Using Multiple Fidelity Models in Motion Planning

Breelyn Kane Styler

CMU-RI-TR-18-15

27 April 2018

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Reid Simmons, Chair
Maxim Likhachev
Siddhartha Srinivasa (University of Washington)
Kanna Rajan (NTNU/UPORTO)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

Hospitals and warehouses use autonomous delivery robots to increase productivity. Robots must reliably navigate unstructured non-uniform environments which requires efficient long-term operation that robustly accounts for unforeseen circumstances. However, unreliable autonomous robots need continuous operator assistance, which decreases throughput and negates a robot's benefit. Planning with high fidelity models is more likely to lead to more robust plans, but is not needed in many situations. More specifically, a complex model is an inefficient use of plan-time computation resources when a robot navigates a flat simple environment, but a simple model, that can generate plans quickly, may fail to capture complex environment locales leading to task impediments.

This thesis presents a planning framework that reasons about multiple models for efficiently generating a single motion plan without sacrificing execution success. The framework chooses *when* to switch models and *what* model is most applicable within a single trajectory. This has the effect of focusing the use of complex models only when necessary. The framework also addresses uncertainty by adding variable uncertainty to models in the form of robot padding. The footprint padding is not fixed but is automatically chosen during plan-time based on the ability to find robust plans.

The approach is evaluated by simulating a mobile robot with attached trailer through various hospital environments. Our simulation experiments demonstrate that multi-fidelity model switching increases plan-time efficiency while still maintaining execution success.

Acknowledgments

I've greatly enjoyed my time at CMU which includes my early time as a master's student, and an original Girls of Steel mentor (thanks George). I would like to acknowledge all of my valuable RI friends and colleagues (my support group which includes my terrific husband), the RI support staff (Karen and Suzanne!), my immediate family (Alex, Amayla(1yr), Morty(crazy vizsla)), and my family (my Dad [1949-2015] who started my CS path, brother, mom, step-mom, step-father, forever young 94-yr old grandma, and all extended family including the in-laws). Lastly, I would like to acknowledge my thesis committee (Max, Sidd, and Kanna!), and most importantly my advisor Professor Reid Simmons. I am forever indebted to Reid for all of his mentoring. He is honest and has a keen attention to detail. I will always respect him as a smart ethical researcher, colleague, and friend. I will miss my time at CMU.

Contents

Chapter 1

Introduction

Autonomous transport robots increase efficiency and productivity of common tasks. For example, hospitals utilize robots for medication and linen delivery [72], [9], and autonomous pallet trucks help keep shelves stacked and organized [57]. For these robots to be usable in the future, they need to operate dependably, long term. Dependable autonomy is also important in domains with limited human intervention, such as large teams where operators must divide their attention between robots [15], [19], or exploration environments unsafe for humans [4], [75]. To be successful, these domains require a robot to act independently for long time periods. In these cases, there is limited operator support, therefore the autonomy of the robot is essential.

Additionally, robots working in environments with humans must dependably execute tasks. For example, a person having to care for a sick relative could utilize a robot for routine chores allowing the caregiver to focus time on more meaningful patient tasks. The robot would be expected to perform assistive tasks without needing constant monitoring. In this case, the reliability of the robot's autonomy directly impacts the caregiver's autonomy. This reliable autonomy is also important in environments where task operation with humans is not hardcoded, such as hospitals [61] and homes [88], [34]. Hospitals and homes require successful execution of the same task over differing conditions. Unlike the static environment present in automotive factories, the robot must adapt reliably as the task's circumstances vary.

Robot systems ensure reliable autonomous operation in various ways. Related work, [98], has enumerated hard coded contingency strategies for plans that deviate from expectation. For rare situations, however, this enumeration is difficult to pre-determine, as not all cases are seen in advance. An alternate approach is to learn control strategies [59], but without a model, gathering enough data to account for these unexpected situations is challenging. These non-model based approaches suffer because it is impossible to anticipate all possible scenarios.

Model based approaches generalize the planning space with model approximations to cover unforeseen problems, and, therefore, anticipate a wider range of potential failure conditions. For planning efficiency, model approximation also allows tractable global planning times. Models can be of arbitrary fidelity for more accurate interaction modeling. Unfortunately, a single model is insufficient because it does not anticipate the non-uniform environments encountered during a robot's long-term operation. For that reason, model approximation sacrifices model fidelity for computation feasibility which increases failure rates in complex environment locales where simple lower fidelity models are insufficient. In summary, a single model may be successful the

majority of the time, as the robot encounters common environments, but not in all cases. For example, a hospital robot navigating a hallway that largely goes unchanged might assume a simple model with flat terrain and known obstacles. Issues can occur, however, on a rare busy, rainy day that introduces puddles and misaligned gurneys to the hallway. Navigating these unforeseen obstacles now requires a more complex environment view, yet was not anticipated. This is why we take a multi-model approach. Multiple models are useful for unstructured environments which include open areas appropriate for simpler models and constrained or difficult areas that require modeling more detailed environment interactions. The decision of which planning model to use introduces a tradeoff between efficiency and robustness.

Our approach uses a multi-model hierarchy for navigating environments efficiently and robustly. We want the robot to conserve planning resources when appropriate while also executing successfully. Efficient planning requires being conservative with the use of complex higher fidelity models which capture more precise interaction but are often expensive to use. Alternatively, a simpler lower fidelity model allows for tractable planning times by decreasing the planning search space, but fails to capture detailed environment interaction. Therefore, there should be a balance during planning of when to choose between high or low fidelity models. This requires reasoning about a model choice. As previous work suggests, [37], it is unclear the correct interaction level to model for successful task execution. Rather than arbitrarily choosing a planning model, we leverage a hierarchy which organizes models by fidelity in order to determine the model detail necessary for a particular part of the environment.

Multi-model switching tries to balance the tradeoff between planning efficiency and robustness. Many previous works switch between models for improved planning efficiency but often limit this to only two tiers. Such works have had success changing the planning resolution locally around the robot as it travels, [89], or using higher global plans to guide more informed local planning and search, [48]. Other works have successfully switched between models and discuss *when* to switch, but do not specifically search among models to determine *what* model is most useful to plan in at a particular point in the task, [32]. We believe reasoning about *what* model to use is important for robustness.

If the robot always planned using the highest fidelity model possible, its failure rate would be reduced, so why not always use the highest applicable model? Planning in the highest fidelity space may not be necessary at all points during the robot's task. For example, if the robot is moving through large unconstrained open spaces that do not require many turns, a lower fidelity model that only considers geometric planar motion might suffice. Therefore, the highest fidelity space could be computationally expensive. Secondly, higher fidelity models often require more information (such as the coefficient of friction of the terrain) that is not known or unavailable. Without knowing this information, the value in planning in the higher fidelity model is lost.

We provide a planning framework that generates a single motion plan from varying levels of model fidelity. We switch between a richer set of models than previous work to further balance planning resources and robust execution success over varied environments. We are not developing a new planner, but creating a framework for current motion planners to leverage multiple models. The types of planning models we are focusing on describe state and action information explicit to the robot's interaction with the environment. We assume these models are given. We also assume the higher fidelity the model the more accurately it captures details of robot interactions with the environment and more closely approximates the robot's underlying controller.

We present a probabilistically complete plan-time approach that uses Rapidly Exploring Random Trees (RRTs) for path generation over multiple models in continuous space. First models are organized in a hierarchy based on fidelity. Then an initial motion plan is generated in the lowest model. The approach then detects whether any parts of the lower fidelity plan are infeasible for execution. This is done by checking the current plan within the highest possible model in our hierarchy. The partial infeasible sections of the plan are repaired using re-planning through model selection. The model selection process autonomously selects the most applicable higher fidelity model in the hierarchy. This is done by translating lower models into higher models, and then choosing to re-plan in the model which fails to propagate the infeasible part of the plan. This higher fidelity model is used to locally plan to an intermediate goal where the previous lower fidelity plan is resumed.

In addition to varying fidelity models, our approach also reasons among models with added uncertainty. For low fidelity models, uncertainty can cover imprecise details without requiring knowledge of specific robot interactions that are hard to capture. Additionally, for higher fidelity models, adding uncertainty is a way to deal with an inaccurate model that does not match the robot's interactions exactly. Our approach reasons about variable amounts of uniform uncertainty by changing the amount of padding over the robot's footprint. This type of uncertainty accounts for inaccuracies in the robot's motions and sensed obstacle locations. The variable padding amount is automatically determined at plan-time.

The framework not only determines when to switch between different model types, but also what model is most appropriate at a given point in the path. We address inaccurate modeling through model selection. At any point in the plan the model selector either chooses to switch to a model that includes uncertainty with how the robot interacts or switches to a model with more accurate fidelity representation. Switching has the effect of combining the robustness of a high-fidelity plan with the efficiency of an abstract representation by using higher fidelity models only when necessary. While choosing between low and high fidelity presents a tradeoff between plan time and execution success, switching among models with variable footprint padding introduces path cost. Low fidelity models with added padding save on plan time, compared to the highest model, and increase success rates, but do so at the cost of longer paths. Also, models with padding sometimes cannot even find a path if there is too much padding. This is why it is important to not use larger footprint padding amounts than necessary.

Multi-model switching can occur at plan-time or execution time. The decision of when to switch between models occurs when new information necessary for higher fidelity models becomes available. Therefore, it is possible to use a lower fidelity model with more uncertainty and then switch to a more certain model as information becomes known during execution. Execution time switching is necessary when new information becomes available that did not exist at plan-time. Alternatively, if the information is available at plan-time it does not make sense to wait for the robot to fail before re-planning with a different model.

We ran experiments in a high fidelity physics simulated hospital world with a differential drive robot. Our testing uses multiple wheeled mobile robot planning models. In our results, we demonstrate failure rates and planning times when using a fixed single model versus autonomously switching between models of varying fidelity.

1.1 Motivating Robot Examples

This work is motivated by observing various pallet truck and hospital cart transport scenarios.

One example scenario is a forklift robot trying to pick up a pallet. An initial planning model might consider interactions between two static objects. The robot would plan a motion path to align with the pallet. The robot would then proceed to move under the pallet with its forks lowered. If the pallet was empty, the robot could fail to pick the pallet up by instead pushing it along the floor. A model that reasoned about the pallet's mass and the pushing interaction between two objects that could move (i.e., not static) could reason about the minimum force necessary to pick up the pallet without pushing it.

A second example is for a hospital robot delivering linens. A failure could occur with transporting an object around a tight turn. The original model for plan generation may just consider maneuvering through an open space. The robot may not be able to take the turn directly on the planned trajectory, or might have to stop if it passes the waypoint. A model that contained information about trailer swing and transport object mass could reason more accurately about the robot's turning radius around tight areas.

1.2 Problem Statement

In summary, we increase robot robustness by deciding when and what higher fidelity model to re-plan in that inherently incorporates more information. We assume the reason for failure is because the robot's current planning model does not have enough detail for the current task. In this paradigm, a robot's ability to make decisions is only as good as its models. Therefore, the robot's goal is to decide at any point during its plan generation what the most appropriate level of fidelity is for the given task.

We formulate this as a motion planning problem where the goal is to find a continuous collision free path, ζ, that takes a robot from a start to end configuration. Planning requires [55]: a *workspace* W, with defined obstacle regions, that contains a *robot representation*, R, and the *configuration space*, C, of possible transformations that can be applied to the robot.

Our models can contain different representations of W, R, and C. We generate a continuous collision free multi-model path that must also check collision free within our highest model. Model selection determines at any point in the generation of ζ what model is necessary to take $c_n \in C$ successfully (collision free) through W. This requires generation of partial ζ_i until ζ includes c_{start}, $c_{end} \in C$ and checks collision free in the highest model of W, R, and C.

1.3 Thesis Statement

> An autonomous robot can balance planning efficiency and navigation robustness through non-uniform environments by automatically selecting from a hierarchy of models considering fidelity and uncertainty.

We would like to address the following questions:

- What planning model contributes the 'best' information at a particular time?

- How do representations capture information and account for uncertainty?
- How does the model selection strategy search for a planning model that maximizes task success while remaining computationally tractable?

1.4 Contributions

The primary contribution for this thesis is **providing a framework for multi-fidelity plan-time model switching**. We also discuss **adding uncertainty to models by automatically determining variable padding**, and their placement in a hierarchical model graph. Our framework includes **model selection strategies for models with higher fidelity and variable padding**. Model selection addresses *when* to switch models and *what* model is most applicable. Lastly, we demonstrate **advantages of the multi-fidelity model switching framework** through simulation experiments.

1.5 Overview

The remaining thesis chapters are summarized below:

Chapter 2 - Related Work In this chapter we provide an overview of how other works have motivated our current work. We touch on how related works deal with plan failure due to model approximation, and how those specific to model switching address the tradeoff between planning efficiency and robust execution. Lastly, we touch on related work that deals with uncertainty and how our work compares with those that utilize safe planning regions.

Chapter 3 - The Model Hierarchy This chapter discusses our definition of model and how the models are organized hierarchically. It includes information for models both with and without added footprint padding.

Chapter 4 - Planning with a Hierarchy of Models Our main approach is discussed in this chapter. We describe the parts of our approach and how we generate a single mixed fidelity plan. This chapter also includes how we decide to add footprint padding to our original models and how these additional padding models affect the initial approach. We include some discussion and enhancements that directly relate to our results. Lastly, we also include a formalization of the approach at the end of this chapter.

Chapter 5 - Implementation for Experiments In this chapter we provide details about our simulation testing environments. This includes the type of robot we simulated, our simulator, and more details on the planning models that were used. We then touch on more implementation specific details of our algorithm for running these models for our experiments.

Chapter 6 - Testing Environments and Experiments In this chapter we discuss the simulation environments in more details including results of our approach for various environments. We provide tests for evaluating single model performance, switching models, and switching models with added uncertainty. For each of these evaluations we also provide information on average statistics over all worlds as well as highlighting some example world results.

Chapter 7 - Future Work In our future work, we list possible framework improvements and extensions which include details for creating a more informed model selector.

Chapter 8 - Conclusions Our conclusion section lists the takeaways from our results and lists our thesis contributions.

Chapter 2

Related Work

The related work is divided into three sections. The first discusses model work in robotics. This motivates the use of multiple models in our approach. Then we discuss how work mitigates errors that arise from approximating the planning space with models. This is divided into two parts. It starts with a high level overview of robust planning in robotics and then includes a section which relates to our framework's uncertainty models. The last section discusses related work in model fidelity which balances between plan efficiency and execution success. This is a tradeoff that we also consider with our model switching framework.

Our novelty is in providing a framework that hierarchically leverages more models than related work and reasons explicitly about how each model contributes to execution success. Our work attempts to minimize planning time without giving up execution success.

2.1 Models in Robotics

Models are often used in robotics. Their application shows a tradeoff between model fidelity and computation time where higher fidelity models reduce uncertainty in the environment. Previous work demonstrates examples of this by showing increased robustness for humanoid balancing [91] and higher fidelity wheeled robot models that more accurately capture real world trajectories [85]. Our work leverages models already created in the robotics community.

Related work also recognizes the possibility of representing the world with multiple models. Work by [35] discusses multi-modal motion planning techniques. They describe the configuration space as containing varying dimensions that can span multiple sub-manifolds. This space discretization inspires our use of multiple models as does other works which scope the planning model space at different detailed levels. This includes work in physics based planning [102], physics based models for manipulation planning [22], and a hierarchical perception/control structure [21].

Models also approximate different parts of the planning environment. Kuipers divides the world symbolically [52]. Works in motion planning break up the proximity of grid cells near the robot into coarse or fine resolution[45], and [8]. There are works in the combined task and motion planning community that divide the planning space into a hybrid space, mixing both symbolic and geometric representations [42], [20], and [99]. Also, work in [7], and [26]

hierarchically group underlying states in the planning space into different state approximation levels. Lastly, ontologies provide a knowledge base similar to a model collection. Ontologies represent relationships and concepts for common understanding of a domain [87] [28].

All of these works inspire the idea that richer models allow a robot to be more robust when making decisions.

