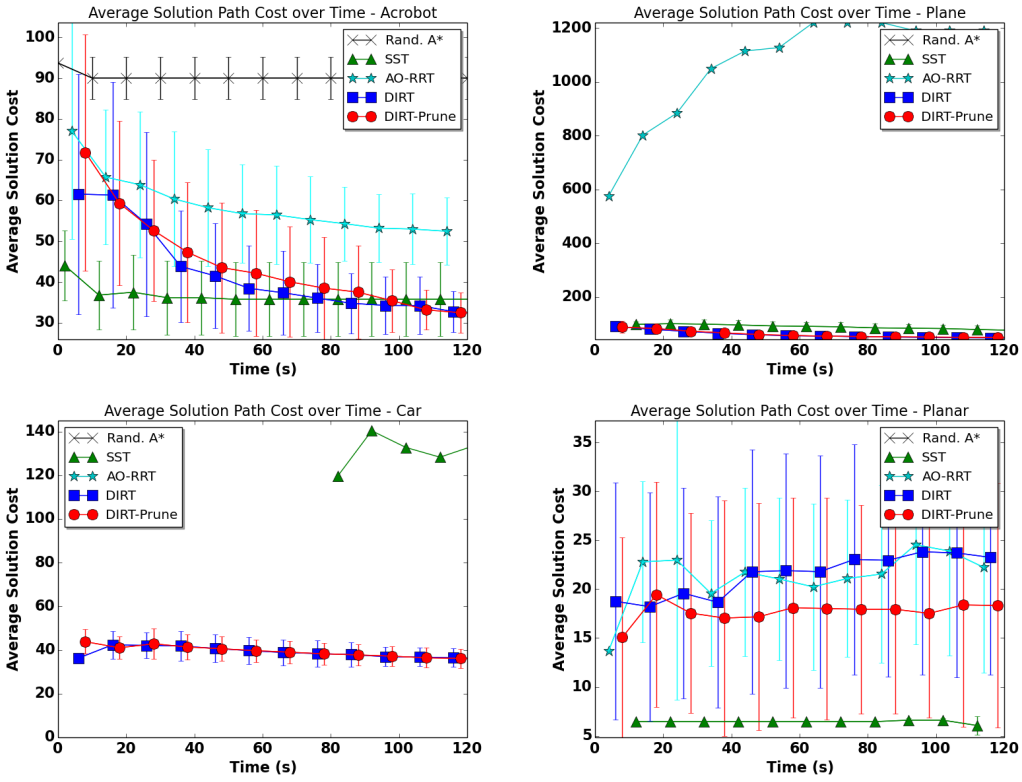

Figure 6.5: The average solution quality returned over time for each of the evaluated algorithm/system scenarios relating to DIRT. Each planner instance is ran 50 times to account for different random seeds (which results in some variance in solution quality). Results from [60].

difficult benchmark due to the large number of obstacles and large state space. Heuristic guidance is essential for finding solutions and DIRT is able to take advantage of it.

Figures 6.4 and 6.5 (right) provide results for the planar manipulator case. SST and Randomized A* were ineffective in this space. SST did find some solution trajectories, but the number of successes was small. The DIRT methods found the most solutions. Encoding a heuristic with more knowledge about how the manipulator approaches the object could help. In terms of path quality, when the algorithms find solutions, DIRT has slightly better solutions on average.

6.4 DIRT and Probabilistic Completeness

The proofs of probabilistic completeness and asymptotic near-optimality of DIRT follow the same Markov chain argument as SST. The propagation probability is the

same as previous, just requiring that a node is selected enough times to revert to the `MonteCarlo-Prop` procedure, $\gamma_{\texttt{DIR}-prop} = \gamma_{\delta}(\gamma_{\texttt{DIR}-select})^{\mathbf{BF}}$. If the selection probability is non-zero, this probability will also be non-zero.

There are two versions of the selection probability to consider here. First, the version of the dominance regions that are not pruned, is the simplest case. The probability that a node is selected is directly proportional to how large the dominance region. The dominance region for a node is always some distance to another node. So, its volume is always non-zero and the probability of selecting this node is also non-zero, i.e., $\gamma_{select} = \frac{\mu(\texttt{DIR}(x))}{\mu(\mathbb{X})*|X_{cand}|}$. It is likely that the true probability of node selection in the covering ball sequence is much higher, since multiple nodes may be in the same ball, and the dominance region for nodes in this ball may have larger volume larger than the covering ball. This probability also does not include cases where a random sample does not fall into an existing dominance region. In the case of the pruned dominance regions, when nodes are removed, that is only an event that occurs when another node's dominance region completely contains the other, so another node is likely in the covering ball when the first node is removed. In the worst case, the proof for probabilistic completeness can make use of the extended Markov chain used to prove `Sparse-RRT` was probabilistically complete.

Chapter 7

Planning in Challenging Domains

The DIRT motion planning algorithm is intended to be a motion planner with "plugin" capability for particular problem domains, with the aim that these plugged in features would be interpretable by the user. Instead of changing some pruning radius until good performance is achieved, instead the propagation length can be tuned, and good maneuvers can be generated before runtime. The heuristic function being defined in a task space is also helpful for interpretability. This chapter first mentions an application of the DIRT algorithmic framework in the context of manipulation, and then following that, a deep dive on applying DIRT to the tensegrity robot prototype, SUPERball, is presented.

7.1 Application Area: Manipulation

A large portion of this dissertation has been discussing the use of random controls in sampling-based motion planners. Sparse-RRT, SST, and DIRT are methods that can effectively make use of random controls. But, as expected, when a steering function is available, other motion planning methods may outperform these methods (see Figure 4.5 results for the two-dimensional point and three-dimensional rigid body). One domain that leverages these steering functions well is manipulation with linked arms.

In the context of manipulation tasks, one recent event showcasing state-of-the-art progress in this area is the Amazon Picking Challenge [15]. In preparation for this event, it is important to consider what type of local planning methods to use when building roadmaps and trees to perform pick and place tasks in the constrained environments for the challenge. In addition to this, the end-effector of the arm may be unsuited for certain objects. In previous work, motion planning methods using different steering

Figure 7.1: A Motoman SDA10F robot carrying the UniGripper tool and an Amazon-Kiva Pod stocked with objects.

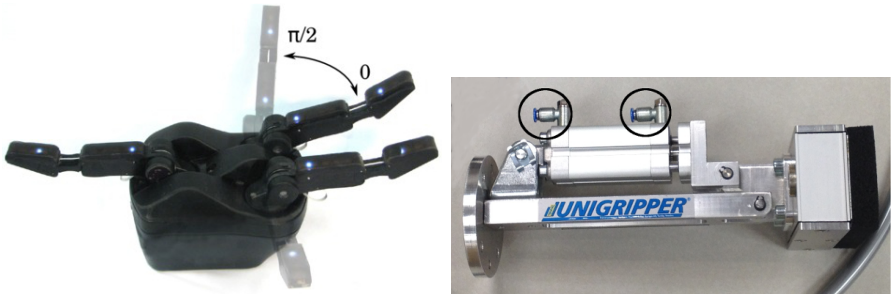

Figure 7.2: The end-effectors evaluated for the 2015 Amazon Picking Challenge. The RightHand Robotics "ReFlex" hand and its preshape motion. An end-effector using UniGripper's suction technology with a 1 wrist-like DOF.

methods were used to evaluate different end effector choices for the manipulation task of the Amazon Picking Challenge [68]. This is similar to how `SST` was used to evaluate the effectiveness of distance functions for belief space planning, where a motion planner is used as an impartial judge of some other factor.

The physical evaluation was performed with a Motoman SDA10F (Figure 7.1) and the two grippers evaluated were the "ReFlex" hand from RightHand Robotics and a custom designed vacuum gripper from UniGripper. The motion planning strategy is outlined as follows (Figure 7.3):

Transit Plan: Takes the arm from an initial state to a "connection" state. This plan is retrieved from a precomputed `PRM`*roadmap [40], which takes into account the arm and the static geometry. To find the shortest collision-free path given new objects in

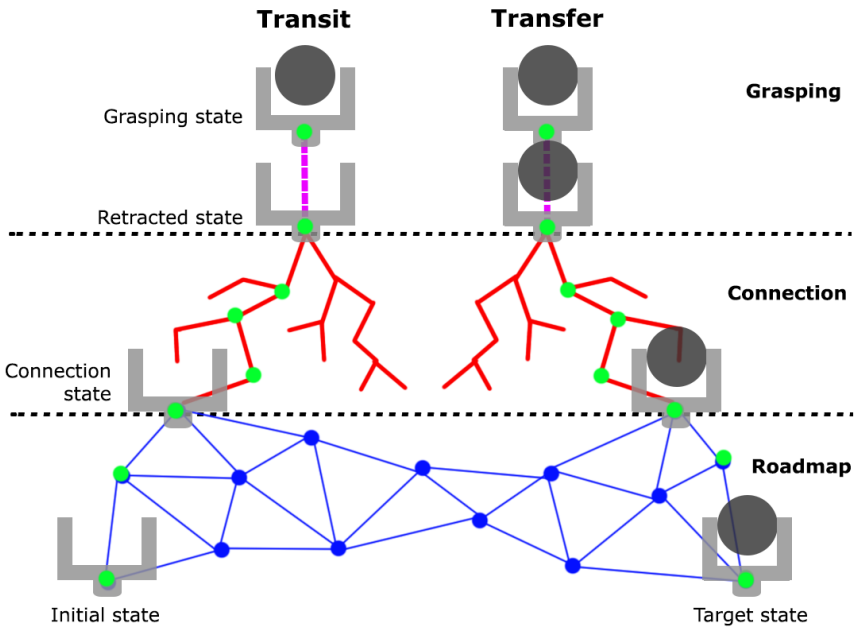

Figure 7.3: A visual breakdown of the two types of motion plans computed: (left) transit and (right) transfer plans in the end-effector evaluation process.

the world, an iterative lazy evaluation process is followed.