2.2 Mitigating Model Approximation Error

Artificial Intelligence (AI) planning techniques depend on models that generate deliberative actions and controls for the robot to execute. The planning community uses these models to make an otherwise difficult planning problem tractable. This is true in cases where it is intractable to model higher dimensions, or information is not available at plan start. Work that supports the necessity of approximation discusses assumption architectures for tractable planning, [69]. They argue that it is not possible to generate a plan from initial conditions to handle every case and assumptions are required. Therefore, we hypothesize approximations are necessary to plan in the high dimensional space of the real world and misinformation arises in planning models due to this approximation. Adele E Howe [40] suggests that a lack of understanding with complex environments introduces errors. She recognizes failures from "obsolete, uncertain or limited information". Work by [37] also discusses issues that occur due to abstractions. They focus on robust robot assembly noting that many failures occur during the robot's interactions with the environment. Our work focuses on using AI planning techniques to overcome misinformed models. The following section discusses how related work has addressed model inaccuracies. Next we discuss various robust planning techniques in robotics.

2.2.1 Robust Planning in Robotics

Robot autonomy requires the robot to reason independently and react appropriately to unanticipated events. Work by [11] recognized that related work for reliable robot autonomy techniques reason about a lack of information to increase autonomous robot robustness. In the following section, we touch on a small subset of approaches for handling model inaccuracies through planning. These include: contingency strategies, single model adaptation, plan adaptation, and re-planning.

One approach is to enumerate hard coded planning strategies during the design phase. Contingency planning is an example of a priori responses created from known events or learned anticipatory considerations. Works such as [98] come up with path alternatives in advance. These planners also reason about uncertainty in an attempt to capture all possible events that could occur during execution. A review of planning under uncertainty is presented in [10] where extensions for capturing misinformation and uncertainty are considered in both classical and decision theoretic planning. They recognize that these approaches rely on complete, detailed models.

Many works also reason about potential alternatives during plan time and decide when to switch to specific plans during execution. Work in [98] generates contingency plans in advance for each non-deterministic action, and for each time an observation action must be taken. They

use execution monitoring to determine which branch to take at runtime. Other examples of work that switches between planning space hierarchies during execution occurs in [42], and [29]. The switching delays decision making until execution to help reduce uncertainty.

Precoded strategies are also present in works that use universal single models to capture all situations. Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs) are examples. Similarly, single trajectory optimization improves a single model. Trajectory optimization depends on features chosen for training and a cost function that captures robust scenarios during training. Motion planning algorithms that use optimization iterate on naive initial trajectories to minimize the cost of a final solution trajectory as discussed in [82] and [43]. Lastly, learning is also used to address model inaccuracies. This includes works which adapt a model that deviates from expectation as in [66] and [83].

Other works adapt the plan as it is formed. They do not alter the underlying model but vary the search space strategy, bias toward informative heuristics, or re-plan and repair the plan from scratch. Sample-based motion planning algorithms sample the state space randomly and generate connections between neighboring sample points. One example of this is with the Rapidly Exploring Random Trees (RRTs) algorithm [54]. Execution Extended RRT rebuilds from scratch and adapts plans by biasing search to previous plan waypoints [13]. Other methods focus more on adaptation by eliminating a range within the invalid path and re-growing a search tree in that invalid space [25]. Work by [105] also maintains entire previous searches by using a forest of disconnected RRTs for planning. Another way to bias the search in sample-algorithms is to bias the sampling rather than bias towards old plan searches, this was shown in [106]. They bias samples to features within the workspace, the 2D or 3D environment representation that describes the robot and obstacle geometry.

Re-planning is a technique that has been in symbolic planning since early mobile robots [27]. Re-planning is also prevalent in the motion planning community. Motion planning uses re-planning in incremental search and sampling based algorithms. Re-planning when new information is found. Incremental search in motion planning builds off the graph based A^* search algorithm. This algorithm finds a path from start to goal by minimizing edge costs on a directed graph. Incremental heuristic search combines both re-planning (restarting at plan deviations) and plan adaptation (using more informed heuristics and past information) to speed up the search process. Examples include: D^* [90], LPA^* , and D^* Lite [50].

Re-planning has also been shown to handle probabilistic planning domains by outperforming universal model algorithms (MDPs). The success of a deterministic action selection planner, FF-Replan [100], in the International Probabilistic Planning Competition demonstrates this. They describe FF-Replan as having a similar style to contingent planning without explicitly needing to consider action effects. They leverage a deterministic planner known as the Fast-Forward planning system (FF) [38]. They recognize that the competition problems were simple by not "exercising the full difficulty" that is present in a probabilistic domain. They also note the algorithm's strength is due to the progression of fast and efficient informative heuristics.

Our work uses a combination of plan adaptation and multiple model switching. We generate a single multi-fidelity motion plan by re-planning over non-uniform environments.

2.2.2 Models with Uncertainty for Navigating Safely

Uncertainty is another way to address inconsistencies in approximate models. In this section, we touch briefly on related work with navigating under uncertainty, and discuss how such works are relevant to the padding models which extend our approach.

Navigating with uncertainty has been studied vastly and we will not provide an exhaustive overview. Many works investigate robot state uncertainty and localization, but we are interested in those that focus on uncertainty in robot motion and obstacle locations. We draw motivation from: uncertainty works that hierarchically break up the space [41], works that leverage a higher level controller for execution feasibility checking, and uncertainty work that focuses on safety.

It is important to take into account the execution controller when considering path feasibility rather than take the planned path as a given. Two related works that identify this include [95], [14]. They recognize how motion uncertainty can cause path deviations, and that it is important to consider these deviations offline. The first work generates many candidate paths and evaluates their success rates, and the second builds a single path by reasoning in a belief space. Our work differs by seeding the path generation with multiple simpler representations to minimize planning times while utilizing a higher fidelity model for considering path deviations.

Planning with varying robot footprint padding changes the robot's distance from obstacles with respect to uncertain obstacle locations. This closely ties to related work that reasons about robot navigation safety. Two recent works include: [3] and[94]. Each work contains motivations that we share. The first work touches on multiple topics that also motivate our approach such as uncertainty reasoning is cheap and being overly conservative is inefficient. The second paper recognizes that collision probability " from a configuration onwards depends on prior history. ". We use this idea when re-merging re-planned partial paths as the path configuration history is necessary for properly checking in the highest model and accurately determining repair points.

Unlike our work, both safety works referenced above, set a hardcoded risk factor, or max convergence time for generating the safe path. Our algorithm varies a footprint padding threshold during the planning process. Both works have a more sophisticated geometric way of determining safety buffers than using uniform padding over the robot's footprint.

Lastly, there are related works in [63], [64], and [84] where hardcoding a padding value around the robot's footprint is used to increase successful robot execution. Therefore, hardcoding a footprint padding value is common, even if it's not the object of study. Even though a hardcoded value is sufficient for frequently encountered environments, it may not be possible to determine a single value during long term navigation. This is why automating the padding amount, as is done in our approach, is important.

The novelty with our uncertainty padding extension is creating an approach which combines multiple ideas from the literature which include: switching between models with added padding for decreased planning times, determining safe padding amounts automatically, utilizing a higher fidelity model checker for execution feasibility, and staying probabilistically complete with respect to the highest model.

2.3 Balancing Planning Time and Execution Success

The concept of using higher fidelity to gain a more accurate representation and improve plan quality is discussed in many related works. This includes works which consider model fidelity but do not explicitly switch models. Related work that changes fidelity must also reason about computation costs incurred with higher fidelity models. The tradeoff between planning representation and efficiency has been recognized in the planning community since the 1980s, [62]. In this section, we discuss related work that reasons about fidelity for balancing planning efficiency and execution success.

2.3.1 Switching Between Approximate Models

The model choice to plan in creates an inherent tradeoff between planning time and robust path execution. Many previous works use different forms of model switching to address this balance. Previous work in Variable Level-Of-Detail Planning, by [103], relax the local space to ignore more complex details that occur far in the future. A similar approach is applied to multi-robot planning with sub-dimensional expansion [97] where each robot initially plans independently. The search space dimensionality increases to a joint robot space after robot interactions are found. This effectively is a switch between different approximations of the space. Additionally, works that break the planning problem into sub-problems also reason about different approximations [17]. This includes relaxing optimality by changing the heuristic used to search the space [2]. Similarly, adjusting the resolution in a plan is a form of switching. The following works adapt the planning space resolution locally around the robot, but do not vary the model fidelity throughout the same plan[45], [89], [8]. Our strategy varies models throughout the same plan searching directly in continuous space and changing dimensionality in both the state and action space.

The following paragraphs review previous works which focus on when to switch between models rather than reasoning about what detail the model contains before switching. Our work contains an additional model selection stage that most related works do not. This stage determines what detail level is most applicable for re-planning.

Fixed strategies focus on the *when* for switching between levels of detail. This is apparent in [39] and work that only has two levels, such as a higher level global plan that guides a low level continuous planner [48]. Examples of approaches that often take a two tier approach, where the planning representation is relaxed at various points during the planning process, include: [86], [78], [77], [39], and[48]. Hybrid planning also switches between a distinct discrete plan and more focused continuous planner [78] as in the SyClop planning framework. This also occurs in [16] where a contingency "channel" with many possible motion plans guides a lower-level potential field controller. Other work [29] divides the planning space between task and observation level plans similar to the division between global and local planners. They switch between a fast sequential "classical" planner, and more expensive decision-theoretic planning for abstracted sub problems. Thesis work in [101] notes that dual-layer approaches are weak because global plans are generated without low level details. A global plan may be generated that is impossible for a controller to follow because the global plan does not provide any guarantees on the local planner's ability to find a valid solution.

Work by [42] demonstrates switching between state spaces in a fixed top-down fashion. Much of the hierarchical planning community plans top-down increasing the level of planning detail. [26], [99], [6]. This includes anytime approaches which find an initial solution and then consider more detail to further refine it as time allows as shown in [104]. The motion planning community also include anytime variants of the shortest path algorithm [96], [58].

Multi-modal and multi-stage work also contain fixed switching strategies. Work in [36] and [35] switch between different planner types based on a modal discretization, but do not reason explicitly about the detail they are switching to. Lastly, work by [93] for task motion multi-graphs also contain predefined points where switching occurs. The decision of when to switch is predefined between tasks and the selection criteria is based on computation time.

Our work resides in the motion planning domain. Work more similar to ours utilize re-planning and change fidelity throughout the planning space. One such work graduates motion primitive fidelity along a state lattice [76]. They change the fidelity between re-plans in the motion planning workspace. They also recognize that "partially or completely unknown" regions of the space can use lower fidelity representations than regions most relevant to the current problem. The work also claims that previous multi-resolution work is more systematic while theirs allows different resolution regions to move over time. The fidelity around the robot is fixed and moves like a sliding window, which is different than our mixed fidelity plan.

Our work is mainly inspired by Gochev's previous work with adaptive dimensionality [32] [30] [31]. That work divides the state space into two parts; a high dimensional and low dimensional graph with defined transition probabilities. They use a state lattice planner with pre-computed grid transitions. Unlike other works, they are able to mix the two subspaces into a single plan. Our work also has this same effect, of generating plans that mix dimension spaces, through adaptive switching. They have a tracking phase (similar to our plan checking stage) to determine where to insert higher fidelity states in a low fidelity plan. This phase requires a search in the higher space during tracking that is computationally more expensive than our checking phase. Their algorithm reasons about a single high fidelity state at a time. Our algorithm does not require this explicit reasoning. We insert partial paths found directly from the continuous path space. These partial paths automatically detect the size of the infeasible region and plan accordingly. Our novelty is we use a complete hierarchy of models. Therefore, our algorithm contains an additional model selection stage.

Chapter 3

The Model Hierarchy

Multiple models are useful for motion planning in non-uniform environments. Simpler models allow for faster planning times and generalize over more environments. However, they miss details that occur less often and are important for reliable long-term operation. Complex models provide a more exact model of the robot's interaction with the environment increasing successful execution rates but require more computation

In this chapter we define what we mean by model Section 3.1, and describe our definition of uncertainty model 3.1.1. We then discuss what we mean by fidelity and introduce a definition for translating between models, Section 3.1.2. Lastly, we describe the organization of the model hierarchy used in our framework, Section 3.2.

3.1 Model Definition

In this work we use the term *model* to refer to a subset of the planning space used for generating a continuous motion trajectory from a start to goal configuration subject to vehicle and obstacle constraints. At each time-step the validity of the robot's configuration state is checked by detecting if it intersects applicable obstacles in the environment using a collision checker. Figure 3.1 describes what is needed to generate a collision free motion plan. The first two boxes describe information contained in a model. Models include a robot model, and environment model. The robot model describes the robot's state and motions that can vary from geometric spatial actions to differential motion equations. The environment model represents obstacles in the planning workspace. A more formal definition of model is included in Chapter 4.5.

Robot models contain a state vector $[x(t) \mid x \in R^n]$, optional control input $[u(t) \mid u \in R^n]$, and a description of motions between states. This motion mapping may be defined as a system of motion equations of the form $\dot{x} = f(x(t), u(t), t)$ which map from $u(t)$ to $x(t)$, subject to kinematic and dynamic constraints. These can be described using first order differential equations for velocities and higher derivatives for accelerations. The robot's state also includes the robot geometry which can contain aspects such as shape, mass, material, and coefficient of friction. The models we use consider only the shape attribute and contain both first and second-order differential motion equations.

What is Needed to Generate a Plan?

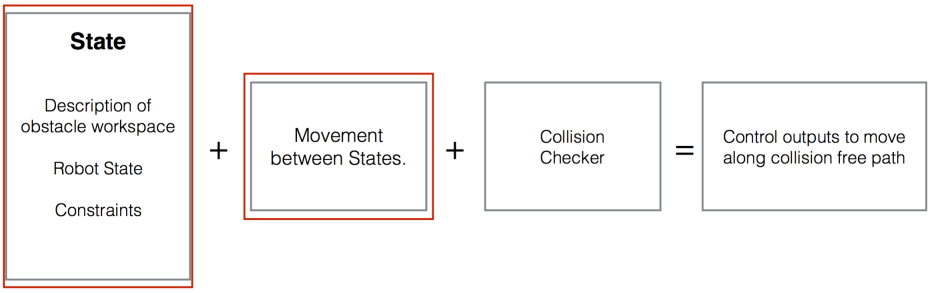


Figure 3.1: First two boxes define model.

Environment models describe the obstacle geometry represented in two or three dimensions. These models also contain time dependent features such as traffic lights or automatic doors.

The collision checker checks if two objects intersect. The collision checker is the same for all models. The planner propagates the robot's control inputs between states by integrated the equations of motion. At some pre-determined time-step, the planner checks the validity of the robot's current state by collision checking the robot's mesh, at that waypoint, with environment obstacles.

Models vary in fidelity and can be fully contained within each other, partially overlap, or not intersect at all, Figure 3.2. Robots with multiple parts further divide based on subsystems. For example, models may be specific to a robot base, or a robot manipulator. Higher fidelity models could combine subsystems such as the base and manipulator into a single model.

This work does not focus on the construction of planning models. It assumes motion planning models are available. We leverage models from the motion planning community (see, for instance, [55].). More specifically, we leverage models for a wheeled mobile robot.

The unique variables used to describe the configuration of the robot at any point in time is described by the vector $\vec{q}$. Where $\vec{q} = [x(t), u(t)]^T$ for a particular t. We use this configuration vector to label the models.

Four possible wheeled models are:

$$
\begin{aligned}
M_1 &= \vec{q} = [x, y] \\
M_2 &= \vec{q} = [x, y, z] \\
M_3 &= \vec{q} = [x, y, \theta] \\
M_4 &= \vec{q} = [x, y, z, \theta]
\end{aligned}
$$

In this simple example, models $[m_2, m_4]$ contain an additional z-variable which increases the fidelity of robot and obstacle meshes sent to the collision checker. A three-dimensional environment is used instead of the two-dimensional environment collision checked in $[m_1, m_3]$. Models $[m_1, m_2]$ generate trajectories with planar motions. The motion equations describing models $[m_3, m_4]$ use constant curvature control arcs to generate s-curve trajectories. Planning

(a) Models can overlap, and be entirely contained within each other.

(b) Wheeled mobile robot models increase in fidelity with additional configuration variables.

Figure 3.2: Simple Wheeled Robot Models.

within varying fidelity model spaces changes the computation amount necessary for collision checking and generates varied trajectories.

One aspect of models is uncertainty and all models to some degree have uncertainty. When describing the models we divide them into two categories: Base Models and Uncertainty Models (with uncertainty greater than the base model). A base planning model is a model that is initially provided to the algorithm with some nominal footprint padding (nominal uncertainty amount). They are present throughout the entirety of the planning process and do not change.

3.1.1 Adding Uncertainty to Models

Our goal is to increase the robot's planning efficiency by adding footprint padding to existing models. Uncertainty models are base models with additional robot footprint padding. These are models added after the initial planning process. They are added automatically at run-time by determining variable footprint padding amounts.

It may be difficult to capture detailed environment interactions required for more complex models. To address these limitations, it is possible to add noise to a model in the form of uncertainty. Uncertainty models are a good intermediary model (between simple and complex models) with low plan times and higher success rates that allow the planner to stay longer in a lower representation.

Uncertainty may be represented in various way such as parametric gaussians[[80], [24], [67]] or even probabilistically with non-parametric distributions [[79] [65]]. We have chosen to represent uncertainty as padding around the robot's footprint by investigating the addition of uniform uncertainty. Uniform uncertainty addresses inaccuracies in a robot's movement and sensed obstacle locations. The uncertainty value can change by increasing the robot's footprint padding. We do not want to focus on specifics of object properties such as color or texture. Explicitly listed, the uncertainty types are:

- Robot state and actions caused by uncertainty in the motion space.
- Objects and obstacles position in the environment's state caused by uncertainty in perception space.

We increase the number of fidelity models by adjusting the robot's footprint size which is sent to the collision checker. This does not change the underlying model motions, but does change the number of collision checks within the environment. It is important for this padding amount to be conservative since planning with larger footprints create longer paths. The decision of when and how much padding to add is described in Chapter 4.2.

3.1.2 Defining Higher Fidelity and Model Translation

The main criteria for knowing when a model is higher fidelity than another is if there exists a lossless translation between models. This allows lower fidelity models to translate into higher order spaces. For example, [x y] can translate into [x y z] by assuming a constant z, but [x y z] can not translate into [x y θ] due to the loss of the third dimension. Therefore, if a lossless translation exists from model A to model B, then model B is higher fidelity than model A.

Translation through the model hierarchy is transitive. If model A can be translated into model B, and model B can be translated into model C then model A can be translated into model C. This transitive property means that it is necessary to provide translations only between adjacent models that have a direct precedence relationship. We provide examples of translation between models in Section 5.3.5.

Higher fidelity models can generalize lower fidelity models along different dimensions (for instance through more detailed environment representations, control inputs, or vehicle-terrain interaction models). It is also possible to include differential constraints in higher fidelity models to more accurately approximate the actual robot controller; other higher fidelity models include more accurate representations of the environment. We want the hierarchy to generalize along different dimensions because choosing the most appropriate model for a given task and situation is more important than just always using a model that has more information. It might be the case that a more appropriate model that is constrained to consider fewer details is more applicable. For example, different robots may have different maximum velocities for operation on particular terrains. A model that constrains this space accurately is more beneficial in environments with more slip than a model that allows any possible velocity to be sampled. Since the environment terrain can change during a robot's task, it is important for the robot to have a variety of models. The robot can make large gains by switching among subsets in the model graph before settling on specific models that help with a certain task.

3.2 Model Organization

This notion of higher fidelity leads to a hierarchy. The ability to translate one model into another without losing information forms a partial ordering. The hierarchy is organized where a model may have multiple ancestors and multiple descendants. In some cases, if there is a lowest and highest model this can form a lattice. Higher fidelity models cover lower fidelity models. The covering relation is used to compare elements in a partially ordered set. Therefore, if the higher model contains the lower model plus some additional information it also covers it.

Emulating models in others is not always straight forward. It is the case that some models may be partially but not entirely more general than others. This means that some models may have no precedence relationship between each other. In this case, they would be grouped on the same horizontal level in the hierarchy. Therefore, there is no ordering within a horizontal level between multiple models.

The organization forms a finite acyclic directed model graph $G(\vec{q}, E)$ connecting low fidelity models to higher fidelity models. First an edge is drawn between each model for which a lossless translation exists. Once all edges are found between models the graph can be reduced to form the final hierarchy. The final partial ordering is the transitive reduction of the initial directed acyclic graph of models [1]. This is also known as the minimum equivalent graph, [68], where the final model graph has as few edges as possible while maintaining the same reachability relation as the original graph. As a simple example, Figure 3.3 shows the organization of four wheeled robot models: $M_4 = \vec{q} =$ [x y z θ], $M_1 = \vec{q} =$ [x y], $M_3 = \vec{q} =$ [x y z], and $M_2 = \vec{q} =$ [x y θ]. By definition, an edge is drawn between models with lossless translation as shown on the left side of Figure 3.3. Redundant edges are then eliminated, the right side of Figure 3.3, to create the directed graph's transitive reduction which maintains the longest edge paths.

We note that base models are organized in the model graph before the planning process begins. These models comprise the initial hierarchy. Uncertainty models with additional footprint padding are added to the hierarchy at run-time based. The decision of when to add these models is described more in Chapter 4.2.

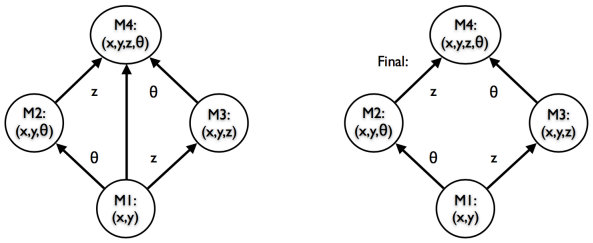


Figure 3.3: Transitive Reduction of Original Model Graph.

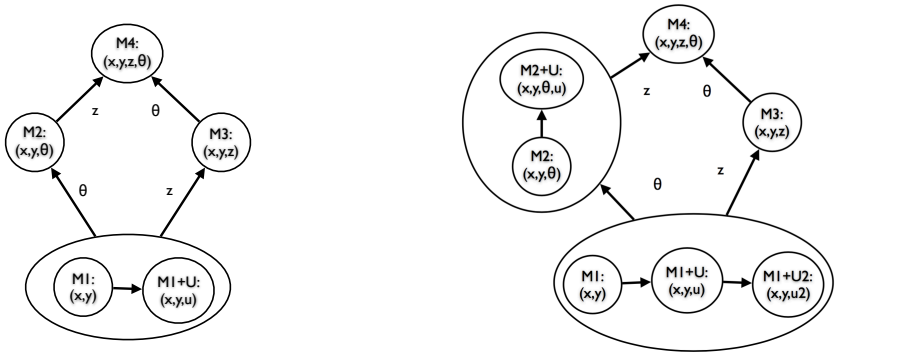
Ordering the models hierarchically is also important for model selection (discussed in Section 4.1.3). The last model that was successfully planned in for a particular space is the starting model node to search from. The model selector searches through models using the hierarchy. This hierarchy groups correlated models. The lower fidelity models should be used more for simple environments. This then leads to the more detailed models being used infrequently, which means there is little computational disadvantage to having them around. Therefore, it does not hurt to keep a lot of different models.

3.2.1 Augmenting the Planning Model Hierarchy for Uncertainty Models

Translating to a space with uncertainty means increasing the uniform robot footprint padding amount. This translation maintains the same dimensionality of the robot's state and environment.

It only changes the size of the robot's footprint. A lossless translation still exists when adding uncertainty to achieve a higher model.

If an uncertainty model is chosen for re-planning, it then needs to be added to the model hierarchy. Figure 3.4 (a) shows a single uncertainty model added to the hierarchy. The hierarchy changes to group all models with similar configuration variables and variable uncertainty amounts into a single node. Uncertainty models are higher extensions of a base model (with nominal padding) and are ordered based on the amount of uniform footprint padding. This is shown in Figure 3.4 (b) where multiple uncertainty models with the same base model are grouped together and then ordered by increasing padding amount. The alternative is to create additional edges to higher models and add equivalent footprint padding amounts for those higher models. We chose to group uncertainty models together with the same base model, and not increase the footprint padding amount for higher models more than necessary. This is discussed more in our model selection approach for uncertainty models in Chapter 4.2.2.



(a) Single Uncertainty Model Addition.

(b) Multiple Uncertainty Models with Different Base Models.

Figure 3.4: Adding Uncertainty Models to the Hierarchy.

In this chapter, we discussed our definition of *model* as it is used in motion planning. We defined model fidelity and showed how models are organized hierarchically. Lastly, we discussed what we mean by *uncertainty model* by explaining how the model hierarchy is augmented to include models with additional robot footprint padding. In the next chapter we will describe our plan-time model switching approach which includes the decision of when to switch between these models, and which model in our hierarchy to switch to for plan repair.

Chapter 4

Planning with a Hierarchy of Models

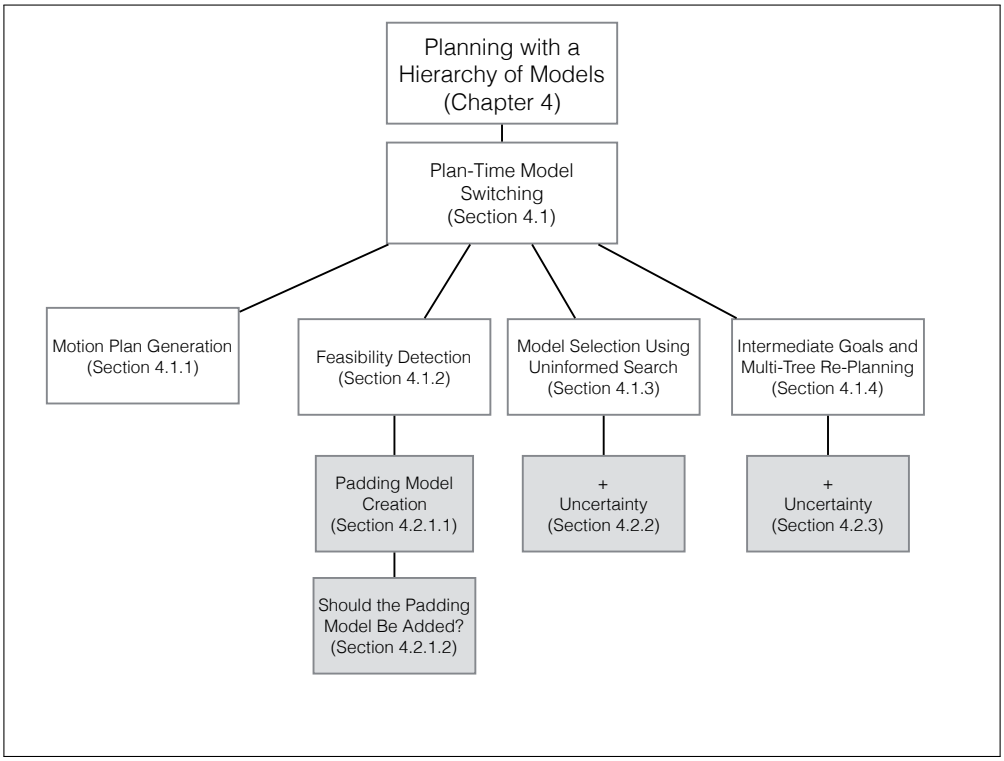


Figure 4.1: Approach Roadmap.

In this chapter we describe our approach to creating a general framework in which robots use varying fidelity models for robust motion planning. The chapter is broken into multiple sections. A roadmap of these sections is displayed above in Figure 4.1.

Section 4.1 provides an overview of the initial plan-time multi-model switching algorithm. This is a probabilistically complete plan-time algorithm for deciding what fidelity model to utilize at each point in a navigation path. We then build on our model collection by deciding whether or not to add models with additional uncertainty as described in section 4.2. With the added ability to choose to switch between models with variable padding, we expand our model selection

algorithm to now include these models.

4.1 Plan-Time Model Switching

Our robust plan generation algorithm consists of four main stages that loop:

1. Motion plan generation.
   A plan is generated for a particular start and goal state using a given model.
2. Feasibility detection.
   The plan is checked using the highest fidelity model to see whether it may cause problems during execution. This stage answers the question of *when* to switch between models in the plan.
3. Model selection.
   This stage decides *what* model to switch to for re-planning. The model selector reasons over the model hierarchy to determine what model may be sufficient to repair the plan.
4. Multi tree re-planning.
   Re-planning from various starting points to various goal points are interleaved to provide an efficient, but probabilistically complete, search algorithm.

The robot's task is to navigate from a start state to a goal state. Our system uses an environment map to generate a plan from start to goal in the lowest applicable fidelity model (Motion plan generation). This plan is the initial global plan.

The plan is then checked in the highest fidelity model to determine parts of the plan that might need repairing (Feasibility detection). Even though the high fidelity model is complex, checking a single plan does not incur much computation time.

If re-planning is necessary, due to a detected impediment such as an anticipated collision with objects in the environment, the algorithm moves to stage three (Model selection). The model selector finds the lowest fidelity model that can still feasibly generate a repaired plan. This gives further computation savings by using the lowest fidelity model applicable rather than always switching to the highest available model. The re-planning model is selected by re-testing the partial plan segment, which needs repair, in successive higher fidelity models. The first model in which checking of the previous plan segment is *unsuccessful* is the model selected for re-planning. This model is assumed to be a more informative approximation of the space.

A new plan is generated in the selected model. To try to minimize re-planning time, the use of the selected model is localized around the detected impediment by re-planning to intermediate goals on the remainder of the infeasible plan rather than re-planning all the way to the original goal. Also, waypoints before the impediment are set as separate start states which plan concurrently to guarantee probabilistic completeness (Multi tree re-planning). The first start and goal to find a plan creates a partial plan repair. The new partial plan is then merged back into the global plan. The last three steps of the process repeats until the feasibility detection stage does not find any more impediments. If this is the case, the full global plan is determined to be executable.

4.1.1 Motion Plan Generation

Motion planning generates a collision free robot trajectory from a start to goal location. The planner is given as input a description of the obstacle environment and robot configuration state that can include differential constraints. It then outputs a series of waypoints and associated waypoint controls that drive the robot through a collision free waypoint path.

What is Needed to Generate a Plan?

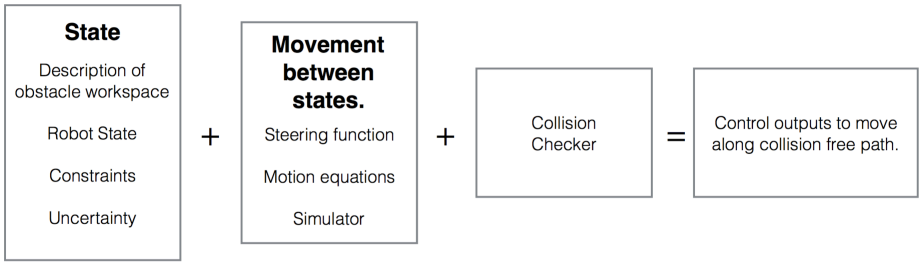


Figure 4.2: Input and output components for a motion plan

The planner takes a description of the state, motions to move between states (actions), and a collision checker. The state consists of the obstacle workspace, and robot state. It also includes the specification for the start and goal state. The obstacle workspace defines the positions of occupied obstacles in the free configuration space. The robot state includes possible configurations of the robot such as: position, orientation, joint space, and any differential constraints the robot must satisfy. The obstacle workspace and robot state can also include uncertainty to cover imprecise models.

For our purposes a model can vary the descriptions of the state and movement shown in the first two boxes in Figure 4.2. The collision checker remains the same for all of our models. The model dictates what dimensions are used during collision checking. The planner uses the boxes on the left as input to produce the output box on the right in Figure 4.2.

Our work does not create a new planner, but provides a framework where existing planners are used. We use Rapidly Exploring Random Trees (RRTs) from the sample based motion planning community to generate motion plans [54] since RRTs can handle various models, including those with differential constraints and dynamics.

Motion Plan Generation Details

Planning occurs in geometric models or those that include controls. Models that do not include controls sample target waypoints in the state space and collision check along the interpolated line between waypoints. Models with controls define the steering function for propagating the planner towards sampled target waypoints. The steering function includes a definition of ordinary differential equations, a maximum propagation step size, and set range for the control duration. At each time step the robot state propagates by numerically integrating the equations using Runge Kutta. Implementation details for our tested models can be found in Section **??**.

Algorithm 1 Robust Plan Generation

```
1: [p , planResult] = generatePlan(m);
2: if planResult == success then
3:     globalPlan = savePlan(p, globalPlan);
4:     tm = translateToModel(p, highest);
5:     [planCheck, b4repair, after] = propagateWhileValid(tm, highest)
6:     if planCheck == infeasible then
7:         m = MODELSELECTOR(globalPlan, b4repair, after);
8:         [p, planResult] = multiTreeReplan(globalPlan, b4repair, after, m);
9:         Goto line 2
10:    else
11:        executeResult = sendToRobot(globalPlan);
12:    end if
13: else
14:    Failed to find plan.
15:    Goto line 7
16: end if
```

The collision checker is the same for all models. A state is determined valid by collision checking the appropriate robot model mesh against any corresponding environment obstacles. This is determined by using an occupancy grid where obstacles are inflated by the robot's radius. All obstacles which intersect this state in the occupancy grid are collision checked agains a robot mesh defined by its model. For example, $[x, y, \theta]$ is collision checked in the x, y, and θ directions for each overlapping obstacle mesh. Implementation details for our selected collision checker can be found in Section 5.3.2.