Transit Connection Plan: Takes the arm from the "connection" to the "retracted" state. This is computed online using a goal-biased `RRT`[57]. The start state is the "retracted" state and the candidate goals are the nearest roadmap nodes to the "retracted" state. The plan is executed in reverse to reach the shelf.

Reaching Plan: Takes the arm from the "retracted" to the "grasped" state. To accommodate for the time it takes for the *Unigripper* to establish suction with an object, the reaching portion of the plan was slowed down. A slight "pushing" action, to enable better contact between the suction sponge and the object surface.

Retracting Plan: Takes the arm from the "grasped" to the "retracted" state. It results in the arm moving a short distance away from the initial object pose.

Transfer Connection Plan: Takes the arm from the "retracted" state to a new "connection" state. This plan is computed online using a goal-biased `RRT`, similar to the transit connection, but with the object in hand.

Transfer Plan: Takes the arm from the new "connection" to the "target" state. This plan can be retrieved from a precomputed roadmap using lazy evaluation. The resulting

path is accepted only if it is collision-free given that the arm is holding the object.

The results of this study of end-effectors showed that the fingered end-effector is especially suited to non-uniform objects while the vacuum gripper is good for flat-surfaced objects. Getting back to the motion planning method itself, no care was taken to consider the path quality (disregarding that the PRM* method was used for motions outside the shelf). The motions into the shelf (the reaching and retracting portions) were not evaluated for path quality directly, but the DIRT method could have been used as a more effective tool to introduce improved path quality in these tasks.

7.1.1 The JIST Approach

The DIRT algorithm is the baseline of the method used for the Jacobian Informed Search Tree (JIST) [45]. This approach calculates a heuristic function for guiding a manipulator's end effector toward a target object in order to grasp that object. This heuristic is computed via a graph in the workspace of the end effector, and is built by considering a free-flying end effector and computing a roadmap for that. Then, when planning for the full manipulator the Jacobian can be used to try to follow those roadmap edges, using an approach similar to Blossom. A figure illustrating a core component of the method is in Figure 7.4, which describes the heuristic computation strategy that guides the motion planner.

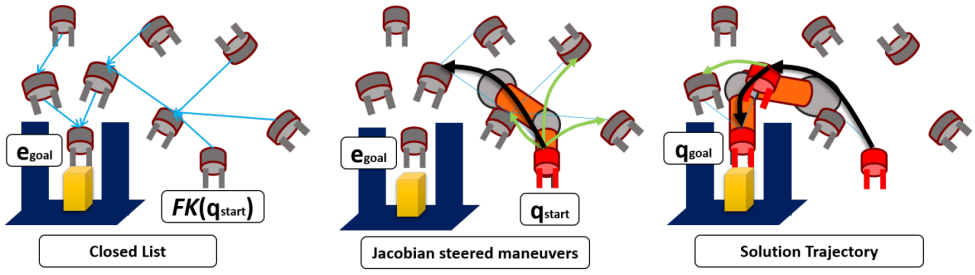

Figure 7.4: Highlight of the JIST algorithm for manipulation in clutter [45]. Using a graph computed with just the end-effector as a rigid body as a task-space heuristic, JIST is able to guide a tree-based planning strategy based on DIRT to grasp objects in tight areas with the full manipulator.

An excerpt of results from the original paper [45] are presented in Figure 7.5. These results show that the JIST approach is effective at both finding solutions in complex

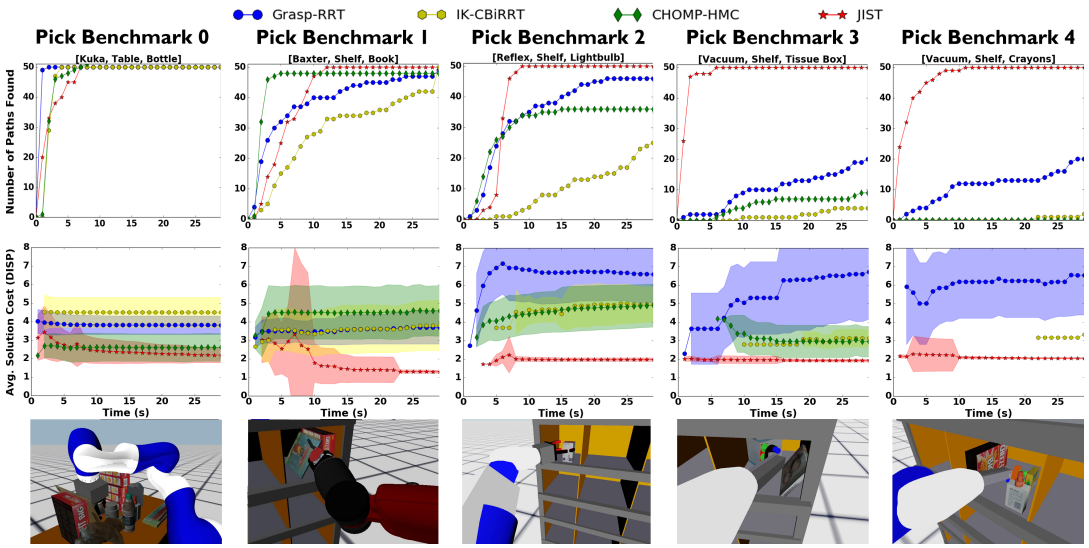

Figure 7.5: Success Rates, Solution Costs, and Example Picks for the Kuka+ReFlex (0), Baxter+Parallel (1), Motoman+ReFlex (2), Motoman+Vacuum (3,4)

scenarios, but also finding low cost solutions as well.

7.2 SUPERball: Next Generation Tensegrity Rover Prototype

SUPERball is a tensegrity-based robot prototype build by NASA Ames Research Center (seen in Figure 7.6). Tensegrity is a portmanteau of "tensional" and "integrity" and refers to structures that have isolated pieces that are suspended in compression by tensile elements. It is commonly seen in architecture, and recently has become a class of robot frameworks [29, 95]. There have been a wide variety of works that explore the capabilities of tensegrity-based robots for uneven terrain traversal, biological joint modeling, and pipe cleaning [82, 72, 6, 11, 10, 23].

SUPERball is aimed at being an exploration rover that allows for study of dynamic locomotion and path planning [8, 91]. This robot has a simulation framework built for it, based on the Bullet physics engine [16, 99] and has been verified against the real hardware prototype [73].

One of the goals of the SUPERball project is to explore the motion planning possibilities for this platform. Solving this problem is challenging since the state and control spaces for SUPERball are high dimensional and its dynamics are highly complex as

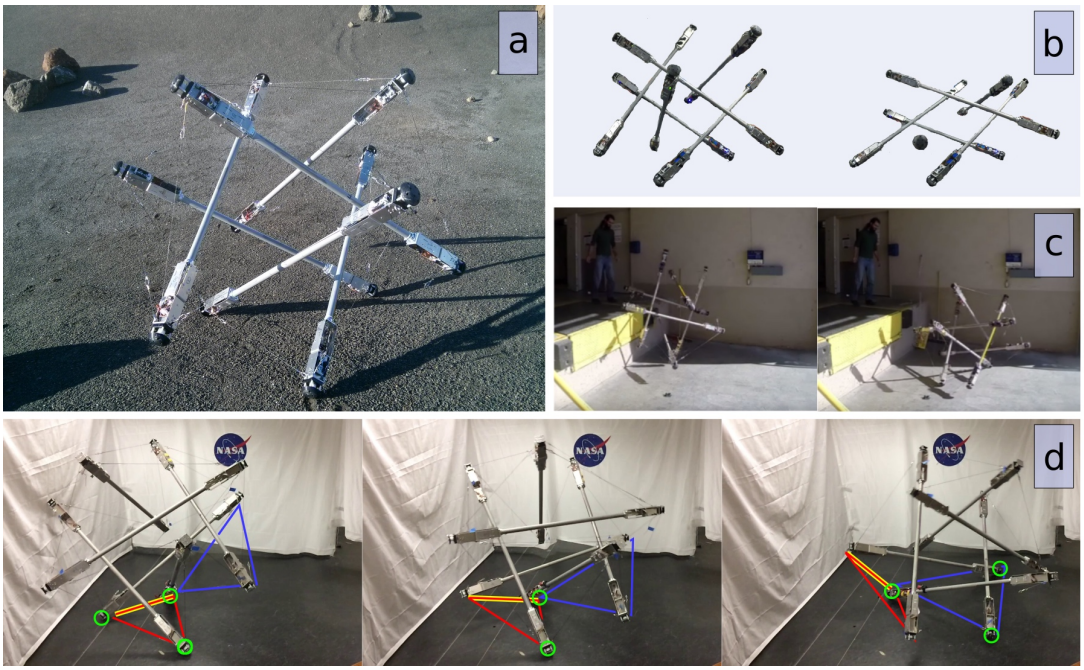

Figure 7.6: SUPERball tensegrity robot prototype: a) during a test at NASA Ames. b) active crouching by contracting cables (background and cables removed) c) deformation during drop-test d) rolling from face to face.

well (as evidenced by the need for a physics-based simulator). All of the cables impose forces on the structure, and there are many collisions from the ground that affect SUPERball's motion. Because of these dynamic effects, a forward simulator that a planner would use must make use of a physics-based simulator, such as the NTRT framework [99]. The use of such a simulator imposes significant runtime overhead in a motion planner, since each edge expansion requires a call to this simulator.

7.2.1 Previous Motion Planning Strategies

Some early planners provided deployment and shape changes through optimization to generate statically stable configurations [84, 105]. Further work accounted for self-collisions [30, 111]. Recent approaches plan paths for tensegrities but assume a control process slow enough to eliminate dynamics [86, 111]. This quasi-static approach is popular in civil engineering [89] but undesirable for mobile robots, which may need to visit unstable states and can furthermore benefit from energetic motion. Methods have been developed for utilizing dynamics locally, either along static equilibrium manifolds

[96] or through linear feedback [3] and Lyapunov-based controllers for 3D systems [110]. These approaches generally have not accounted for self-collisions or external contacts, limiting real-world applicability. A previous attempt at kinodynamic planning using `SST` was also explored [62], but was computationally prohibitive, due to heavy reliance on the physics-based simulator and many unsuccessful node additions due to pruning.