Our system uses an environment map to generate a plan from start to goal in the lowest applicable fidelity model. We start by planning in a default model space, typically [x,y] (Algorithm 1, line 1). This produces a plan in the form of a series of waypoints, such as that shown in Figure 4.3 (a).

4.1.2 Feasibility Detection

The plan is then checked in the highest fidelity model to determine parts that might need repairing (Algorithm 1, line 6). Feasibility checking is inexpensive because it does not require a new search. The lower plan checks in the higher model along the previous plan's waypoints. For example a lower model considers only geometric constraints while a higher model considers dynamic constraints. Even though the highest fidelity model is complex, checking a single plan does not incur much computation time. If the plan is feasible, that is there are no detected collisions, it is sent to the robot for execution (Algorithm 1, Line 11). If the plan is not feasible, the infeasible plan segment is sent to the model selector. In Figure 4.3 (b), an infeasibility is detected between waypoints four and five.

Checking in the highest model requires a translation step to match the configuration inputs of the higher model, and collision checking between the robot and obstacles in the environment along the plan.

Plan Translation

Plan translation is necessary for checking paths in different models. In our work, there is a distinction between translating between models and translating between plans. As discussed

previously in Section 3.1.2, the definition of how a hierarchy is formed includes a lossless translation function which describes translation to higher fidelity models. Unfortunately, this does not include all information necessary when checking between plans. Models that include passive states are re-propagated from the start along the path. Examples of passive states include an additional robot trailer position, and indexing in the environment by time. Another example is modeling non-infinite accelerations in higher models. The robot needs to propagate the actual velocities since it may not reach the desired velocity by the next waypoint.

Another consideration with plan translation is how to improve the checker's fidelity to account for what the controller does. For example, our [x y] model requires directional thetas to be added for higher fidelity spaces where θ is modeled. Additionally, the plan is translated to more closely match the controller by augmenting angular velocity to stay along the path. This forces the feasibility detector to follow a path that most closely resembles the robot's actual path during execution. Implications of the differences between how explicitly the feasibility checker matches the underlying robot controller are discussed in Section 4.3.1.

Note that the model space is separate from the underlying robot controller. To illustrate this separation, imagine people racing cars on switchbacks. They are able to apply a controlled skid to take tight turns at high speeds by estimating an internal momentum model. The internal model they use for their driving plan is separate from the underlying application of braking, hitting the gas pedal, and steering necessary for controlling the car.

Feasibility Checking Details

The feasibility detector retrieves the current globally maintained path and translates it into the highest model. The translation step adds necessary states then propagates from the start to end to update passive states without collision checking. It does so by propagating and collision checking using the global path as a reference path from which the next target point is retrieved. The feasibility detector propagates between waypoint guides by using the planner's ODE integration functions while waypoint follows the path like the underlying robot's controller. This includes the same logic that decides when waypoints have been achieved and adjusting controls to remain on the path similar to the real robot.

We note that plan generation for the highest model uses the same propagation functions as used during feasibility detection. The main difference is that when propagating for feasibility detection the next waypoint(s) guide what the waypoint follower does; but during planning the next waypoint is not known so the level of fidelity is different even when planning in the highest model.

It is important for the feasibility detector and the underlying controller to match. Mismatches create false positive and false negative rates that directly impact switching results. This is further mentioned in Section 4.3.1. Initial nominal padding amounts used by the checker can help influence these mismatch rates. This is further discussed in Section 5.4.1.

4.1.3 Model Selection

As an example of how the model selector works, assume that an infeasibility is detected between waypoints 4 and 5 of Figure 4.3 (b). The model space used for the plan segment (between

waypoint four and five) that needs repair (initially the [x,y] model) is the first node to search from in the model graph. The model selector determines the appropriate model to re-plan in. Our model selection process (Algorithm 2) uses Breadth First Search to explore the model graph. As an example of how the model selector works, assume that it starts with the first higher model of the M_1 = [x,y] model which is the M_2 = [x,y,θ] model. For each model, we first test the infeasible plan segment in that new model space. The previous [x,y] plan segment is tested in this model (Algorithm 2, line 8). If the plan succeeds, we assume that the [x,y,θ] model does not accurately capture the infeasibility. We then choose the next higher model for [x,y], which is M_3 = [x,y,z], and again test the previous plan in this higher fidelity model. If the plan again succeeds, we then try the M_4 = [x,y,z,θ] model. If testing the previous plan finally does *not* succeed, we assume this model captures the space of the infeasibility and has information not present in the original model used to generate the plan. It is then selected as the model to use for re-planning. In this approach it is possible to produce a final plan that skips between model tree levels when choosing the next model (in this example, we skipped from [x,y] to [x,y,z,θ]).

It is important to note again that the re-plan model contains information *relevant* to the detected infeasibility. We define a more informative model as one that contains information useful for re-planning in the current environment context. This means that not all higher fidelity models are useful for re-planning. For example, a higher fidelity model that reasons about velocities and slip constraints contains more information than a model that represents the z-dimension of environment obstacles. The model which contains z is more applicable to a world with a tall robot which must navigate overhangs even though it does not have the most information.

Algorithm 2 The model selector does a Breadth First Search by testing old infeasible plan segments in higher fidelity models until the old plan fails. If the old plan fails, this indicates the model contains information that may be relevant to the infeasibility and it is chosen for re-planning.

```
1:  function MODELSELECTOR(p, b4failIndex, afterfailIndex)
2:      mLast = findModelFor(p.getNode(b4failIndex));
3:      m = mLast;
4:      p = resizePlan(p.getNode(b4failIndex), p.getNode(afterfailIndex));
5:      setUsedModel(m);                                  ▷ Last model used becomes root node.
6:      m = getNewModelBFS();
7:      tm = translateToModel(p,m);
8:      planResult = propagateWhileValid(tm, m);          ▷ Does collision checking.
9:      if planResult == success then
10:         Goto line 5
11:     end if
12:     return m
13: end function
```

The efficacy of this approach versus an execution time recovery approach depends on the expected failure rate, expected severity of the failure, and uncertainty in execution knowledge of the environment. Since information is available to the robot at plan-time, model switching can occur before execution.

Model Selection Details

The model selector also has a translation step for elevating the partial path being checked to the selected re-plan model. The translation steps are the same as the feasibility checker. Model

configuration variables are added as necessary and then a propagation step translates passive variables from the start waypoint until the waypoint after the detected collision. The propagation function changes based on the model that is being checked. Then the model selector collision checks between the two waypoints where the collision was detected by the highest model feasibility checker. We do not collision check earlier along the path because it is possible for models lower than the highest model to prematurely detect a collision and the repair area should not change from what the feasibility detector determined. This is because if the model being checking in is not the highest model, then it is assumed to not be the *best* representation of how the actual robot moves in the environment. So even if a lower model (say $XY\theta$) deviates from the lower reference path (an XY portion) and this is before where the feasibility checker detected a collision, the repair should not move since this is not the most accurate evaluation of the robot moving in the world. The purpose of the model selection is to choose the lowest applicable model to cover the collision detected during feasibility detection. Therefore, it checks the path along models that do not move exactly as the robot does, but are hopefully still beneficial in indicating that they contain some information to circumvent the collision.

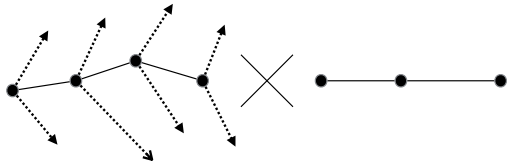
4.1.4 Intermediate Goals and Multi-Tree Re-Planning

(a) Example global path.

(b) Infeasibility exists between waypoints four and five.

(c) Waypoints five, six, and seven form a goal set to plan towards. They are probabilistically sampled using an inverse distance metric from the infeasibility.

(d) Waypoints one, two, three, and four are the starts of multiple re-plan trees. A tree is grown each cycle based on probabilistically sampling the starts using an inverse distance metric from the infeasibility.



(e) Re-planning concurrently plans from multiple start trees towards a set of intermediate goals.

Figure 4.3: Localizing around the infeasible area by generating multiple re-plan trees towards intermediate goals, using the new selected model, rather than re-planning to the original goal.

Infeasible areas vary in size and closeness to previous plan waypoints. Therefore, our re-planning stage accounts for planning from multiple starts and towards multiple goals that vary

in distance from the infeasibility. To accomplish this, waypoints along the globally maintained plan, Figure 4.3 (a), before the infeasibility, are considered the starts of re-plan trees. Similarly, remaining waypoint nodes after the infeasibility are considered intermediate goals.

The remaining waypoints along the path, Figure 4.3 (c), form a set of intermediate goals. We sample from this multi-goal set based on an inverse distance metric from the previous infeasibility to encourage returning to re-using the previous path. This allows the algorithm to concurrently plan to all remaining intermediate goals, and be more efficient with path re-use. For example, in Figure 4.4 a goal set is created for all unachieved waypoints (5 through 7). The remaining nodes after the infeasible area create a goal set. The planner then re-plans to this goal set and finds waypoint 6 is successful. If we successfully plan to an intermediate goal we can switch back to using the previous path for the remainder. Using waypoints as a cache was also done in work by [13]. We expand this for multi-fidelity nodes and plan reuse.

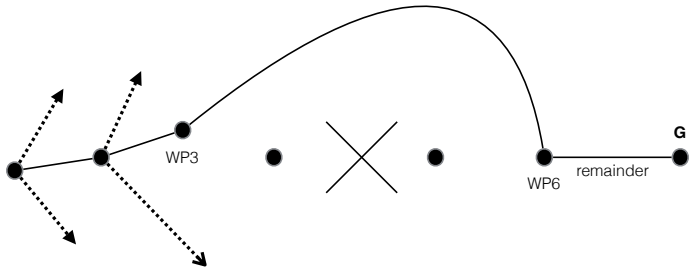


Figure 4.4: Re-planning from waypoint 3 to 6 and then resuming the original path.

We also plan from all previous waypoints for probabilistic completeness guarantees, shown in Section 4.5.1. In Figure 4.3 (d), we generate start trees using all previous waypoints since it may not be possible to find a plan from the waypoint start closest to the infeasibility to any goal. Planning from previous waypoints could be done sequentially but we create a multi-tree set that concurrently generates trees from all previous start waypoints. The growth of each tree towards the goal set, Figure 4.3 (e), also uses a heuristic weighting metric to probabilistically sample what tree to expand next. Further analysis for probabilistically weighting the start trees and goal set are discussed in Section 4.3.2.

The globally maintained plan contains nodes of multiple model types. Consequently, it may not be possible to re-plan in the selected model for all previous nodes along the global path. For example if the re-plan model is [x y z], we would want to maintain previous portions of the path planned in [x y θ]. This is because that model was found to be useful for some previous portion of the path and it makes sense to use at least that high fidelity of model when re-planning along the path. Since all models have at least one common higher model (LCM) (such as M4 in Figure 3.3), previous waypoints along the global path are elevated to this common model before re-planning occurs. In this example, the least common higher model (LCM) between [x y z] and [x y θ] is [x y z θ]. Additionally, the remaining intermediate goals after the infeasible area are translated to the appropriate LCM between itself and the currently selected model when set as possible goals.

We do a further refinement when re-planning from previous waypoints to ensure that lower models do not overwrite previous higher model plans. In addition to finding the least common

higher model between the current waypoint and the model selected for re-planning, the LCM is compared for all waypoints before this waypoint. The LCM for all previous waypoints, this waypoint, and the selected model is then the model for re-planning.

The first re-plan tree that connects to an intermediate goal creates a partial plan which is merged back into the global plan. Therefore, this re-planning phase may create shortcuts connecting start trees earlier in the path to later intermediate goals. The overall process repeats until the feasibility detection stage determines the full global plan is executable and sends it on to the robot.

Multi-Tree Re-Plan Details

As stated before, to know which model to translate the start and respective goal set into we need to find the least common higher model (LCM) in our graph between the start node of the planning tree in the global path, the model selected to re-plan in, and all previous waypoints before this current waypoint. Once the waypoints are translated the new start and goal(s) are checked to see if they still remain valid. Invalid start or goal waypoints are skipped. This also means it is possible that there are no valid re-plan trees. If this is the case, our implementation goes back to the model selector to determine the next higher re-plan model.

The multi tree re-planning planner is a wrapper around the RRT planner that at each time step, while an intermediate goal has not been achieved, selects the next tree to expand based on a tree expansion heuristic, Section 4.3.2. It keeps expanding trees concurrently until a solution is found. Then it takes this partial path and re-merges it into the maintained global path.

Algorithm 3 A global plan saves the proper model with each plan node. Partial paths are merged back into the global plan.

```
1: function SAVEPLAN(p, globalPlan)
2:     if globalPlan!=empty then
3:         planPrepend = findPartial(p.start(), p.end(), globalPlan)
4:         planRemainder = findPartial(p.end(), globalPlan);
5:         globalPlan = planPrepend + p + planRemainder;
6:         addConnectionPoint(planPrepend.end(), p.start());
7:         addConnectionPoint(p.end(), planRemainder.start());
8:     else
9:         globalPlan = p;
10:     end if
11:     return globalPlan
12: end function
```

The initial plan that is generated is saved to a global path. The model type is maintained to know what model generated the plan for each node. When merging the partial path (Algorithm 3), the global path up until the waypoint node that matches the start index of the re-planned path is the prepend path. The re-planned path's start node is added to the end of the prepended path at a connection node. This connects the original path to the start of the partially re-planned path. Similarly, if the partially re-planned path connects to an intermediate goal the original path remainder after this node is retained. The new partial path is then merged where the node that matched the intermediate goal is the connection to the path remainder. The connection nodes also update configuration values to match between different model types.

4.2 Plan-Time Model Switching and Uncertainty Models

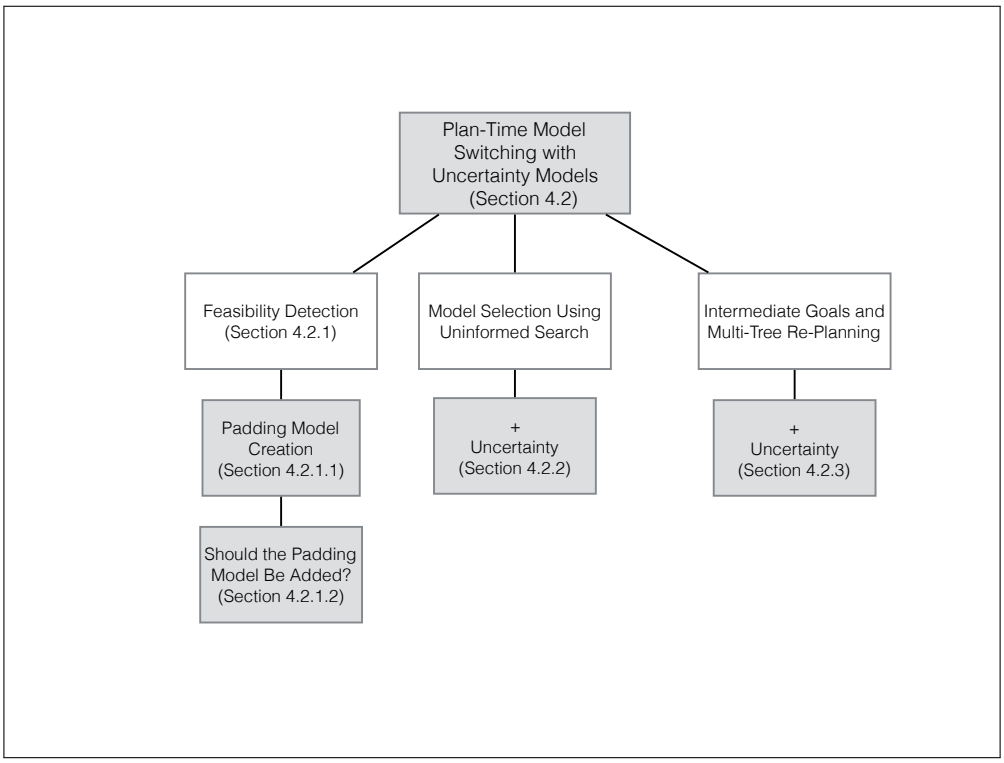


Figure 4.5: Approach with Added Uncertainty Roadmap.