7.2.2 Motion Planning for SUPERball with `DIRT`: Setup

The mechanisms that `DIRT` provides for motion planning, i.e. the task space heuristic guidance, and edge generation procedures, are the core components of the following sections. A large portion is dedicated to the edge generation procedure, both for offline generation of motion primitives for SUPERball, and also an online generation strategy using the kinematic structure of SUPERball to generate dynamic motions [66, 67].

SUPERball's structure is comprised of 6 rigid bars and 24 cables that are actuated to spool and contract or unspool and release. All of the rigid bars cross the interior of the icosahedral outer structure, while the cables that actuate the structure are roughly on the exterior. The convex hull of the SUPERball consists of triangular faces of two types:

- $\Delta_i,\ i \in [0, 7]$: These eight faces have cables along all three edges and never share an edge with another Δ_i. Their neutral shape is equilateral.
- $\Lambda_i,\ i \in [8, 19]$, These 12 faces have cables on only two edges. They occur in pairs with a shared cable-free "virtual" edge. Their neutral shape is isosceles.

In its neutral configuration, SUPERball exhibits *pyritohedral symmetry*: there are 24 unique combinations of rotations and reflections that result in an equivalent configuration of the robot. This regularity is also expressed topologically: all nodes are connected to four cables, all Δ_i faces are bordered by three Λ_i faces with the same relative handedness, etc. Figure 7.7 presents a 2D illustration of the system topology in relation to its 3D expression.

Since SUPERball has this symmetry property, it is possible remap each of the 24 symmetrical configurations into a base configuration using a topological remapping [100]. This drastically reduces the effective search space for gaits, where the gaits relate

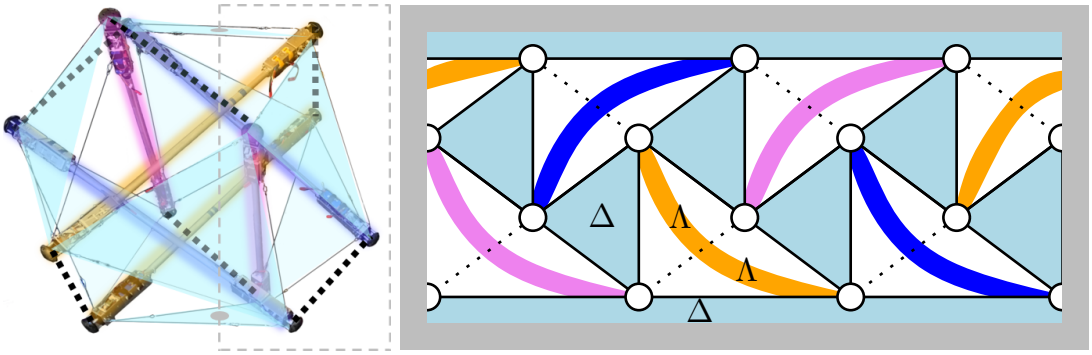

Figure 7.7: SUPERball topology flattened onto the plane after making a cut, shown as gray boundaries. Endcaps are represented by circles, actuators by black lines, virtual edges by dotted lines, and equilateral faces in light blue. Matching bars are parallel in physical space.

to what type of face is currently touching the ground, and what the target face type is. Since the surface geometry of SUPERball contains 24 edges occupied by cables and six additional virtual edges, 60 transitions between faces are possible when accounting for directionality, but after accounting for symmetry, only three are needed:

$$\widetilde{\Upsilon} = \{\tilde{v}_{\Lambda\Delta}, \tilde{v}_{\Delta\Lambda}, \tilde{v}_{\Lambda\Lambda}\} \tag{7.1}$$

Once these motion primitives have been computed, either offline or there is a controller that can perform these maneuvers online, then it is possible to plug those into the `Blossom` procedure of `DIRT`.

7.2.3 Gait Generation for SUPERball

The motion of SUPERball is inherently dynamic, due to nontrivial influences from gravity, friction, momentum, etc. Physically accurate simulation of the motion requires significant computational effort for propagation of a detailed dynamical model. For this reason, most of the past work has used simpler models to reason about the static stability of the vehicle and determine shape changes that result in gravity-induced locomotion [10, 11, 44].

The following setup is used to describe a small 4 dimensional but meaningfully varied input space, (relative to the full 24 dimensional control space), to efficiently

narrow down any search over possible actions.

In order to reduce the effective search space for good controls, the following kinematic reasoning is used. By simplifying the problem down to a manageable four dimensional space, the control process can be interpreted more readily than the 24-dimensional control space. The aim is not to maximize the fidelity or precision of geometric solutions, but rather to provide a mapping from a convenient and interpretable space onto a manifold of the much higher-dimensional control space that is useful for dynamic execution.

Consider the expression of the kinematic state of SUPERball as the position of all the endcaps of the rigid bars $\mathbf{q}_i$, i.e., the endpoints at which bars and cables are connected to each other. Collected within the vector $\mathbf{Q} \in \mathbb{R}^{36}$, these positions give the full geometric state. Ignoring the elasticity of the cables, the cable lengths are fully determined by the endcap positions. In addition, the center of mass of the system is also determined by the endcap positions, $\bar{\mathbf{q}} = \Sigma \mathbf{q}_i / 12$. The goal of this kinematics-based parametrization is to influence the movement of $\bar{\mathbf{q}}$ from being over its support triangle (the face of SUPERball on the ground) into its new target face. It is then assumed that this shift will induce dynamic motion to roll over to the next face.

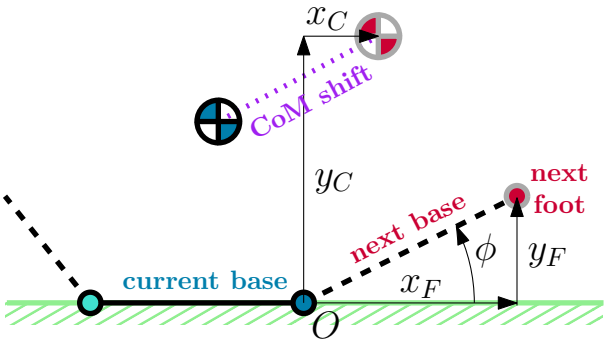

Figure 7.8: Main geometric parameters governing kinematic motion primitives, shown from a simplified side-on view.

Figure 7.8 illustrates the relevant values in a 2D frame. The vertical axis $\hat{\mathbf{y}}$ corresponds to the vertical direction in physical space, while the horizontal $\hat{\mathbf{x}}$ is the "forward" direction, such that $\hat{\mathbf{x}} \times \hat{\mathbf{y}}$ is parallel with the edge that is being pivoted over. Using an origin O located in the center of the pivot edge, static instability is achieved when the

x-component of $\bar{\mathbf{q}}$, x_C, becomes positive. In addition, a larger x_C produces more instability while larger y_C adds more potential energy, adding more momentum to the pivot. Given this kinematic description of SUPERball's geometry, an optimization procedure to get desired (x_C, y_C) and (x_F, y_F) points is needed.

The optimization procedure to move all the endcap configurations to achieve a desired center of mass shift falls into a coordinate descent framework. The full problem has 36 degrees of freedom. Achieving the desired $\bar{\mathbf{q}}$ and foot locations according to Figure 7.8 adds 6 constraints, and the bars of SUPERball constrain another 6 degrees of freedom, leaving 24 dimensions, the same dimensionality as the cables that are directly controllable. The costs being optimized are error functions penalizing deviation from the desired $\bar{\mathbf{q}}$, foot placement, and deviation from default cable lengths. The optimization must respect max cable length limits, maintaining the strict bar lengths, and achieving the $\bar{\mathbf{q}}$ position, while minimizing the error in foot placement and minimizing the change in cable lengths. All of these cost functions have a Jacobian matrix and provides gradient information for the optimization. Finally, the following four step process is repeated until convergence.

1. Apply a large step to reduce center-of-mass cost.
2. Apply a smaller step to improve foot positioning and minimize cable deformation (weighted by a factor w).
3. Ensure that cable bounds are approximately satisfied using one gradient step.
4. Ensure that bar lengths are exactly satisfied using multiple gradient steps.

Using this approach, basic kinematic behaviors can be achieved. But this optimization was performed in order to execute dynamic motions, so this optimization is evaluated in a dynamic setting. To that end, when computing the cable lengths that achieve the desired shape change, a time parameter is chosen (t_1) to reach the desired cable lengths, and then a second time (t_2) is chosen to maintain that shape in order to let dynamic effects occur. In all, this becomes 7 parameters of a kinematic control function:

$$\boldsymbol{\theta} = [x_C, y_C, x_F, y_F, w, t_1, t_2] \tag{7.2}$$

7.2.4 Gait Evaluation for SUPERball

Now that there is a procedure available to that can achieve a desired $\bar{\mathbf{q}}$ shift, the dynamic effects are evaluated as follows. Each chosen optimization parameter set from Equation 7.2 has several metrics considered about its resulting motion using the NTRT simulator. For each value of $\boldsymbol{\theta}$, 1000 seconds of using that control method is simulated in NTRT, resulting in several hundred control choices one after another. This evaluation tests how effective the parameter choices are from many different initial configurations. Each control choice from the several hundred in each evaluation is grouped by behaviors:

1. *None:* Remained on the original base face, essentially just shifting in place.
2. *Correct:* Ended on target base face, the result of careful motion.
3. *Multi:* Continued beyond target base, the result of additional momentum.
4. *Wrong:* Transitioned, but never reached target base, resulting in undesired motion.

If the target base triangle is not considered, the categories are simplified down to *None*, *Single*, and *Multi*. In addition, due to the geometric distinctions between the three transition types ($\{\tilde{v}_{\Lambda\Delta}, \tilde{v}_{\Delta\Lambda}, \tilde{v}_{\Lambda\Lambda}\}$), it is natural to use a different parameter vector $\boldsymbol{\theta}$ for each one. A standard set of values $\Theta = \{\boldsymbol{\theta}_{\Delta\Lambda}, \boldsymbol{\theta}_{\Lambda\Delta}, \boldsymbol{\theta}_{\Lambda\Lambda}\}$ is quickly established here in advance of exploring alternate characteristics of motion. The procedure for setting these standard values begins by measuring the x_F and y_F associated with the neutral configuration. Then x_C and y_C are set so that the relationship between the shifted center-of-mass and the next base is similar to that between the original center-of-mass and the current base, while still maintaining sufficient positive x_C for instability. Finally, the remaining parameters t_0, t_1, w are set via coarse manual search, directly

inspecting behavior that results from a sequence of several $\tilde{v}$ beginning from the neutral configuration and favoring *Correct* performance.

The standard parameter set Θ provides a basis from which to formulate a more systematic search of the parameter space, which is desired for revealing qualitative trade-offs. To maintain reasonable dimensionality, the search space is defined as $\delta\boldsymbol{\theta}$, such that the parameters of Figure 7.8 are altered identically for the two simpler transition types:

$$\Theta' = \{\boldsymbol{\theta}_{\Delta\Lambda} + \delta\boldsymbol{\theta}, \boldsymbol{\theta}_{\Lambda\Delta} + \delta\boldsymbol{\theta}, \boldsymbol{\theta}_{\Lambda\Lambda}\} \tag{7.3}$$
$$\delta\boldsymbol{\theta} = [\delta x_C, \delta y_C, \delta x_F, \delta y_F, 0, 0, 0]$$

The more difficult $\tilde{v}_{\Lambda\Lambda}$ transition is not altered due to its relative fragility.

The four-dimensional space of $\delta\boldsymbol{\theta}$ was explored using a grid search and evaluated using the categories from above. The result of this controller parameter search is a set of controllers with differing behaviors for SUPERball:

- The *Steady* controller Θ_S is intended for providing reliable, if not maximal, forward progress, and indeed does so on more than 90% of command cycles.
- The *Fast* controller Θ_F provides higher rates of multiple-transition outcomes, but at a high risk of causing movement in the wrong direction.
- The *Aggressive* controller Θ_A spends more time at low speed in order to recover from its more extreme deformations, which may be useful for ascending slopes or passing small obstacles with a brief surge of momentum.
- The *Big*-step controller Θ_B takes steps that are 15-25% larger than the other controllers on average, which may provide more options for aligning with narrow passages in the environment.

These four categories of gaits partition the behaviors of SUPERball. Figure 7.9 provides heatmaps for relevant parameters of the gait, such as movement speed, expected face transitions, and stride of foot. Additional in-depth displays of each chosen parameter's performance are in Figures 7.10 and 7.11.

Due to the symmetry of SUPERball, these same controller parameters can be used

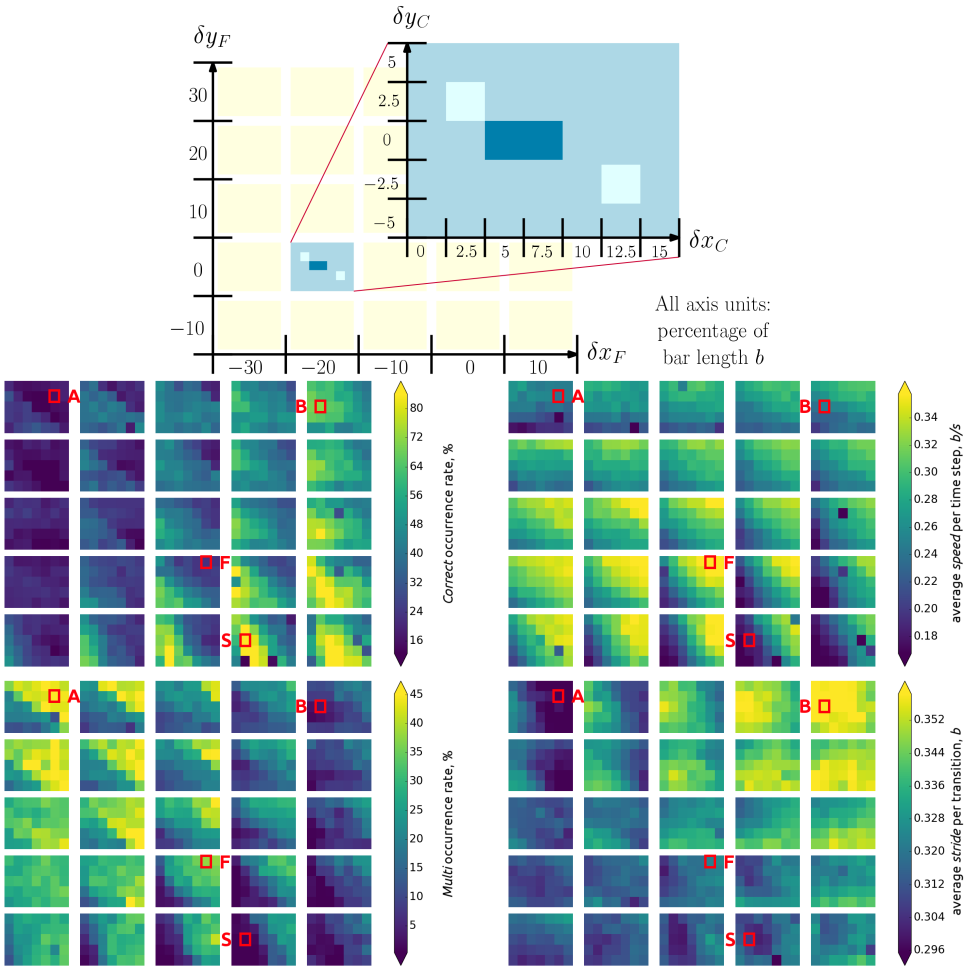

Figure 7.9: Each of four different controller evaluation metrics are plotted as a heat map on the four-dimensional space $\delta\boldsymbol{\theta}$, shown as nested 2D axes. Red boxes mark the selected controllers, Steady (S), Fast (F), Aggressive (A), and Big (B).

whichever of the 20 surface triangles of SUPERball are providing support. All of these different controllers end up becoming the candidate controls in the `Blossom` procedure of `DIRT`. The dominance regions are defined with respect to the $\bar{\mathbf{q}}$ in order to provide useful distance measures in the planning process.

7.2.5 Evaluation of SUPERball Gaits in `DIRT`

SUPERball was tasked with planning in several environments which aimed to test the effectiveness of the controllers, shown in Figure 7.12.

- **Easy:** An open, flat environment in which, the number of obstacles is minimal and a simple path to the goal is achievable.

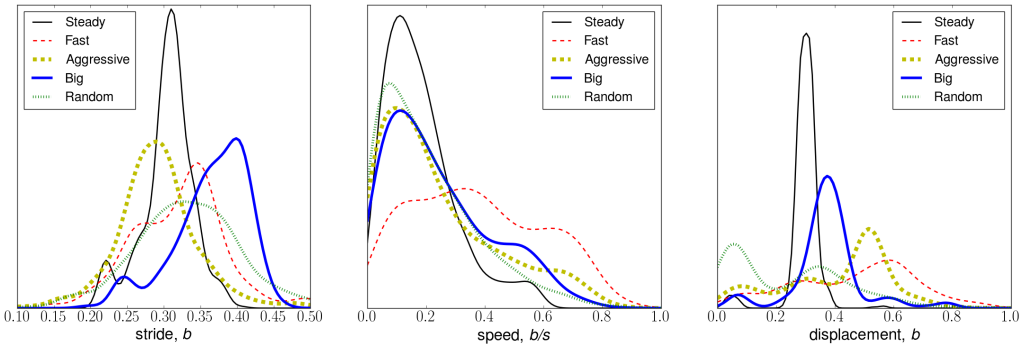

Figure 7.10: Distributions of key metric values for the four selected controllers and random controls.

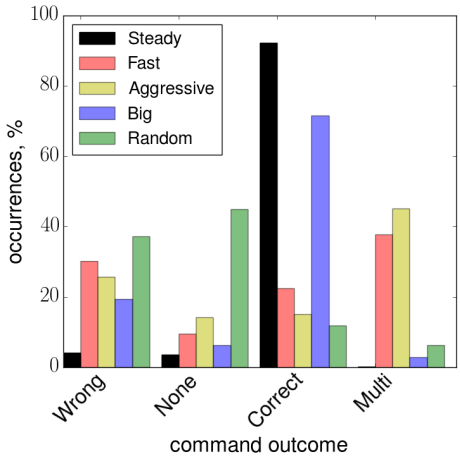

Figure 7.11: Occurrence rates of different command outcomes for the four selected controllers and random controls.

- **Scattered:** An environment with a number of small obstacles distributed throughout it. Collisions are not allowed between the robot and these obstacles, making high momentum maneuvers likely to be invalid.
- **Steps:** The robot can take a short path to the goal by first ascending a set of stairs and "limboing" underneath a bar. Alternately, it may follow a longer path on flat ground through a tight corridor.
- **Narrow:** The robot must pass through narrow obstacles and below a low ceiling. Collision with the ceiling is not permitted, making this environment very challenging.

Three different algorithms were evaluated in these environments. The first approach

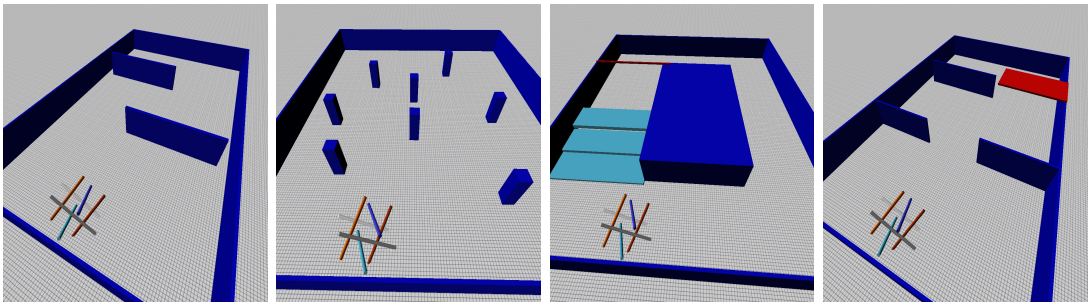

Figure 7.12: Environments used for evaluating planning for SUPERball: The goal is always located in the back left corner.