Uncertainty is a good metric for capturing information that is difficult to represent. One example is information that is hard to capture in a higher fidelity model such as complex terrain interaction features that cause the robot to veer from the controller's trajectory. This may require information about a vehicles attitude, chassis configuration, or slip parameters that are not available. Rather than leave the information completely un-modeled, uncertainty can be used to represent it more generally.

Uncertainty may also arise from inadequate sensing. Even if information becomes available later it may be impossible to detect it completely. For example, lifting a pallet and transporting it across a constrained space might require detailed information about the pallet's shape and position on the robot's forks. The pallet's position and shape might be hard to detect, especially if the pallet forks do not include a sensor to determine this. Uncertainty can be used to cover possible pallet locations on the robot's forks.

Our approach represents uncertainty by increasing the robot's footprint with uniform padding, that is, all the dimensions of the robot are increased by the same amount of padding. While the padding is uniform, in our approach the amount of padding is not fixed. Fixed padding amounts may be too high, such as in constrained passageways, or need to vary. For example, a navigating robot might include footprint padding to increase its distance from obstacles and ensure more reliable execution. Navigating past tight doorways may need less padding than when navigating near people that require a higher safety buffer. Padding specific to doorways may be inaccurate if

door sizes change. Therefore, our solution automatically determines the least amount of padding necessary to be successful. We choose the minimum padding amount necessary to circumvent the infeasibility as to not create unnecessarily long paths and minimize time spent planning in higher models.

4.2.1 Feasibility Detection

The highest model feasibility detector, discussed previously in our plan-time model switching algorithm (Section 4.1), starts the process for determining the model padding amount. If the detector determines an infeasibility, the algorithm then decides a padding amount to add, and if a new model that includes that padding amount should be added to the model hierarchy.

Padding Model Creation

The padding amount to add to the lower model is determined based on where the detected higher level impediment is found. This is a minimum amount so that the padded robot can fit through constrained spaces. First the feasibility checker finds a collision point. Then the minimum padding amount necessary to cover this collision point in the lower model is found by comparing the distance between the robot footprint and the collision point along the lower path.

For example, the reference path is the solid path shown in Figure 4.6. The dotted path is the robot path propagated and checked in the highest model which finds a collision point indicated by the star. The minimum distance between the starred collision point and the robot's footprint along the original reference path gives the padding amount.

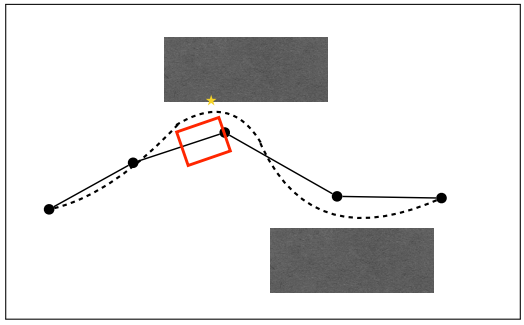


Figure 4.6: Reference path versus checked path as compared to a detected collision point.

The problem is that if too much padding is added to the footprint the robot may not fit between obstacles. Therefore, we want to find the maximum padding that can be added and still fit through the space. We do this by using Voronoi diagrams. We add a further refinement to be more conservative with the padding amount by using a Voronoi diagram. The environment's Voronoi diagram is used to find the Voronoi edge closest to the collision point. This edge may be closer to the collision point than the planned robot path. To test this, the original robot footprint is checked along the Voronoi edge to determine the minimum padding amount necessary to cover the collision point. Then this amount is compared to the previous reference path padding amount and the overall minimum padding amount is used. For example, in Figure 4.7 the dark

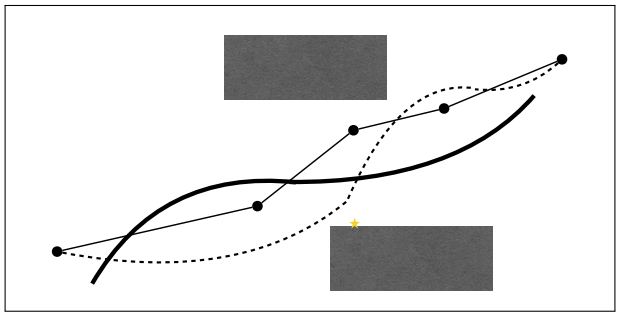


Figure 4.7: The dark line is an approximation of the Voronoi edge between obsacles as compared to a refence path and checked path.

black line approximates the Voronoi edge equidistant between the two obstacles. The line with waypoints represents the original reference path and the dotted line is the path checked which approximates the robot's execution. The distance between the lower star, where the checked path finds a collision, and the Voronoi edge is less than the distance to the reference path. Therefore, when using the robot's footprint to find the minimum padding necessary to cover the collision point we would use the result from comparing the footprint along the Voronoi edge rather than the reference path. Using the padding found through the Voronoi diagram will ensure that the padded robot footprint can fit between obstacles, at least at the point of the collision.

Should the Padding Model Be Added?

We check two cases for deciding whether or not to use the uncertainty model for re-planning.

If the robot footprint already intersects the higher model collision point then no additional padding is necessary and an uncertainty model is not added. This case occurs when the collision checked obstacles change in the higher model (as with time-indexed obstacles) and checking the robot along the lower reference path is found to already intersect with the collision point. In this case, the distance is zero. This is an example of an environment that would not benefit from added footprint padding, but could use models with variable uncertainty in the time dimension (which are beyond the scope of this thesis).

The second case is when the footprint padding amount to add is so large that it would not be possible to find a path. This can be determined without having to plan by using a Voronoi diagram. First we use the Voronoi Diagram edges to find the padding distance between obstacles for different homotopy classes. A percentage of this value is set as the max traversable value. Then for each edge we check if the re-planning padding amount exceeds the traversable amount for that edge and mark if it is traversable. For all traversable edges, the algorithm finds if a path exists from the start to end edge that fits this model's padding amount. If there is not, we assume no path can be found forcing a switch to re-planning in a higher fidelity model.

Model Padding Details

There are further details considered when selecting the padding amount. The first is to only compare the robot footprint distance between waypoints where the collision occurred. This is

because the global reference path is multi-model and we only want to check the failed model section. Secondly, it is possible for the new re-planned portion of the path to find a collision during feasibility checking. If this occurs repeatedly, the minimum padding amount may not be enough to avoid the infeasibility and a planning and checking loop can occur. To resolve this issue, we track the Voronoi edge last used for deciding additional padding with this model. The Voronoi edge is not used to find the padding amount if the same edge was previously used.

4.2.2 Model Selection with Uncertainty

With the addition of padding models, the model selector now either selects a model with more uncertainty or a higher model with more configuration variables and unchanged uncertainty. The model selector still uses Breadth First Search through the model graph, but switches to Depth First Search within the group of padding models. This is because, typically, lower fidelity models with increased padding take less planning time than higher fidelity models, even if they have less padding. The padding model is assumed cheaper for planning so the model selector conservatively checks all padding models for a particular base model before going back to Breadth First Search. When a padding model is selected, the planner now uses the selected padding amount. This padding creates re-planned partial paths which either shift waypoints further from obstacles in the repair region or help circumvent environment areas where the robot's dynamics cause deviation from following the lower path.

As an example, Figure 4.8 shows a model graph with four base models and five padding models which were added during the planning process. The padding models for a common base model are ordered by increasing uncertainty. These models comprise an uncertainty group for that base model. The base model $[x, y]$ has three padding models with padding values 0.12, 0.18, and 0.2, and two higher fidelity configuration models $[x, y, z]$ and $[x, y, \theta]$. The base model $[x, y, \theta]$ has two padding models with padding values 0.1, 0.22, and one higher fidelity configuration model $[x, y, z, \theta]$. The model graph is numbered based on the model selection order which is Breadth First Search through the base models and Depth First Search through the padding models.

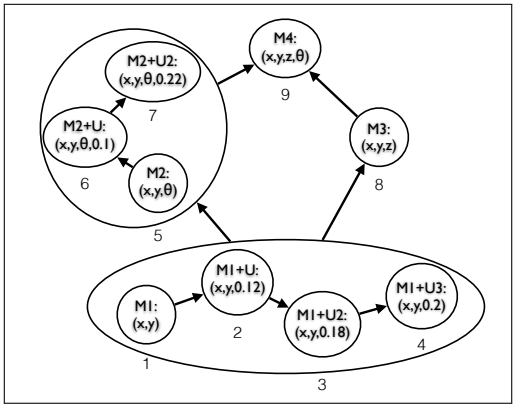


Figure 4.8: Model graph with various padding amounts that are numbered to show the model selection order.

4.2.3 Intermediate Goals and Multi-Tree Re-Planning with Padding Models

As stated previously in Section 4.1.4, it is necessary to determine the least common higher model (LCM) between the selected re-plan model and previous model nodes when there is no lossless translation between them. Padding models introduce new cases to consider when deciding the LCM. The model selector could select a model with the same configuration variables as the previous path model but a different padding amount. For example in Figure 4.8, suppose the model selector decides to re-plan with model $[x, y, 0.2]$ and the rest of the path has been planned with model $[x, y]$. In that case, the least common model should be the one with higher uncertainty, [x, y, 0.2]. Translation between models with different padding amounts is straight forward because a change in uncertainty is a change in the footprint padding amount that is collision checked. Now consider the case where the model selector could determine a model that is an indirect higher model $[x, y, \theta, 0.1]$ of the previously planned model $[x, y, 0.12]$. Even though the padding amounts are different, the LCM would then be the higher configuration model with determined padding $[x, y, \theta, 0.1]$. Lastly, in the case where there are two models that are not comparable (in configuration variables) and different padding amounts, such as $[x, y, \theta, 0.22]$ selected for re-planning over previous planned $[x, y, z]$, the least common higher model is the LCM without additional padding $[x, y, z, \theta]$.

The start tree states and goal sets are also validated after translation into the LCM, as mentioned in Section 4.1.4. This is because waypoints may get translated into models that cause invalid states. An example of this occurs when a model with some added padding now causes the footprint at this waypoint to intersect an obstacle. Goal waypoints are also checked, but without their passive variables because these values can not be predicted without having planned the repaired path. Furthermore, it is possible for different goals to be valid for different trees since they can be elevated to different padding models. If the start state is invalid or there are no valid goals, the tree is not used for re-planning.

4.3 Plan-Time Model Switching Discussion and Enhancements

In this section we describe two additional considerations for our plan-time algorithm. The first describes a statistic for evaluating how well the feasibility checker matches the robot's execution controller and how this statistic can effect our results. The second describes a re-planning heuristic that effects overall planning times.

4.3.1 False Positive and False Negative Rates

Our work assumes the highest model matches the real world as closely as possible, if this does not occur it increases the presence of false positive and false negative results. A false positive result occurs when the highest model used to check the path detects a collision, but the robot would have executed the path successfully. For example, this can occur in a swinging doors world where door states are discretized with each time-step, but in reality door swing is continuous. A false negative result occurs when the highest model checker does not detect a collision but the robot

encounters an execution failure. This can occur when ineffectively modeling dynamics, such as mass, which could increase or decrease a swinging trailer as compared to the highest model. False positives and negatives have adverse effects. False positives increase planning times since re-planning may occur unnecessarily, and false negatives decrease overall success rates of our approach.

For successful model implementation we seek a balance between successful execution rates and shorter planning times. To achieve this it is important to balance between the false positive and false negative rates that can occur between inconsistencies with the planner's path checker and the robot's underlying controller. We found that as a first step it is possible to consider checking paths with a nominal footprint padding value and also plan with a different initial nominal padding value. This is described further in Section 5.4.1.

4.3.2 Multi-Tree Re-Plan Weighting

The re-planning phase creates partial repair paths from and towards reference waypoints planned using a lower model. Sometimes lower model reference paths are generated in areas difficult for higher models to find plans causing longer re-plan times. This is why a heuristic is used for probabilistically selecting which re-planning tree to expand next and for weighting an intermediate goal to expand towards.

The heuristic we use for tree expansion accounts for three things. It favors tree expansion for waypoints closer to the collision point for path re-use, flattens the tree selection distribution to be equal as planning times increase, and preferences start nodes further away from obstacles. Re-planning from and towards waypoints closer to the collision point localizes the repair and encourages the re-use of the previous path. Therefore, the first part of the heuristic contains an inverse distance weighting, $1/(1 + distance)$, where the distance is to the waypoint after the detected collision. As the distance from the collision increases the weight decreases. Simultaneously, it is not known how close the collision is to these re-planning waypoint guides, and favoring tree expansion too close to the detected collision area causes the search to stall. This is alleviated by re-planning from multiple start trees towards multiple intermediate goals and with the second addition to the heuristic. The planning time is added to balance the preference of keeping the repair small while minimizing stall cases. As planning time increases the expansion probabilities flatten to have an equal weight. Planning time is added to the denominator, $1/(1 + (distance/planTime^2))$ decreasing the effect of the distance value over time as $1/(1 + epsilon) = 1$. Lastly, the third refinement favors start nodes further from obstacles by including a logistic weighting function as shown in Figure 4.9. Since RRTs increase node expansion in constrained areas, re-planning can stall when any previous waypoints are too close to obstacles, and not just those closer to where the impediment was detected.

The logistic sigmoid function groups lower distance values closer to zero as they approach negative infinity and higher distance values closer to the max as they approach positive infinity. As the robot moves in the world there is a distance that is too close for the robot to be able to effectively re-plan from an obstacle, and anything less than this distance is equally discouraged. Likewise, once the robot is a certain distance away from an obstacle any higher distance values are equally beneficial. Figure 4.9 demonstrates this effect by weighting the obstacle distance where anything less than 20 cm is considered too close (the lower 10%) and anything greater

than a half meter is equally good (the upper 90%). This logistic function is defined as: $1/(1 + e^{-15*(x-0.35)})$.

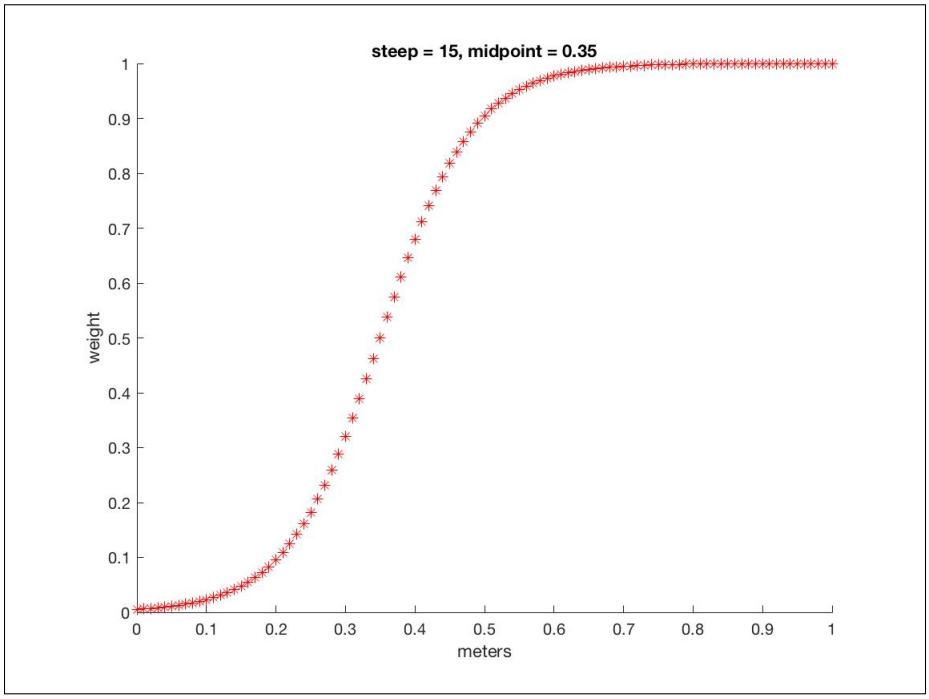


Figure 4.9: Logistic curve for obstacle distance weighting. Anything less than 20 cm is weighted in the lower 10%, and anything greater than half meter is weighted in the upper 90%.

4.4 Plan-Time Model Switching Summary

In summary, our approach assumes we are given a discrete model collection. Using these models we construct a hierarchical graph. Lower fidelity model abstractions lead to higher model supersets that better approximate the real world. The model organization considerations are described in Section 3.2.

Detecting task impediment focuses on the decision of *when* to switch to a different model. This occurs when the feasibility checker checks the reference path in the highest given model while waypoint following similar to the robot's execution controller, Section 4.1.2. Our approach automatically determines *what* padding amount to add to the robot footprint for additional uncertainty models, Section 4.2.1, and leverages the model hierarchy to choose *what* model to re-plan in during the model selection process, Section 4.1.3 and Section 4.2.2. The path is repaired by planning in the selected model from multiple start trees toward intermediate goal sets, Section 4.1.4. This process repeats until a determined execution feasible multi-model path is sent to the robot for execution.

We highlight some initial assumptions that we have addressed.

- Models of varying fidelity from simple to complex are given.

- Lower fidelity models can be translated into higher fidelity models assuming a lossless translation.
- Models with added uncertainty vary the amount of robot footprint padding.

4.5 Formalization of the Approach

We begin by providing a list of assumptions for our approach that we will refer to in the paragraphs that follow.

1. The approach is for any *domain* that does motion planning with a rigid-body robot through a space of known obstacles.
2. The domain includes a definition of a single highest-fidelity planning model.
3. The domain can be decomposed into varying fidelity planning models which include *lower* and *higher* fidelity models.
4. Each model defines a manifold of configuration space and a state transition function which defines the propagation between robot configuration states.
5. These models fit into a hierarchy based on a partial ordering of the model's fidelity.
6. The domain defines a translation for the ability to propagate lower fidelity plans in higher fidelity models.
7. This translation defines the update of lower fidelity state into higher fidelity state, which includes: actively controlled state variables and passive state variables (that depend on the history of this model's plan).
8. The translation sets active variables to default values and updates passive variables through path translation.
9. A control law exists for trajectory following plans in the highest possible model.
10. The control law for feasibility detection matches the one used by the robot during execution.

We first define the class of problems for our approach. We believe this approach would be useful for any rigid body robot that is motion planning through a space of known obstacles. This is a class of robot systems where it is possible to decompose the dimensionality of the planning space. These decompositions range in fidelity from lower to higher where fidelity is defined in Chapter 3.1.2. Each of these separate decompositions can be used to generate a path. We refer to each decomposition as a model. We require a discretization of the motion planning space into at least f fidelity models, where f is at least 2, and includes a description of the highest possible model. If the problem class allows the discretization of the highest model into varying fidelity models, where $f > 2$, then our approach is more beneficial.

We start by describing more formal examples of lower and higher dimensional models as required by the class of problems. Then we describe the necessary inputs and outputs for the ability to propagate lower models in higher model spaces. Once we have established what is necessary for this propagation we can describe the use of the feasibility detector and model

selection steps. Then we go on to define completeness guarantee for our framework with respect to the highest model.

Our approach defines parts of a model which are the basic components necessary for motion planning, [55]. This includes a definition of configuration space, which includes the robot and environment state, and a transition function. The configuration space contains an obstacle space which describes the occupied parts of the environment, C_{occup}, and a collision free space which describe the feasible configurations of the robot state C_{free}. A continuous path is generated from an initial start and goal configuration in C_{free} where we require a transition function which defines the propagation between configuration states of the robot. The transition function t takes a given action a and a current feasible state $c \in C_{free}$ to a next feasible state $c' \in C_{free}$, $c' = t(c, a)$. The transition function describes either a discrete-time state transition or transition on a continuous stat space $\dot{x}$ which may yield a velocity but can look similar to a discrete transition. For example, this may be defined using optional control inputs for transitioning by integrating velocity between states $\dot{x} = t(c, a) = f(x(t), u(t), t)$, and then determining it is in C_{free} or C_{occup}.

Formally defining our models requires understanding the robot's configuration space which we refer to as the topological state used in motion planning, [55]. We expand on our model definition, Chapter 3.1, of the configuration space. The configuration space defines unique configuration variables of the robot at any point in time as defined by the vector $\vec{q}$. This configuration space is a manifold [23]. A manifold is defined from a topological space which requires a notion of dimensionality and contains a homeomorphic property which implies reflective, transitive and symmetry properties in the space. From Lavalle's book, [55], a topological space $S \subseteq R^n$ is a manifold if: 1) for every state $x \in S$ an open set exists such that 2) x is in the open set and 3) the open set is homeomorphic to a real vector space R^n, where n is the fixed dimension of the manifold, S. Therefore, at every point a manifold behaves like a surface without crossing points or self-intersections. We note that this manifold describes robot transformations in the absence of configurations which cause the robot to collide with obstacles or itself, S_{free}. The dimensionality on a single manifold does not change. Therefore, the robot's state for different fidelity models represent different manifolds.

The definition of a model also includes constraints. The two constraint types are static constraints on the state or constraints on the transition function. For example, there can be a constraint on the allowable velocity at every point in the configuration space. Constraining the velocity at particular points creates a tangential space to the full configuration space defining a vector space. By discussing the tangent space, we can discuss differentiable manifolds and describe how the tangent space can also be described as a manifold. Therefore, the motions the robot takes can be thought of as actions between states represented as a differentiable mapping [23]. It is also possible to constrain the transition function which affects the reachability between configuration states. For example, possible constraints on the system can occur with the addition of independent links for a robot body. The full range of motion for a robot may not be possible depending on how independent links of subsystems are joined. Additional links create additional constraints on the system for higher dimensional models. Constraints are expressed as a function $g(q, \dot{q})$, [55].

For uncertainty models the dimensionality of the configuration space does not change but the volume of the free configuration space, C_{free}, is reduced due to a larger robot footprint.

This could be thought of as a volume-reducing collision avoidance constraint which represents a subset of the base model's configuration manifold. Therefore, uncertainty models can also represent configuration state with a manifold.

Decomposing the space breaks the planning space into what we refer to as *lower* fidelity, M_l and *higher* fidelity models, M_h. One way of discretizing the model space is by defining new topological spaces or manifolds. For example, our robot model's state can include the real vector space manifold R^2 (state for XY) or the higher fidelity special euclidean group SE(2) which is defined by combining $R^2 \times SO(2)$ (state for $XY\theta$), the two dimensional real vector space and the yaw rotation group. The topology of a rigid body (the robot) does not change when translated or rotated. Therefore, the Cartesian product generates new manifolds from existing ones, [55]. For example, models that include velocity add a time dimension which creates a different manifold. An example model defines $[x, y]$ from R^2, heading θ from $SO(2)$ and velocity, then the robot configuration state is defined as a manifold: $S = R^2 \times SO(2) \times T$. As another example, including a trailer also creates a manifold with an additional trailer heading where: $S = R^2 \times SO(2) \times SO(2)$ and the volume of C_{free} is reduced to include the trailer footprint. Multiple independent subsystems can also be defined where multiple rigid bodies move independently. Each of these could describe a separate manifold of configurations such as the robot's base versus the robot's manipulator. The configuration spaces of each subsystem may also be combined by Cartesian products. Since the class of problems for our approach requires a discretization of the planning space, we believe that just as it is possible to create kinematic chains of multiple bodies that are allowed to move independently by combining their configuration spaces, it is possible to separate them. For a multi-subsystem robot this could mean planning the robot's base separately from its manipulator, or breaking up the full joint space of a robot arm to subsystems which just include the forearm with wrist and end effector, or just the end effector. If different subsystems of the robot represent different fidelity models, than combined subsystems lead to higher fidelity models. The model hierarchy would place separate subsystems along the same level of the graph and then combine into higher fidelity models where the highest model includes the complete multi-subsystem robot.

We formally define the translation function, as lossless in Chapter 3.1.2, from lower to higher fidelity models. This occurs by changing the dimensionality of lower fidelity variables which consist of two types. Some dimensions are actively controlled and do not depend on the history of the path $x_a \in x(t)$, and others (a second group) are passive variables, $x_p \in x(t)$ which depend on the previous lower model's path history (a function of this lower dimensional path). Examples of passive variables include the trailer heading for a passive trailer or the update of the time dimension. Both dimensions are added, $x' = a(x_a, x_p)$, to match the state for the higher model manifold's configuration space $q_h(x_h(t) \mid \vec{x'} \in x_h(t)$. First, the actively controlled variables are set using default values; second, the passive variables are set using path translation. We describe the change in configuration space for adding both variables and then the path translation for updating passive variables in the following paragraphs.

First the ability to translate between models requires a change in configuration space. This is the addition of configuration dimensions. For a lower fidelity model's configuration variables $(q_l(\vec{x_l}(t)) \mid x_l \in R^n)$ as defined for manifold $(x_l \in S_l, S_l \in M_l)$ and a higher fidelity model's configuration variables $(q_h(\vec{x_h}(t)) \mid x_h \in R^m)$ as defined for manifold $(x_h \in S_h, S_h \in M_h)$

where $n < m$, additional configuration variables (both active and passive) are added to state x_l as necessary for $n == m$ to match the state dimensionality x_h of higher model M_h. These additional dimensions are set to default values.

The second step is a path translation. The passive variables which depend on history information can not be properly set until path information is known from the lower model space. The passive variables are updated through a path translation which requires the use of the higher model's transition function for defined propagation between waypoints $\dot{x_h} = t(c, a)$, and a control law. Formally, this is a trajectory follower, where given a sufficiently smooth trajectory (reference path) $\bar{x} : [0, T] \rightarrow R^n$ finds controls $u(.)$ that steer the state $x(t), t \in [0, T]$ *approximately* along $\bar{x}$, as defined in [92]. Here *approximately* is defined as tracking a desired reference path, in our case originally generated in a lower dimensional $\bar{x_l} : [0, T] \rightarrow M_l$ elevated to the higher model, $\bar{x_{lh}} : [0, T]$, in a neighborhood of a point in the elevated model M_h. Therefore, the control law $u_h(t) = f(t, \bar{x_{lh}}(t), x_h(t), \dot{x_h}(t))$, is an input output map for path translation between models $M_l \rightarrow M_h; \bar{x_{lh}}(t) \rightarrow u_h(t)$ which takes an elevated lower dimensional tracking path, the current state information (elevated to the higher model's configuration space) and transition functions (defined propagation) in the higher model for following this lower model reference path. The control law outputs path following controls for simulating the robot along the trajectory in the higher model. This means that state configurations from a lower dimensional manifold S_l are used as guides, $c_l \in C_{free}$, where c_l now contains added configuration state for the higher dimensional manifold.

We have established the class of problems where there is a decomposition of the space into varying fidelity models that are *lower* and *higher* with translations between. We require that a transition function exists for propagation in each fidelity model. We also assume a control law exists for path following that propagates (using the transition function) elevated *lower* model reference paths in a higher fidelity model. Since we can always translate into the highest model, the robot's execution controller similarly assumes a control law exists which can follow a trajectory of waypoints generated from any underlying model translated into the highest model. This control law outputs controls for the robot to stay along the path including logic of knowing when the next target waypoint is achieved. This is the same trajectory following logic used in path translation. Therefore, the control law which exists for path following during plan-time also allows the robot to follow the path during execution. The definition is the same, $u_h(t) = f(t, \bar{x_{lh}}(t), x_h(t), \dot{x_h}(t))$, the transition function is an execution time-step in the environment where an output of controls keep the robot along the path.

We note there can be deviations from the trajectory that is being followed. The control law tracks a desired trajectory by following in a neighborhood of a points. The ability to reach a point may not be exact because these *future* waypoints are used as guides and the lower fidelity models do not match the robot's characteristics as well as the highest model. Deviations between a planned path and the control law's ability to closely track the path lead to the necessity of a feasibility detector described in our approach. Feasibility detection determines at plan-time where deviations may cause an infeasibility and these are the points where switching to a different model occurs.

The feasibility detector propagates $u_{h*}(t)$ and collision checks $c(x_{h*}(t))$ the path at each time-step , $d(u_{h*}(t), c(x_{h*}(t)), t)$, in the highest possible model which requires a translation

into the highest model space $x_l(t) \rightarrow x_{lh*}(t)$ such that passive variables are updated for path $\bar{x_{lh*}}(t)$. This detection does path translation as described, and follows the path using $u_{h*}(t) = f(t, \bar{x_{lh*}}(t), x_{h*}(t), \dot{x_{h*}}(t))$, where the configuration state and transition function is for the highest possible M_{h*} defined model of the domain. The feasibility detector and robot controller use the same control law for waypoint following. They take the same input values and include the same logic for knowing when waypoints are achieved. The feasibility detector effectively simulates the robot in the highest model (along a multi-fidelity model reference path translated into the highest fidelity level) and the robot's execution controller uses the control law to drive the robot to follow the same reference path. When the feasibility detector does not match exactly how the robot executes in the environment it either detects more collisions than occur in execution (is too conservative) or does not find collisions that occur during execution (is not conservative enough).

If the feasibility detector determines *when* a switch occurs, then the model selector determines *what* model to switch to. The model selection stage also translates a path into the model being checked $x_l(t) \rightarrow x_{lh}(t)$. This translation adds configuration state to match the higher model it is checking in and uses the higher model transition functions for updating passive variables at each time-step $\bar{x_{lh}}(t)$. The difference between model selection and feasibility detection is that the model selector does not always follow paths in the highest possible model $h*$. The model fidelity changes depending on what model the selector decides to check in. The model selector also requires a definition of the control law $u_h(t)$ for following lower model paths in a higher fidelity model. It is also possible to follow paths without collision checking which just translates necessary parts of the path into higher models without checking the validity of the configuration state. The model selector at each time-step of the path collision checks the validity of the robot configuration, in this specific model, between the waypoints where a feasibility was detected. Therefore it propagates for all t, but only collision checks for a partial part of the path, $m(u_h(t), c(x_h(t1 : t2)), t)$, where h can be of any model fidelity.

Now that we have discussed the formalities for translating between models, and inputs required for deciding when and what model is necessary to switch to for re-planning, we can show completeness guarantees for our approach. Assuming that a plan can be generated in the highest model (the straw man) then it is possible discuss probabilistic completeness with respect to the highest model.

4.5.1 Probabilistic Completeness Guarantees

We start by re-showing Algorithm 1 and 2 so they can be more easily referenced by the probabilistic completeness proof. The algorithms presented previously (1-3) form the basis of a probabilistically complete planning algorithm.

Definition:
For probabilistic completeness, the probability that the highest fidelity model from the hierarchy will find a solution approaches one as the number of states approaches ∞, [54]. To illustrate this, we must show in the worst case our algorithm generates a plan from the start state to the goal state in the highest fidelity model in finite time.

We assume a model hierarchy exists where every model has at least one common model of higher fidelity. Therefore, there exists a highest fidelity model that is higher than all models.

Algorithm 1 Robust plan generation algorithm repeated from earlier in the chapter.

```
[p , planResult] = generatePlan(m);
if planResult == success then
    globalPlan = savePlan(p, globalPlan);
    tm = translateToModel(p, highest);
    [planCheck, b4repair, after] = propagateWhileValid(tm, highest)
    if planCheck == infeasible then
        m = MODELSELECTOR(globalPlan, b4repair, after);
        [p, planResult] = multiTreeReplan(globalPlan, b4repair, after, m);
        Goto line 2
    else
        executeResult = sendToRobot(globalPlan);
    end if
else
    Failed to find plan.
    Goto line 7
end if
```

We also assume a transition functions defines transitions between states. For instance, steering functions for the control input has been found to maintain probabilistic completeness in RRT planning when there is a fixed time-step as discussed in [53].

Algorithm 2 The model selector repeated from earlier in the chapter.

```
1:  function MODELSELECTOR(p, b4failIndex, afterfailIndex)
2:      mLast = findModelFor(p.getNode(b4failIndex));
3:      m = mLast;
4:      p = resizePlan(p.getNode(b4failIndex), p.getNode(afterfailIndex));
5:      setUsedModel(m);                          ▷ Last model used becomes root node.
6:      m = getNewModelBFS();
7:      tm = translateToModel(p,m);
8:      planResult = propagateWhileValid(tm, m);  ▷ Does collision checking.
9:      if planResult == success then
10:         Goto line 5
11:     end if
12:     return m
13: end function
```

Show: N path waypoints are generated in the lowest model (M_l) (Algorithm 1, Line 1). A lower bound of $N-1$ edges are checked in the highest fidelity model during feasibility detection (M_h, $M_l < M_h$) (Algorithm 1, Line 5). If an infeasibility is found the model selected for re-planning always *increases* in fidelity.