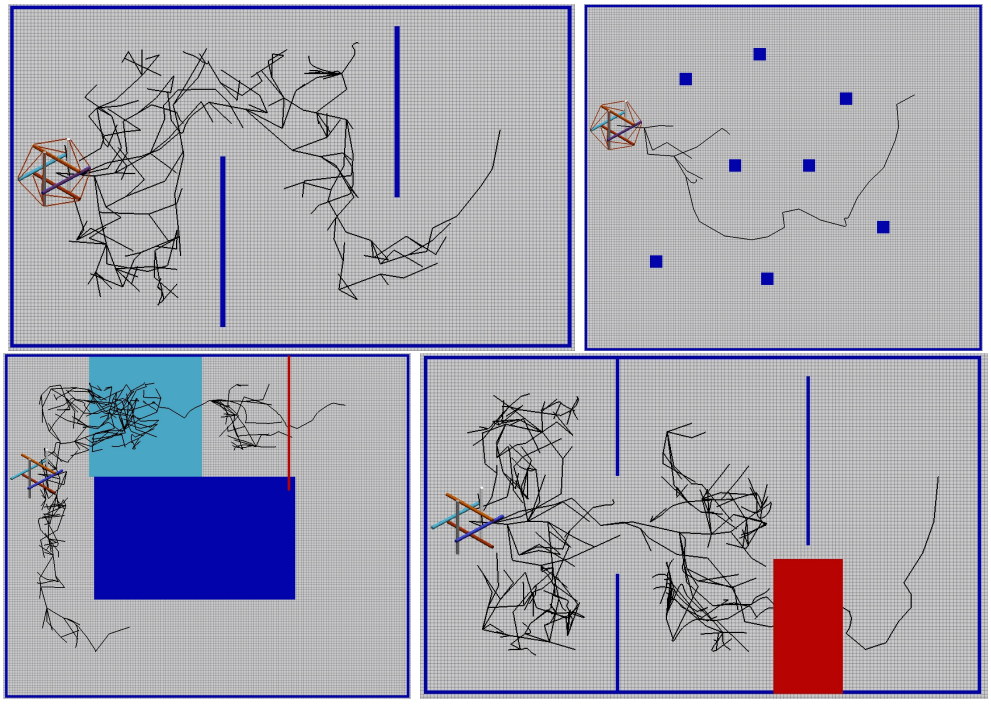

Figure 7.13: Example `DIRT` search trees in the "easy", "scattered", "steps", and "narrow passage" environments.

corresponds to the informed search process based on weighted A*, which employs the motion primitives as available controls. This method is used as an evaluation of the primitives themselves, since no randomly sampled controls are considered. Two different sets of motion primitives are evaluated:

- Offline-generated primitives using the Steady control parameterization, Θ_S.

$$\{\tilde{v}_{\Lambda\Delta}, \tilde{v}_{\Delta\Lambda}, \tilde{v}_{\Lambda\Lambda}\}_S$$

  This experiment closely resembles the gait experiments from [67].

- Online-generated primitives using all parameterizations, with a reset primitive added to the set.

$$\{\tilde{v}_{\Lambda\Delta}, \tilde{v}_{\Delta\Lambda}, \tilde{v}_{\Lambda\Lambda}\}_{S,F,A,B} \cup \{\tilde{v}^*\}$$

By considering each of these primitive sets, a gradual progression from low- momentum motion to more dynamic gaits is evaluated. Since this is a continuous space search, a small pruning radius is used (0.2m in these experiments). This mimics a "rewire" operation that can occur where a better path reaches the same node. This would never happen in continuous space, so this pruning mimics this behavior.

The second motion planning approach is an `RRT`, which utilizes the random controls. This comparison point illustrates the use of none of the motion primitives, and is considered a naive baseline that is not informed in any way.

The third approach is `DIRT`. Each of the motion primitive sets used for the weighted A* experiments is also used here. All algorithms are evaluated in terms of two criteria:

- Time elapsed until the first discovery of a valid solution
- Trajectory quality as a function of total search time

Trajectory quality is computed as the total duration of its physical execution within the simulation testbed.

These three representative planners were chosen to highlight the effectiveness of the primitive sets, while also emphasizing that the primitives cannot deal with every setup. The `RRT` method and the weighted A* are meant to be two extremes on a spectrum

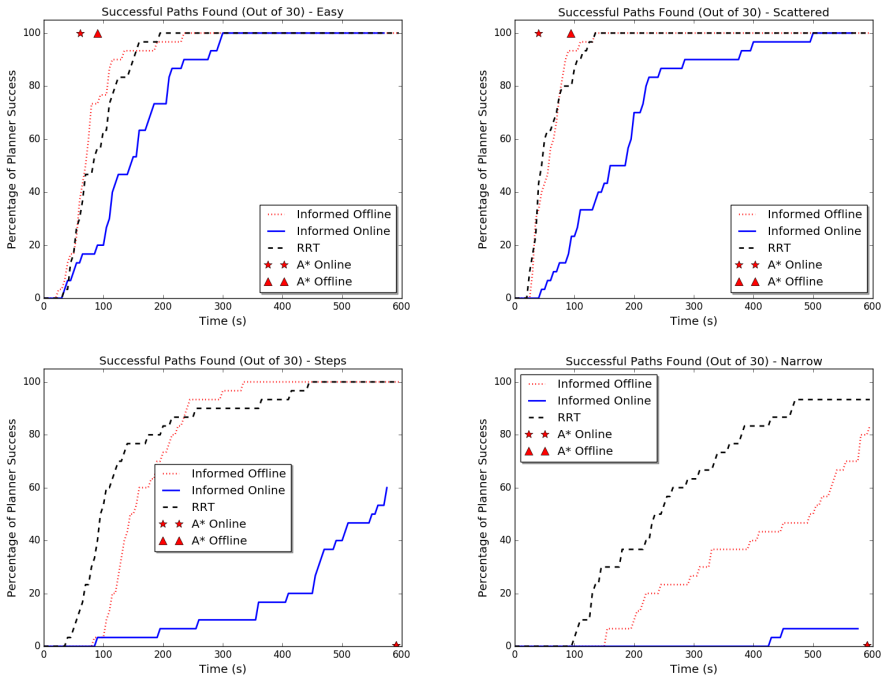

Figure 7.14: Number of solutions found over computation time for each algorithm in the "easy", "scattered", "steps", and "narrow passage" environments (from left to right). The sampling-based algorithms were executed 30 times each. A* planners do not find solutions in the last two environments.

of "random sampling of controls" vs "using a discrete, fixed set motion primitives", respectively. `DIRT` highlights a strategy that balances this tradeoff. It prioritizes the designed motion primitives and as a fallback when those primitives are not suitable to the environment, it uses random controls.

Figure 7.14 reports the methods' success ratio as a function of computation time. Figure 7.15 provides the average solution cost over time for the different environments. In the easy and scattered environments, A* is able to find solutions using only the gait controls. The more difficult environment experiments did not produce an A* result within the ten minute time limit. This can occur due to inability of the gaits to traverse the environments due to collisions, or due to hitting the time limit of ten minutes. Adding more diverse gaits to the gait library could potentially remedy this issue, but each additional gait parameterization imposes additional computational cost, leading to much longer planning times.

When using `DIRT`, the tradeoff of using a larger gait library is discovered. In all

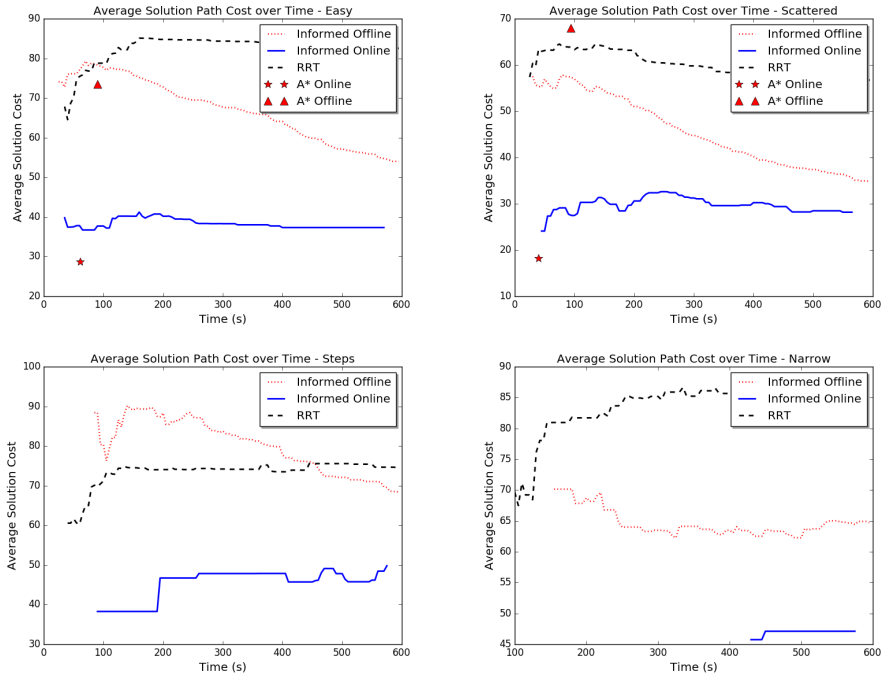

Figure 7.15: Average solution quality over computation time (with standard deviation) for the three methods in the "easy", "scattered", "steps", and "narrow passage" environments (from left to right). The sampling-based algorithms were executed 30 times each.

the environments, the use of many online gaits reduces the rate at which the planner can find solution trajectories. This is compared to the offline gaits, where in all but the narrow passage environment DIRT is competitive to the exploring RRT. The use of offline gaits allowed the planner to find initial solutions quickly, and then improve them over time. Example search trees can be found in Figure 7.13. As stated previously, the environments are configured so that most collisions are considered invalid, making the motion planning problem difficult in these scenarios. Random sampling is useful to overcome these challenges when gaits are unable to handle the environment.