The Worst Case: An infeasibility is detected (Algorithm 1, line 6) in the last edge. (Algorithm 2, afterFailIndex = N-1). If the next model selected, (M_n), fails the model check then all previous start waypoints are elevated to at least the fidelity of M_n, and new re-plan trees of this dimension are initialized. Once a re-planned path is found it is then included in the maintained global path. If the partially re-planned path repair (Algorithm 1, Line 8) continues to have detected infeasibilities, then by BFS (Algorithm 2, line 6), the re-planned path will continue to get elevated until reaching M_h. This is also true for uncertainty models with added padding. The next higher fidelity model (by BFS) is selected for switching to when the determined padding amount is too high to find a plan, or there are no valid re-plan trees. Also, if a plan is not found during multi-tree re-planning, the model search will still select a higher fidelity model (Algorithm 1, line 6).

If the highest model is used as the start of a re-planned tree then all previous waypoints along the path are also elevated to this higher model. This is so lower model paths can not re-plan over previous higher model paths. If the feasibility detector determines that a repaired edge planned in the highest model M_h is also infeasible, the algorithm will re-plan from start to the end in the highest model. Therefore, in the worst case all previous model edges are elevated to M_h during tree re-planning generating a tree from the start to the elevated goal set in M_h.

Chapter 5

Implementation for Experiments

In this chapter we describe implementation details specific to the models and platform used in our experiments. First, we introduce our simulation environment. Then we describe the robot controller for executing in the high fidelity Gazebo simulator. Next we list the differential drive robot models, and discuss collision checking. Then we describe how the differential drive robot models are used in planning and model translation. We then give details on how partial path repairs are merged back into the path maintained for robot execution. Following this, we describe how that reference path is propagated and what we mean by propagation, path checking and path translation. This includes a discussion for reducing the false positive and false negative rates that occur between the feasibility detector and execution controller. Finally, we include some implementation details for using a Voronoi diagram for the uncertainty model part of the approach.

5.1 Simulation Environment

For this work, the real world is represented by the Gazebo simulator (`gazebosim.org`) which models dynamics by simulating rigid-body physics. We simulate multiple hospital worlds which highlight environments with two kinds of obstacles. The first are environments that contain randomly placed gurneys. The other is an environment with an automatic swinging door which opens as the robot approaches. Figure 5.1 shows a close view of hospital gurneys, Figure 5.1 (a) and (b), and the swinging door, Figure 5.1 (c). The exact environments we simulate are described further in Chapter 6.

5.2 Robot Controller

The simulated robot follows the execution path using a pure pursuit controller. The pure pursuit controller implementation is described in [18]. Pure pursuit adjusts the angular velocity to stay along the path based on a lookahead distance, measured along the path arc. Our lookahead threshold is currently fixed to 0.5 meters, but could be adjusted to change with time as linear velocity varies. The same controller is used regardless of the multiple model types in our final robot path.

(a) World with gurneys. (b) Another world with gurneys. (c) World with automatic swinging door.

Figure 5.1: Worlds with gurneys and an automatic swinging door.

The controller adjusts linear velocities to be within acceleration limits. At each time-step, the linear velocity is constrained to be achievable from the previous velocity, given an acceleration limit and time-step duration. This acceleration limit may change for any parts of the paths generated using higher models that compute acceleration.

In order to index the robot position along the path, and note when the goal has been reached, the controller checks if it has reached a target waypoint. This is done by checking if the position of the robot is within a pre-defined range of the target or the robot has crossed a line perpendicular to the path at the current target waypoint. This can easily be computed as the sign of the dot product between the path heading and the robot-waypoint position difference.

It is important to note again that the feasibility detector used in our plan-time algorithm checks paths similar to how the robot controller follows paths during execution. This includes using the same checks for determining if the next waypoint has been passed, staying within acceleration limits, and adjusting the angular velocity through pure pursuit.

5.2.1 Robot Controller More Details

As stated previously, we implemented a pure pursuit controller for waypoint following our multi-model path. The controller takes the next target waypoint and velocities as input. The controller first adjusts the value for the target waypoint based on the lookahead distance (0.5). This is determined by first finding the closest perpendicular point between the robot and the path. If this point is less than the lookahead threshold a closer point is found by interpolating along the path. Therefore, the next point along the path that is within this lookahead distance, which can be closer than the provided waypoint target, becomes the new target the robot goes towards. The controller then determines the displacement of this target point from its current position. This is done by computing the distance and bearing heading from the robot's current position to the target waypoint. Using this information, the vehicle's curvature for steering towards this point is determined. The steering curvature adjusts the angular velocity to stay along the path. Again please refer to [18] for these specific equations.

The controller then sets output controls to the robot's wheels. Therefore, the desired linear and angular velocities for waypoint following are converted to left and right wheel velocities. These equations are found below. We include the wheel separation *ws* (0.34), as a multiplier for

the angular velocity, and the wheel radius, *wr* (0.11).

$$desiredLW = (desiredL - (desiredA * ws)/2.0)/wr \tag{5.1}$$

$$desiredRW = (desiredL + (desiredA * ws)/2.0)/wr \tag{5.2}$$

Our controller operates in discrete time-steps. The left and right wheel velocities are accelerated at the acceleration limit provided by the plan until they reach the desired velocities. These wheel velocities are maintained by using a PID controller for each wheel. Once the PID has updated the commanded velocity for one time-step of the PID loop it is sent as the commanded velocity for the robot's wheel joint.

The next target waypoint is achieved if the waypoint is within 0.1 meters of the current robot's x, y position and 0.09 radians of the robot's heading, or the waypoint has been crossed. For knowing whether a target waypoint has been crossed, we first create a line segment that crosses through the target waypoint at a 90 degree angle from the target heading. This segment is set to a size of 0.25, *range*. The target waypoint's x and y values are labelled as *xT* and *yT*, and the segment's end points are constructed using the following equations:

$$unitX = range * cosine(goalHeading + 90) \tag{5.3}$$

$$unitY = range * sine(goalHeading + 90) \tag{5.4}$$

$$newX1 = xT + unitX \tag{5.5}$$

$$newY1 = yT + unitY \tag{5.6}$$

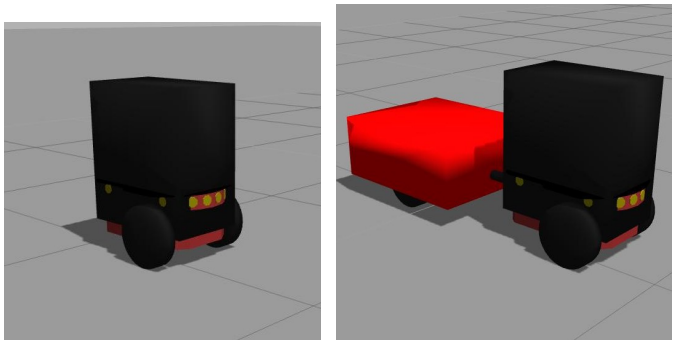
$$newX2 = xT - unitX \tag{5.7}$$

$$newY2 = yT - unitY \tag{5.8}$$

Then we determine if the robot's current position along the path has intersected this line segment. The robot's current position is set as the midpoint for a line segment of the same size as the target segment. Intersection is determined between these two line segments by using vector cross products where four cases are checked. The lines are either collinear, parallel and non-intersecting, intersect at a point, or not parallel but also do not intersect. These descriptions of the intersection of two line segments in three-space are found in [33]. If the line segments do not intersect, then it is possible to use the cross product and the sign of the determinant to know if the waypoint has been passed. This is the cross product between the vector perpendicular to the target waypoint and the line segment created from the current position of the robot. A change in sign of the determinant indicates that the robot's position is on the other side of the vector perpendicular to the target.

5.3 Robot Models

Our testing uses a simulated two-wheeled differential drive robot with a rear trailer attached to the robot's axle using a revolute joint. The robot geometry models are shown without (Figure 5.2 (a)) and with the trailer (Figure 5.2 (b))).



(a) Robot Model. (b) Robot Model with Trailer.

Figure 5.2: The robot model.

5.3.1 Model Hierarchy

We use seven different wheeled robot models in our testing. These comprise our base models. The unique variables used to describe the configuration of the robot at any point in time is described by the vector $\vec{q}$.

1. $\vec{q} = [x, y]$ = XY
   The first model is geometric with no control inputs.
2. $\vec{q} = [x, y, \theta]$ = $XY\theta$
   This model includes θ and constrains motions to s-curves. The linear velocity is held constant.
3. $\vec{q} = [x, y, \theta, \theta_{trailer}]$ = $XY\theta\theta_1$
   This model generates the same motions as the $XY\theta$ model , but also includes simulating the position for the trailer. The trailer's position is calculated using $\theta_{trailer}$.
4. $\vec{q} = [x, y, \theta, t, u_v, u_w]$ = $XY\theta V$
   This model samples the continuous space of linear and angular velocities (where u_v = sampled linear velocity control and u_w = sampled angular velocity control). Accelerations are assumed to be infinite.
5. $\vec{q} = [x, y, \theta, \theta_{trailer}, t, u_v, u_w]$ = $XY\theta\theta_1 V$
   This model is the same as model $XY\theta V$, but also models the trailer.
6. $\vec{q} = [x, y, \theta, t, v, w, u_a, u_\alpha]$ = $XY\theta VA$
   This model samples the continuous acceleration space (where u_a = sampled linear acceleration controls and u_α = sampled angular acceleration control), and now the linear (v) and angular (w) velocity variables become part of the state.
7. $\vec{q} = [x, y, \theta, \theta_{trailer}, t, v, w, u_a, u_\alpha]$ = $XY\theta\theta_1 VA$
   This model is the same as model $XY\theta VA$, but also models the trailer.

For our switching experiments, we use the transitive reduction of the full model graph, as shown in Figure 5.3. This model graph may be augmented to include models with additional padding for our uncertainty experiments, as described in Section 3.1.1.

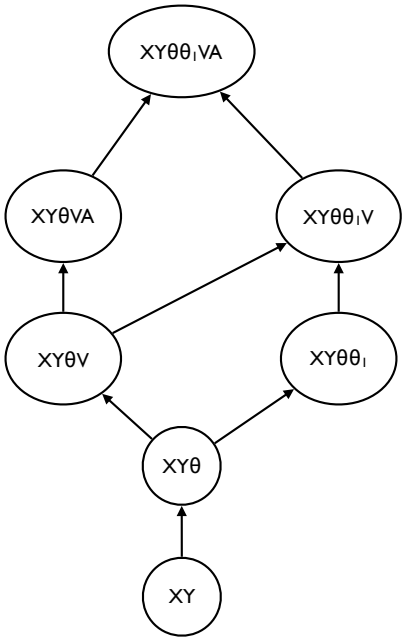


Figure 5.3: Transitive reduction of full model graph.

5.3.2 Collision Checker

The purpose of the collision checker is to determine if a given pose of a robot collides with objects in the environment, given the shape of the robot, footprint padding, and an optional trailer.

Before planning begins, collision meshes are created once and stored for each obstacle in the environment. This is done by generating point clouds for obstacles within the gazebo world. These clouds are then triangulated using a point-cloud library. As collision checking against point-clouds is expensive, an occupancy grid is precomputed for the environment. The grid contains obstacles with shapes inflated by the radius of a circumscribing circle around the robot plus padding. If the state being checked overlaps an occupied cell in the grid, it is then checked by the full point-cloud collision checker.

The collision checker is called in the RRT planner when determining valid states. We use the Flexible Collision Library (fcl) [73] for collision checking between a robot and environment. The collision checker is the same for all models, but the robot properties and environment being checked can change. When checking the validity of a robot state we first decide which configuration variables to use based on our model. This includes the trailer, time indexed values, and directional values such as: x, y, and θ. If the trailer is modeled, it is also checked for collisions. If z is not present, the obstacles and robot footprint meshes sent to the collision checker are a flat plane. The collision checker also takes as input a transform which describes the translation and rotation of the robot mesh and the environment obstacle that are being checked. The robot's transform is set based on the x, y, and θ values for the state being checked, and if heading is not present in the model, the robot mesh is checked with $\theta = 0$.

Collision checking can change depending on our robot model as described below:

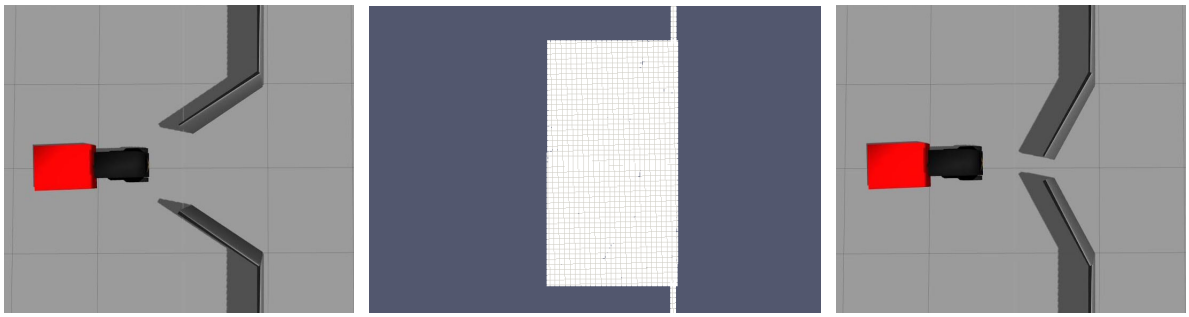
1. $\vec{q} = [x, y]$ = XY
   Collision checking ignores the trailer and assumes the robot heading is fixed: $\theta = 0$.
2. $\vec{q} = [x, y, \theta]$ = $XY\theta$
   Collision checking models the robot angle, but ignores the trailer.
3. $\vec{q} = [x, y, \theta, \theta_{trailer}]$ = $XY\theta\theta_1$
   Collision checking includes both the robot and trailer, with their headings.
4. $\vec{q} = [x, y, \theta, t, u_v, u_w]$ = $XY\theta V$
   Collision checking again ignores the trailer, but now includes a time variable (t) which is used to check collisions against time-sensitive obstacles such as swinging doors.
5. $\vec{q} = [x, y, \theta, \theta_{trailer}, t, u_v, u_w]$ = $XY\theta\theta_1 V$
   This model is the same as model $XY\theta V$, but includes the trailer in collision checking.
6. $\vec{q} = [x, y, \theta, t, v, w, u_a, u_\alpha]$ = $XY\theta VA$
   Collision checking is the same as with model $XY\theta V$: ignoring the trailer and including time-sensitive obstacles.
7. $\vec{q} = [x, y, \theta, \theta_{trailer}, t, v, w, u_a, u_\alpha]$ = $XY\theta\theta_1 VA$
   This model is the same as model $XY\theta VA$, and includes the trailer in collision checking.

For our worlds with automatic swinging doors, the state's time value is used to choose between occupancy grids that have the doors positioned entirely opened, partially opened, or closed. The partially open occupancy grid considers the entire swing area of the doors as an obstacle as shown in Figure 5.4. If the time value does not exist for the model, the obstacle representation of the doors is always in the open position. In our specific implementation the time value is only set if the robot's current propagation position is within two meters in the x-direction and 1 meter in the y-direction of the center of the swinging doors. Otherwise, for models with time, the value is set to a negative value (-99) which flags it to never be used for collision checking time-indexed obstacles. This is also the case for environments that do not include the swinging doors.

Collision checking with the full door swing area, between opening and closing, is an overly conservative approach. This causes a high false positive rate since the feasibility detector (Chapter 4.1.2) finds more collisions than actually occurs during execution, discussed more in Chapter 6.2.3. Alternatively, a less conservative approach would try to further discretize the door angles to better approximate the continuous door swing variable. If the discretization is not fine enough, paths could generate where the robot is located near the angled doors at one time-step and then teleport to the other side at the next time-step. This increases false negatives since the checker does not realize the door swings through the robot during execution. False positives are preferred over false negatives so we choose the conservative approach.

5.3.3 Trajectory Generation with Models

In this section we describe how the planner generates trajectories for each model type. This includes the variables sampled for selecting target waypoints and details for how to propagate the path towards these targets. We use the OMPL library (ompl.kavrakilab.org) for RRT plan-



(a) Doors opening. (b) Collision mesh of the door swing area. (c) Doors closing.

Figure 5.4: Conservative collision checking for automatic door world when doors are swinging.

ning. The Runge Kutta method numerically integrates the ordinary differential motion equations towards the next target point. Example trajectories generated with these different models are shown in Figure 5.5.