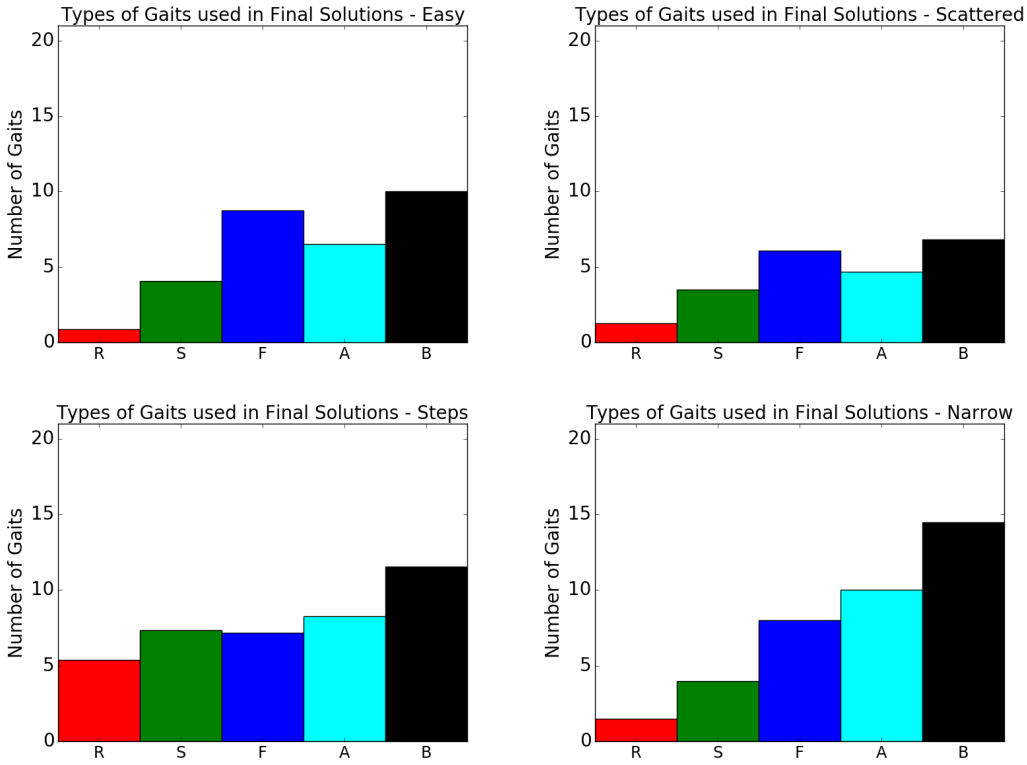

Figure 7.16: Gait composition for successful trajectories computed by DIRT.

When a solution is discovered using the online-evaluated gait library, however, the quality of those solutions is significantly better than all other algorithms except for A* equipped with the same gaits. DIRT using online gaits can also overcome A*'s drawback and find solutions in the steps and narrow passage environments, though with an overhead of having to exhaust all of the gait options before trying out random controls. To examine the gait composition of solutions when using DIRT, see Figure 7.16.

In the given time budget, solutions are still mostly comprised of gaits and not random controls. The random controls that do show up are usually necessary to overcome some environmental difficulty, like in the narrow passage environment and its low ceiling. `DIRT` makes use of all the controller types (Θ_S,Θ_F,Θ_A,Θ_B) to make progress in each environment, but revert back to random controls after exhausting all controller options at a node. Especially in the steps environment, more random controls are needed to climb the stairs. This is expected due to the controller assumption of flat ground. A different controller generation strategy would be needed to more effectively target this terrain (e.g. [101]).

The algorithmic framework provided by `DIRT` is heavily influenced by the user generated primitives provided to the algorithm. This is the main difference between the previous attempt at planning for SUPERball and this approach [62, 66].

Chapter 8

Conclusions

Sampling-based algorithms are an effective class of methods that plan motions for kinodynamic robots. The general steps of these methods are *node selection* and *node expansion*. The canonical kinematic `RRT` method makes use of a Voronoi selection strategy and a *steering* node expansion strategy. However, the kinodynamic `RRT`(`RRT-ForwardProp`) changes the expansion strategy into a *forward propagation* strategy that randomly samples control vectors and durations to propagate those controls with the system dynamics. This is an especially useful strategy when the system dynamics are complex or modeled computationally, such as with a physics engine.

As sampling-based algorithms matured, more focus on path quality was introduced, but this same focus was non-existant for methods that relied on the forward propagation expansion strategy. Chapter 3 discussed the `Sparse-RRT` method, which modified `RRT-ForwardProp` in two major ways. First, the selection procedure was changed to incorporate path costs via the `BestNear` selection primitive. This modification allows for some bias in selecting nodes depending on how good the path from the root node to the selected node is. Secondly, `Sparse-RRT` introduces a pruning primitive after the node expansion step of the tree search. This local pruning operation allows for faster runtime performance by removing suboptimal nodes that are unlikely to provide high quality solution paths to the goal. For the first time, `Sparse-RRT` has been proven to be probabilistically complete, albeit with a subpar convergence rate.

Chapter 4 discusses a small variation on the `Sparse-RRT` method, called `Stable Sparse-RRT`(`SST`). Instead of limiting the pruning regions to be centered around tree nodes, `SST` allows for the pruning regions to be independent of the tree, and instead be "witnesses" to their regions. This extra stability in the pruning operation allows for not

only probabilistic completeness, but improved convergence rates for finding solutions. The solutions that SST finds are also shown to be asymptotically near-optimal.

As an application area of SST, choices of distances functions for planning in belief space are evaluated in Chapter 5. Since SST only requires the "forward propagation" local planner, SST can be used to plan in belief space effectively, given a proper distance function choice. After evaluating many alternatives, the Wasserstein, or Earth Movers, distance was shown to be an effective tool for planning under uncertainty.

Even though SST is an effective algorithm to use, it still has its drawbacks. SST can struggle at finding initial solutions if lots of pruning occurs during the planner's first set of iterations. Furthermore, SST makes use of parameters that can opaque to novice implementers and users. Incorporating task-space heuristics into the Sparse-RRT/SST methods can alleviate these issues. For methods that use the forward propagation expansion strategy, task-space heuristics are an avenue for improving this expansion strategy to provide a middle ground between forward propagation and steering. DIRT is an example of a method that incorporates this heuristic guidance, and was detailed in Chapter 6.

The DIRT method is a general framework for planning for systems and problems were "forward propagation" is the preferred or only local planner choice, and exposes interpretable parameters to this planning problem, namely good edge generation strategies, propagation length, and task space guidance. By exposing all of these mechanisms to the user, each can be examined and tailored to the specific planning problems. This is highlighted with the case study of SUPERball and tailoring the expansion strategy using local controllers, and briefly mentioned in the context of manipulation in Chapter 7.

8.1 Open Issues and Future Avenues for Exploration

The introduction of the Sparse-RRT/SST/DIRT family of algorithms provides much needed alternative methods for sampling-based planning in domains where a steering function is unavailable. This was shown to be effective in areas such as manipulation,

locomotion, and general motion planning. Of note, the DIRT method relies heavily on a good edge generation strategy. If the Blossom procedure is not provided with a good set of candidate edges, DIRT's performance will suffer. By spending effort on this edge generation procedure, improvements can be made for different problem domains. Using machine learning methods to learn good edge generation strategies is a promising avenue for study, and even has some preliminary results to support the idea [94].

This family of algorithms has been shown to perform well in a variety of planning domains and problems. In some cases, the performance of these motion planners is in spite of, and not because of, the analysis presented. In the case of Sparse-RRT and DIRT with the pruning extension, convergence rate results are unideal. In addition, the asymptotic near-optimality bound is weak (a multiplicative bound). Revised analysis techniques or algorithmic changes to enable different analysis strategies will possibly be able to bridge this gap between performance in practice and performance in theory, and some work in this area has begun [46] and extended in upcoming work.

One promising algorithmic paradigm is the AO-X framework [28]. This framework acts as a meta-planning paradigm that makes use of other motion planning methods to improve their solutions over time. Exploring the combination of this with the Sparse-RRT/SST/DIRT family of algorithms may yield interesting insights into analysis and possibly provide improved practical performance as well.

Finally, one of the goals with the DIRT method was to maximize the effectiveness of each edge generated during runtime, even going so far as to perform work offline to generate good edge generation strategies, as was the case with SUPERball. Even in this case, however, due to the use of a physics engine, iterations of DIRT were still coupled to the efficiency of that physics engine. Another future direction of research could examine this runtime dependency and explore strategies for further minimizing the calls to a physics engine, possibly through multi-tiered planning strategies using simplified robot dynamics.

Overall, this dissertation focused on compensating for a sampling-based planner's weaknesses by modifying other mechanisms of the algorithm. Because of RRT's simplicity, it is also simple to try many variants, and most of these variants provide some

practical benefit for some application. By examining some previous variants of RRT, provable benefits are possible by reintroducing such a variant to the research community. This helps create more general planning algorithms, instead of ones that focus on a particular application. This kind of reexamination is helpful, not only to make better algorithms, but to understand their behavior more effectively.

References

[1] Adiyatov, O., Sultanov, K., Zhumabek, O., and Varol, H. A. (2017). Sparse tree heuristics for rrt* family motion planners. In *2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1447–1452.

[2] Ai-Omari, M. A. R., Jaradat, M. A., and Jarrah, M. (2013). Integrated Simulation Platform for Indoor Quadrotor Applications. In *Mechatronics and its Applications (ISMA), 2013 9th International Symposium on.*

[3] Aldrich, J. B., Skelton, R. E., and Delgado, K. K. (2003). Control Synthesis for Light and Agile Robotic Tensegrity Structures. In *American Control Conference (ACC)*, volume 6, pages 5245–5251.

[4] Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. Y. (1998). An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions. *Journal of the ACM*, 45(6):891–923.

[5] Bai, H., Hsu, D., Kochenderfer, M., and Lee, W. (2012). Unmanned aircraft collision avoidance using continuous-state POMDPs.

[6] Bliss, T., Iwasaki, T., and Bart-Smith, H. (2013). Central Pattern Generator Control of a Tensegrity Swimmer. *IEEE/ASME Transactions on Mechatronics*, 18(2):586–597.

[7] Brin, S. (1995). Near neighbor search in large metric spaces.

[8] Bruce, J., Caluwaerts, K., Iscen, A., Sabelhaus, A. P., and SunSpiral, V. (2014). Design and evolution of a modular tensegrity robot platform. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3483–3489.

[9] Bry, A. and Roy, N. (2011). Rapidly-exploring Random Belief Trees for Motion Planning Under Uncertainty. In *ICRA*.

[10] Caluwaerts, K., Despraz, J., Işçen, A., Sabelhaus, A. P., Bruce, J., Schrauwen, B., and SunSpiral, V. (2014). Design and control of compliant tensegrity robots through simulation and hardware validation. *Journal of the Royal Society Interface*, 11(98):20140520.

[11] Chen, L. H., Cera, B., Zhu, E. L., Edmunds, R., Rice, F., Bronars, A., Tang, E., Malekshahi, S. R., Romero, O., Agogino, A. K., and Agogino, A. M. (2017). Inclined surface locomotion strategies for spherical tensegrity robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4976–4981.

[12] Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S. (2005). *Principles of Robot Motion*. The MIT Press.

[13] Choudhury, S., Gammell, J. D., Barfoot, T. D., Srinivasa, S. S., and Scherer, S. (2016). Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4207–4214. IEEE.

[14] Chow, W. (1940/1941). Über Systeme von linearen partiellen Differentialgleichungen erster Ordnung. *Math. Ann.*, 117:98–105.

[15] Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M., and Wurman, P. R. (2016). Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1):172–188.

[16] Coumans, E. (2012). Bullet Physics Engine. http://bulletphysics.org.

[17] Denny, J., Morales, M. M., Rodriguez, S., and Amato, N. M. (2013). Adapting RRT Growth for Heterogeneous Environments. In *IROS*, Tokyo, Japan.

[18] Diankov, R. and Kuffner, J. (2007). Randomized statistical path planning. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1–6. IEEE.

[19] Dobson, A. and Bekris, K. (2014). Sparse Roadmap Spanners for Asymptotically Near-Optimal Motion Planning. *International Journal of Robotics Research (IJRR)*, 33(1):18–47.

[20] Donald, B. R., Xavier, P. G., Canny, J., and Reif, J. (1993). Kinodynamic Motion Planning. *Journal of the ACM*, 40(5):1048–1066.

[21] Ferguson, D. and Stentz, A. (2006). Anytime RRTs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5369–5375.

[22] Fraichard, T. and Asama, H. (2004). Inevitable collision statesa step towards safer robots? *Advanced Robotics*, 18(10):1001–1024.

[23] Friesen, J., Pogue, A., Bewley, T., de Oliveira, M. C., Skelton, R. E., and Sunspiral, V. (2014). DuCTT: A tensegrity robot for exploring duct systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4222–4228.

[24] Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D. (2014). Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. *arXiv preprint arXiv:1404.2334*.

[25] Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D. (2015). Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3067–3074. IEEE.

[26] Geraerts, R. and Overmars, M. H. (2007). Reachability-based Analysis for Probabilistic Roadmap Planners. *Journal of Robotics and Autonomous Systems (RAS)*, 55:824–836.

[27] Grinstead, C. and Snell, J. (2012). *Introduction to Probability*. American Mathmatical Society, Providence, RI.

[28] Hauser, K. and Zhou, Y. (2016). Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space. *IEEE Transactions on Robotics*, 32(6):1431–1443.

[29] Heartney, E. (2009). *Kenneth Snelson: Forces Made Visible*. Hudson Hills, 2009 edition.

[30] Hernández Juan, S., Skelton, R. E., and Mirats Tur, J. M. (2009). Dynamically stable collision avoidance for tensegrity based robots. In *ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots*, pages 315–322.

[31] Horowitz, M. and Burdick, J. (2013). Interactive Non-Prehensile Manipulation for Grasping Via POMDPs.

[32] Hsu, D., Jiang, T., Reif, J., and Sun, Z. (2003). The Bridge Test for Sampling Narrow Passages with Probabistic Roadmap Planners. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 4420–4426, Taipei, Taiwan.

[33] Hsu, D., Kindel, R., Latombe, J. C., and Rock, S. (2002). Randomized Kinodynamic Motion Planning with Moving Obstacles. *International Journal of Robotics Research (IJRR)*, 21(3):233–255.

[34] Hsu, D., Latombe, J., and Motwani, R. (1997). Path Planning in Expansive Configuration Spaces. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 3, pages 2719–2726, Albuquerque, NM.

[35] Islam, F., Nasir, J., Malik, U., Ayaz, Y., and Hasan, O. (2012). RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution. In *ICMA*.

[36] Jeon, J.-H., Cowlagi, R., Peters, S., Karaman, S., Frazzoli, E., Tsiotras, P., and Iagnemma, K. (2013). Optimal Motion Planning with the Half-Car Dynamical Model for Autonomous High-Speed Driving. In *American Control Conference (ACC)*.

[37] Jeon, J.-H., Karaman, S., and Frazzoli, E. (2011). Anytime Computation of Time-Optimal Off-Road Vehicle Maneuvers using the RRT*. In *IEEE Conference on Decision and Control (CDC)*, pages 3276–3282.

[38] Kalisiak, M. and van de Panne, M. (2006). Rrt-blossom: Rrt with a local flood-fill behavior. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1237–1242. IEEE.

[39] Kallman, M. and Mataric, M. (2004). Motion Planning Using Dynamic Roadmaps. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 5, pages 4399–4404, New Orleands, LA.

[40] Karaman, S. and Frazzoli, E. (2011). Sampling-based Algorithms for Optimal Motion Planning. *International Journal of Robotics Research (IJRR)*, 30(7):846–894.

[41] Karaman, S. and Frazzoli, E. (2013). Sampling-Based Optimal Motion Planning for Non-holonomic Dynamical Systems. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

[42] Kavraki, L. E., Kolountzakis, M. N., and Latombe, J.-C. (1998). Analysis of Probabilistic Roadmaps for Path Planning. *IEEE Transactions on Robotics and Automation (TRA)*, 14(1):166–171.

[43] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. (1996). Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation (TRA)*, 12(4):566–580.

[44] Kim, K., Agogino, A. K., Toghyan, A., Moon, D., Taneja, L., and Agogino, A. M. (2015). Robust learning of tensegrity robot control for locomotion through form-finding. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5824–5831.

[45] Kimmel, A., Shome, R., Littlefield, Z., and Bekris, K. (2018). Fast, anytime motion planning for prehensile manipulation in clutter. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–9. IEEE.

[46] Kleinbort, M., Solovey, K., Littlefield, Z., Bekris, K. E., and Halperin, D. (2018).

Probabilistic completeness of rrt for geometric and kinodynamic planning with forward propagation. *IEEE Robotics and Automation Letters*, 4(2):x–xvi.

[47] Kneebone, M. and Dearden, R. (2009). Navigation Planning in Probabilistic Roadmaps with Uncertainty.

[48] Koval, M. C., Pollard, N. S., and Srinivasa, S. (2014). Pre- and Post-Contact Policy Decomposition for Planar Contact Manipulation Under Uncertainty. In *RSS*, Berkeley, USA.

[49] Kuffner, J. and Lavalle, S. (2000). An efficient approach to single-query path planning. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

[50] Kuffner, J. and LaValle, S. (2005). An efficient approach to path planning using balanced bidirectional rrt search. Technical report, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[51] Kunz, T. and Stilman, M. (2014). Kinodynamic RRTs with Fixed Time Step and Best-Input Extension Are Not Probabilistically Complete. In *Workshop on Algorithmic Foundations of Robotics (WAFR)*.

[52] Kurniawati, H., Du, Y., Hsu, D., and Lee, W. (2011). Motion Planning under Uncertainty for Robotic Tasks with Long Time Horizons. *IJRR*, 30(3):308–323.

[53] Kurniawati, H., Hsu, D., and Lee, W. (2008). SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *RSS*.

[54] Kurniawati, H. and Patrikalakis, N. M. (2012). Point-Based Policy Transformation: Adapting Policy to Changing POMDP Models. In *WAFR*.

[55] Lan, X., Vasile, C., and Schwager, M. (2015). Planning Persistent Monitoring Trajectories in Spatiotemporal Fields using Kinodynamic RRT*. In *ICRA*.

[56] Larsen, E., Gottschalk, S., Lin, M. C., and Manocha, D. (1999). Fast proximity queries with swept sphere volumes. Technical report, Technical Report TR99-018, Department of Computer Science, University of North Carolina.

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

[58] Li, Y., Littlefield, Z., and Bekris, K. E. (2014). Sparse Methods For Efficient Asymptotically Optimal Kinodynamic Planning. In *Workshop on Algorithmic Foundations of Robotics (WAFR)*, Istanbul, Turkey.

[59] Li, Y., Littlefield, Z., and Bekris, K. E. (2016). Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564.

[60] Littlefield, Z. and Bekris, K. E. (2018a). Efficient and Asymptotically Optimal Kinodynamic Motion Planning via Dominance-Informed Regions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9.

[61] Littlefield, Z. and Bekris, K. E. (2018b). Informed Asymptotically Near-Optimal Planning for Field Robots with Dynamics. In *Conference on Field and Service Robotics (FSR)*, pages 449–463.

[62] Littlefield, Z., Caluwaerts, K., Bruce, J., SunSpiral, V., and Bekris, K. E. (2016a). Integrating simulated tensegrity models with efficient motion planning for planetary navigation. In *International Symposium on AI, Robotics and Automation in Space (i-SAIRAS)*.

[63] Littlefield, Z., Krontiris, A., Kimmel, A., Dobson, A., Shome, R., and Bekris, K. E. (2014). An Extensible Software Architecture For Composing Motion And Task Planners. In *In International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, Bergamo, Italy.

[64] Littlefield, Z., Kurniawati, H., and Bekris, K. E. Klimenko, D. (2015). The Importance Of A Suitable Distance Function In Belief-Space Planning. In *International Symposium on Robotic Research (ISRR)*, Sestri Levante, Italy.

[65] Littlefield, Z., Li, Y., and Bekris, K. (2013). Efficient Sampling-based Motion Planning with Asymptotic Near-Optimality Guarantees with Dynamics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[66] Littlefield, Z., Surovik, D., Vespignani, M., Bruce, J., Wang, W., and Bekris, K. E. (2019). Kinodynamic planning for spherical tensegrity locomotion with effective gait primitives. *The International Journal of Robotics Research*, page 0278364919847763.

[67] Littlefield, Z., Surovik, D., Wang, W., and Bekris, K. E. (2017). From Quasi-static to Kinodynamic Planning for Spherical Tensegrity Locomotion. In *International Symposium on Robotics Research (ISRR)*.

[68] Littlefield, Z., Zhu, S., Kourtev, H., Psarakis, Z., Shome, R., Kimmel, A., Dobson, A., De Souza, A. F., and Bekris, K. E. (2016b). Evaluating end-effector modalities for warehouse picking: A vacuum gripper vs a 3-finger underactuated hand. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1190–1195. IEEE.

[69] Lozano-Pérez, T. and Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570.

[70] Marble, J. D. and Bekris, K. (2013). Asymptotically Near-Optimal Planning With Probabilistic Roadmap Spanners.

[71] Melchior, N. and Simmons, R. (2007). Particle RRT for Path Planning with Uncertainty. In *ICRA*.

[72] Mirletz, B., Bhandal, P., Adams, R. D., Agogino, A. K., Quinn, R. D., and Sun-Spiral, V. (2015a). Goal directed CPG based control for high DOF tensegrity spines traversing irregular terrain. *Soft Robotics*, 2(4):165–176.

[73] Mirletz, B. T., Park, I. W., Quinn, R. D., and SunSpiral, V. (2015b). Towards bridging the reality gap between tensegrity simulation and robotic hardware. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5357–5363.

[74] Muja, M. and Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2.

[75] Murray, S., Floyd-Jones, W., Qi, Y., Sorin, D. J., and Konidaris, G. (2016). Robot motion planning on a chip. In *Robotics: Science and Systems*.

[76] Paden, B. and Frazzoli, E. (2016). A Generalized Label Correcting Method for Optimal Kinodynamic Motion Planning . In *WAFR*.

[77] Pan, J., Chitta, S., and Manocha, D. (2011). Probabilistic Collision Detection between Noisy Point Clouds using Robust Classification. In *ISRR*.

[78] Pan, J., Chitta, S., and Manocha, D. (2012). Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866. IEEE.

[79] Pan, J. and Manocha, D. (2012). Gpu-based parallel collision detection for fast motion planning. *The International Journal of Robotics Research*, 31(2):187–200.

[80] Papadopoulos, G., Kurniawati, H., and Patrikalakis, N. (2014). Analysis of Asymptotically Optimal Sampling-based Motion Planning Algorithms for Lipschitz Continuous Dynamical Systems. http://arxiv.org/abs/1405.2872.

[81] Paranjape, A., Meier, K., Shi, X., Chung, S.-J., and Hutchinson, S. (2013). Motion primitives and 3-D path planning for fast flight through a forest. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[82] Paul, C., Valero-Cuevas, F. J., and Lipson, H. (2006). Design and control of tensegrity robots for locomotion. *IEEE Transactions on Robotics*, 22(5):944–957.

[83] Perez, A., Platt, R., Konidaris, G., Kaelbling, L., and Lozano-Perez, T. (2012). LQR-RRT*: Optimal Sampling-based Motion Planning with Automatically Derived Extension Heuristic. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

[84] Pinaud, J.-P., Masic, M., and Skelton, R. E. (2003). Path Planning for the Deployment of Tensegrity Structures. In *Smart Structures and Materials 2003: Modeling, Signal Processing, and Control*, volume 5049, pages 436–448.

[85] Platt, R., Kaelbling, L., Lozano-Perez, T., and Tedrake, R. (2012). Non-Gaussian Belief Space Planning: Correctness and Complexity. In *ICRA*.

[86] Porta, J. M. and Hernández-Juan, S. (2016). Path planning for active tensegrity structures. *International Journal of Solids and Structures*, 78:47–56.

[87] Prentice, S. and Roy, N. (2009). The Belief Roadmap: Efficient Planning in Belief Space by Factoring the Covariance. *IJRR*, 28(11-12):1448–1465.

[88] Reif, J. (1979). Complexity of the mover's problem and generalizations. In *the IEEE Symposium on Foundations of Computer Science*.

[89] Rhode-Barbarigos, L., Schulin, C., Ali, N. B. H., Motro, R., and Smith, I. F. C. (2012). Mechanism-based Approach for the Deployment of a Tensegrity Ring-Module. *Journal of Structural Engineering*, 138(4):539–548.

[90] Rocchini (2007). Hausdorff distance. https://en.wikipedia.org/wiki/Hausdorff_distance.

[91] Sabelhaus, A. P., Bruce, J., Caluwaerts, K., Manovi, P., Firoozi, R. F., Dobi, S., et al. (2015). System design and locomotion of SUPERball, an untethered tensegrity robot. In *International Conference on Robotics and Automation (ICRA)*, pages 2867–2873. IEEE.

[92] Shirdhonkar, S. and Jacobs, D. (2008). Approximate earth mover's distance in linear time.

[93] Simeon, T., Laumond, J.-P., and Nissoux, C. (2000). Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 41(6):477–494.

[94] Sivaramakrishnan, A., Littlefield, Z., and Bekris, K. E. (2019). Towards learning efficient maneuver sets for kinodynamic motion planning. *ICAPS Workshop on Planning and Robotics (PlanRob)*.

[95] Skelton, R. E. and de Oliveira, M. C. (2009). *Tensegrity Systems*.

[96] Skelton, R. T. and Sultan, C. (1997). Controllable Tensegrity: A New Class of Smart Structures. In *Smart Structures and Materials 1997: Mathematics and Control in Smart Structures*, volume 3039, pages 166–178.

[97] Smith, R. ODE: Open Dynamics Engine.

[98] Spong, M. W. (1997). Underactuated mechanical systems. In Siciliano, B. and Valavanis, K. P., editors, *Control Problems in Robotics and Automation, Lecture Notes in Control and Information Sciences*.

[99] SunSpiral, V. (2012). NASA Tensegrity Robotics Toolkit. `ti.arc.nasa.gov/tech/asr/intelligent-robotics/tensegrity/NTRT`.

[100] Surovik, D. and Bekris, K. E. (2018). Symmetric reduction of tensegrity rover dynamics for efficient data-driven control. In *Earth and Space 2018: Engineering for Extreme Environments*, pages 877–887. American Society of Civil Engineers Reston, VA.

[101] Surovik, D., Wang, K., Vespignani, M., Bruce, J., and Bekris, K. E. (2019). Adaptive tensegrity locomotion: Controlling a compliant icosahedron with symmetry-reduced reinforcement learning. *The International Journal of Robotics Research*, page 0278364919859443.

[102] Sussmann, H. (1987). A General Theorem on Local Controllability. *SIAM Journal of Control and Optimization*.

[103] Thrun, S. (2000). Monte-Carlo POMDPs. In *NIPS*.

[104] Urmson, C. and Simmons, R. (2003). Approaches for Heuristically Biasing RRT Growth. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1178–1183.

[105] van de Wijdeven, J. and de Jager, A. (2005). Shape Change of Tensegrity Structures: Design and Control. In *American Control Conference (ACC)*, pages 2522–2527.

[106] van den Berg, J., Abbeel, P., and Goldberg, K. (2010). LQG-MP: Optimized Path Planning for Robots with Motion Uncertainty and Imperfect State Information. In *RSS*.

[107] Van Den Berg, J., Patil, S., Aterovitz, R., Abbeel, P., and Goldberg, K. (2010). Planning, Sensing, and Control of Steerable Needles.

[108] Webb, D. and van Den Berg, J. (2013). Kinodynamic RRT*: Asymptotically Optimal Motion Planning for Robots with Linear Differential Contstraints. In *ICRA*, pages 5054–5061.

[109] Wilmarth, S. A., Amato, N. M., and Stiller, P. F. (1999). MAPRM: A Probabilistic Roadmap Planner with Sampling on the Medial Axis of the Free Space. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1024–1031, Detroit, MI.

[110] Wroldsen, A. S., de Oliveira, M. C., and Skelton, R. E. (2006). A Discussion on Control of Tensegrity Systems. In *IEEE Conference on Decision and Control (CDC)*, pages 2307–2313.

[111] Xu, X., Sun, F., Luo, Y., and Xu, Y. (2013). Collision-free Path Planning of Tensegrity Structures. *Journal of Structural Engineering*, 140(4):04013084.

[112] Yershova, A., Jaillet, L., Simeon, T., and LaValle, S. M. (2005). Dynamic-domain RRTs: Efficient Exploration by Controlling the Sampling Domain. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

[113] Zucker, M., Kuffner, J., and Branicky, M. S. (2007). Multiple RRTs for Rapid Replanning in Dynamic Environments. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.