1. Lowest model: XY
   This is our only purely geometric model which does not use an integrated propagation function. The two dimensional state space is randomly sampled for a target. If the target exceeds a fixed maximum distance from the current waypoint (set to 1.0 meters), a target is interpolated at this maximum distance toward the target. The planner then collision checks in a straight line along that path by interpolating at some pre-set checking resolution. As the heading is not captured, it is assumed to be fixed at zero. Similarly, the trailer is not collision checked at all.

2. Models with heading: $XY\theta$, $XY\theta\theta_1$
   For models that include heading, targets are sampled in $[x, y, \theta]$. The linear velocity is set to a constant value. The angular velocity, ω is set based on the linear velocity, v, and the radius, r of the circle arc created between the sampled target and the waypoint being propagating from using $v = \omega r$. Lastly, we constrain the change in curvature between waypoints, and update the angular velocity accordingly, described below in 5.3.4. These controls are sent to the respective motion equation for propagation and collision checked at each time step based on a pre-defined step size. The propagation step count is sampled as described in 5.3.4.

3. Models with velocity: $XY\theta V$, $XY\theta\theta_1 V$
   Models with velocity also sample targets in $[x, y, \theta]$. They sample the next linear velocity using a Gaussian distribution centered around the current linear velocity, constrained within the limits of the robot. This is to reduce the robot's trailer swing. The angular velocity is set as previously described, including the re-computation for curvature change constraints (5.3.4). These velocity control values are set and reached instantaneously since acceleration limits are not considered.

   Velocity models also set a time variable. The time variable is updated by integrating the

durations. In our specific implementation the time value is only set if the robot's current propagation position is within two meters in the x-direction and 1 meter in the y-direction of the center of the swinging doors. Otherwise, for models with time, the value is set to a negative value (-99) which flags it to never be used for collision checking time-indexed obstacles.

Again, the state and controls are numerically integrated, iteratively propagating until a collision is found or the sampled propagation step count is reached.

4. Models with acceleration: $XY\theta VA$, $XY\theta\theta_1 VA$
   Acceleration models initially sample the next target linear velocity from a gaussian, they also sample in $[x, y, \theta]$, constrain the angular velocity to follow an arc towards the next sampled target, sample a propagation step count, and constrain the change in curvature.

   After we have the velocity values, we sample a bounded wheel acceleration value. At each time-step the delta velocity or maximum acceleration value is applied to the ODE propagation. During propagation, the state and control are integrated one step and then collision checked against any applicable obstacles. We continue propagating until a collision is found, or the total step count is reached. If the target velocity is achieved before the total step count is met, the robot coasts at the target velocity for the remainder of the step count.

Propagation equations

Models use the standard differential drive motion equations:

$$\begin{aligned}\dot{x} &= u_v \cos(\theta) \\ \dot{y} &= u_v \sin(\theta) \\ \dot{\theta} &= u_w\end{aligned}$$

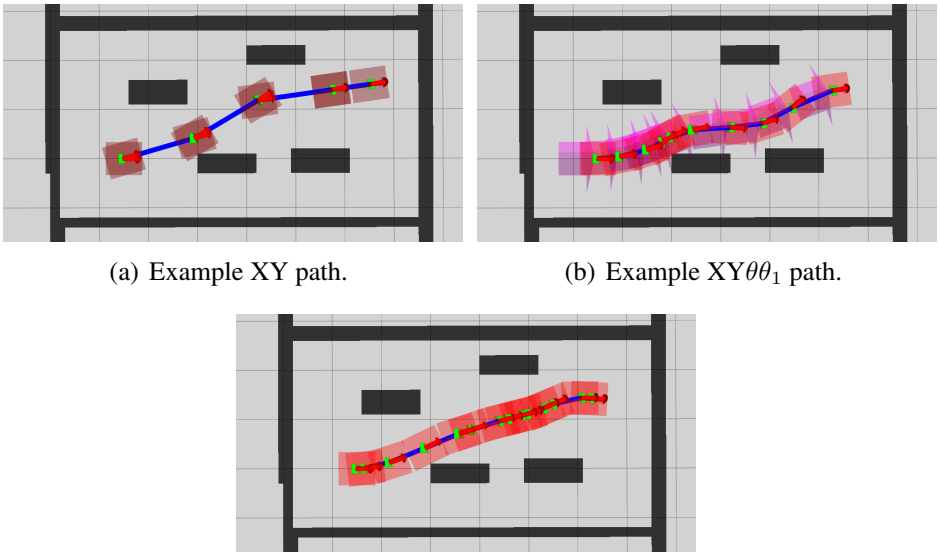
Models that include the trailer have an extra equation for the trailer's theta:

$$\dot{\theta}_{trailer} = (u_v/l)\sin(\theta - \theta_{trailer})$$

Models with sampled acceleration use a double integrator in their motion equations where velocities become part of the state:

$$\begin{aligned}\dot{x} &= v \cos(\theta) \\ \dot{y} &= v \sin(\theta) \\ \dot{\theta} &= w \\ \dot{v} &= u_a \\ \dot{w} &= u_\alpha\end{aligned}$$

Different models create different motion trajectories as shown in Figure 5.5. The XY path (Figure 5.5(a)) contains straight line geometric motions. The XY$\theta\theta_1$ path (Figure 5.5(b)) models the trailer and follows s-curve motions. Finally, the XYθVA path (Figure 5.5(c)) varies accelerations creating a smoother trajectory.



(a) Example XY path.

(b) Example XY$\theta\theta_1$ path.

(c) Example XYθVA.

Figure 5.5: Examples of paths generated using different models.

5.3.4 Trajectory Generation More Details

All models use the propagation functions except the purely geometric XY model. The propagation equations are integrated for one time-step and then collision checked against applicable obstacles. The time-step duration was set to 0.1. If the value is too large collisions with the environment are missed because collision checking is not done at a fine enough resolution with the environment, and if the value is too small it increases planning time. The propagation step count is sampled from a control duration range. This range was set to be $[10, 50]$. The size of this range affects the control duration between waypoints.

As stated previously, all models except for XY also constrain the change in curvature. This curvature is constrained to minimize trailer swing during the execution of these trajectories. This is done by comparing the curvature, k, ($k = 1/r, r = v/w$) of the previous linear and angular velocity to the current values. If the magnitude of curvature is over a set curve threshold (we set to 1.0), we sample a new constrained magnitude, less than 1.0, and use it to calculate a new angular velocity.

Padding for Planning

An initial footprint padding amount added for planning also helps increase overall success rates across simulation environments. We refer to this initial padding amount present on all base models as the nominal planning padding value.

Our current pure pursuit controller does not match the path's heading, instead adjusting the robot's heading to remain on the spatial path. This means that paths with waypoints close to obstacles with slight heading variations have a higher chance of being clipped by the robot. While this could be addressed with padding in the feasibility detector, nominal planning padding has smaller plan-time costs and results in fewer model switches in the final plan.

In summary, all models have some footprint padding, which may vary according to the model (whether it is a *base* model or an *uncertainty* model). The padding is used in collision checking during path generation, Section 5.3.2. We also note there is the model used for feasibility detection that has a different padding value that we describe in Section 5.4.1.

5.3.5 Model Translation

This section provides a more in-depth description for how we translate between models in our graph, Figure 5.3. As stated in Section 4.1.2, translation is a multi-step process. First, configuration variables are added as constants that were previously not considered as described in Section 3.1.2, and then translation for passive variables require knowing the next planned waypoint. We give more details for path translation which requires knowing the planned waypoints next in Section 5.3.6.

The model translation for each edge in the graph is described below.

1. $XY \rightarrow XY\theta$
   This translation requires an additional θ variable, calculated as the arctangent from the next waypoint along the planned path.
2. $XY\theta \rightarrow XY\theta V$
   This translation assumes a constant linear velocity. We set the linear velocity to be the midpoint of the velocity range that is sampled when generating a trajectory for models that contain the velocity value. In our implementation, this is $0.17m/s$. Time, t, is also added.
3. $XY\theta \rightarrow XY\theta\theta_1$
   This translation adds the trailer heading, θ_1, as a variable.
4. $XY\theta V \rightarrow XY\theta\theta_1 V$
   Again, this translation adds the trailer heading.
5. $XY\theta V \rightarrow XY\theta V A$
   This translation updates velocity transitions to respect an acceleration limit, instead of being instantaneous. This is set to $0.2m/s^2$. The robot is controlled by wheel velocities, so explicitly computing the final acceleration profile from the velocity derivative is not necessary.
6. $XY\theta\theta_1 \rightarrow XY\theta\theta_1 V$
   This translation assumes a constant linear velocity. Again, time t is also added.

7. $XY\theta\theta_1 V \rightarrow XY\theta\theta_1 VA$
   This translation modifies the velocity profile to respect acceleration limits, as described above.
8. $XY\theta VA \rightarrow XY\theta\theta_1 VA$
   Finally, this translation also adds the trailer heading.

Next we provide a detailed example of how the waypoint values change during path propagation which includes a path translation of the variables.

5.3.6 Path Translation Detailed Example

We present a detailed example of path translation, we include a table of three waypoint values that initially comprise an XY path, Table 5.1. The following tables contain the configuration variables for each models in column one. Then the remaining columns populate values for each variable of the three waypoints as the path gets translated into higher models. For each translation we show the table with an updated row for these new configuration values. We include this example here because we are showing the path translation as is done for each model during model selection, and equivalently for the highest model during feasibility detection. This path translation (which translates the waypoints) not only integrates the motion equations at each time-step but also follows waypoints using pure pursuit, and decides when waypoints are achieved. Therefore, paths can deviate depending on the model. We show the translation between two models at a time. A lower reference model is translated into a higher model for each edge in the model hierarchy. The waypoints between these models can deviate because of both pure pursuit augmenting controls to stay along the path and the logic for achieving waypoints. Each of these cause differences in the propagation towards the next waypoint. We note that every time we translate into a specific model, it is deterministic. The translated path is always the same for that model, and translated paths vary based on the model we are translating for.

Table 5.1: Initial XY path waypoints.

Model	Waypoint 1	Waypoint 2	Waypoint 3
$[x, y]$	[-2.5, -2]	[-1.54,-1.72]	[-0.65,-1.27]

1. $XY \rightarrow XY\theta$
   An additional θ variable (in radians) is added to waypoints. This is calculated as the arctangent from the next waypoint along the planned path, Table 5.2.

Table 5.2: $XY \rightarrow XY\theta$ path waypoints.

Model	Waypoint 1	Waypoint 2	Waypoint 3
$[x, y]$	[-2.5, -2]	[-1.54,-1.72]	[-0.65,-1.27]
$[x, y, \theta]$	[-2.5, -2, 0.28]	[-1.54,-1.72, 0.47]	[-0.65,-1.27, 0]

2. $XY\theta \rightarrow XY\theta V$
   This translation assumes a constant linear velocity. The XYθ row shows the path translated for this model using the motion equations where we set the linear velocity, as described before, this is $0.17m/s$. The angular velocity is set through geometric functions that calculate the chord and radius for the central angle arc with the next target waypoint. This constrains the robot to travel in arcs towards its next waypoint. Time, t, is computed by integrating the durations of waypoint transitions, computed from the velocities, Table 5.3. The duration between waypoints 1 and 2 is 5.4, and waypoints 2 and 3 is 6.3, where the integration time-step is 0.1.

Table 5.3: $XY\theta \rightarrow XY\theta V$ path waypoints.

Model	Waypoint 1	Waypoint 2	Waypoint 3
$[x, y, \theta]$	[-2.5, -2, 0.28]	[-1.62,-1.73, 0.39]	[-0.64,-1.35, 0.14]
$[x, y, \theta, t, u_v, u_w]$	[-2.5, -2, 0.28, 0, 0, 0]	[-1.62,-1.73, 0.39, 5.4, 0.17, 0.15]	[-0.64,-1.35, 0.14, 11.7, 0.17, -0.02]

3. $XY\theta \rightarrow XY\theta\theta_1$
   This translation adds the trailer heading, θ_1, as a variable, Table 5.4. It is computed using the same propagation functions as models which have a θ_1 value during plan-time, shown in section 5.3.3.

Table 5.4: $XY\theta \rightarrow XY\theta\theta_1$ path waypoints.

Model	Waypoint 1	Waypoint 2	Waypoint 3
$[x, y, \theta]$	[-2.5, -2, 0.28]	[-1.62,-1.73, 0.39]	[-0.64,-1.35, 0.14]
$[x, y, \theta, \theta_{trailer}]$	[-2.5, -2, 0.28, 0]	[-1.62, -1.73, 0.39, 0.21]	[-0.64, -1.35, 0.14, 0.29]

4. $XY\theta V \rightarrow XY\theta\theta_1 V$
   Again, for translations that add the trailer heading, Table 5.5, the propagation functions are used to populate this passive variable.

Table 5.5: $XY\theta V \rightarrow XY\theta\theta_1 V$ path waypoints.

Model	Waypoint 1	Waypoint 2	Waypoint 3
$[x, y, \theta, t, u_v, u_w]$	[-2.5, -2, 0.28, 0, 0, 0]	[-1.62,-1.73, 0.39, 5.4, 0.17, 0.15]	[-0.64,-1.35, 0.14, 11.7, 0.17, -0.02]
$[x, y, \theta, \theta_{trailer}, t, u_v, u_w]$	[-2.5, -2, 0.28, 0, 0, 0, 0]	[-1.62,-1.73, 0.39, 0.21, 5.4, 0.17, 0.15]	[-0.64,-1.35, 0.14, 0.29, 11.7, 0.17,-0.02]

5. $XY\theta\theta_1 \rightarrow XY\theta\theta_1 V$
   This translation assumes a constant linear velocity, and the angular velocity is computed as described above. Again, time t is computed by computing the durations between waypoints, Table 5.6.

Table 5.6: $XY\theta\theta_1 \rightarrow XY\theta\theta_1 V$ path waypoints.

Model	Waypoint 1	Waypoint 2	Waypoint 3
$[x, y, \theta, \theta_{trailer}]$	[-2.5, -2, 0.28, 0]	[-1.62, -1.73, 0.39, 0.21]	[-0.64, -1.35, 0.14, 0.29]
$[x, y, \theta, \theta_{trailer}, t, u_v, u_w]$	[-2.5, -2, 0.28, 0, 0, 0, 0]	[-1.62, -1.73, 0.39, 0.21, 5.4, 0.17, 0.15]	[-0.64, -1.35, 0.14, 0.29, 11.7, 0.17, -0.02]

6. $XY\theta V \rightarrow XY\theta VA$
   This translation updates velocity transitions (Table 5.7) to respect an acceleration limit set to $0.2m/s^2$. The time value and velocity values change slightly from the $XY\theta V$ model because the wheel velocities propagate differently to respect the acceleration limit. The time-step duration is now 5.8 from waypoint 1 to waypoint 2, and stayed 6.2 from waypoint 2 to waypoint 3.

Table 5.7: $XY\theta V \rightarrow XY\theta VA$ path waypoints.

Model	Waypoint 1	Waypoint 2	Waypoint 3
$[x, y, \theta, t, u_v, u_w]$	[-2.5, -2, 0.28, 0, 0, 0]	[-1.62,-1.73, 0.39, 5.4, 0.17, 0.15]	[-0.64,-1.35, 0.14, 11.7, 0.17, -0.02]
$[x, y, \theta, t, v, w, u_a, u_\alpha]$	[-2.5, -2, 0.28, 0, 0, 0, 0.2, 0.2]	[-1.63,-1.73, 0.39, 5.8, 0.17, 0.03, 0.2, 0.2]	[-0.65,-1.35, 0.14, 12.0, 0.17,-0.08, 0.2, 0.2]

7. $XY\theta\theta_1 V \rightarrow XY\theta\theta_1 VA$
   This translation modifies the velocity profile to respect acceleration limits, Table 5.8., as described above.

Table 5.8: $XY\theta\theta_1 V \rightarrow XY\theta\theta_1 VA$ path waypoints.

Model	Waypoint 1	Waypoint 2	Waypoint 3
$[x, y, \theta, \theta_{trailer}, t, u_v, u_w]$	[-2.5, -2, 0.28, 0, 0, 0, 0]	[-1.62, -1.73, 0.39, 0.21, 5.4, 0.17, 0.15]	[-0.64, -1.35, 0.14, 0.29, 11.7, 0.17, -0.02]
$[x, y, \theta, \theta_{trailer}, t, v, w, u_a, u_\alpha]$	[-2.5, -2, 0.28, 0, 0, 0, 0, 0.2, 0.2]	[-1.63,-1.73, 0.39, 0.21, 5.8, 0.17, 0.03, 0.2, 0.2]	[-0.65,-1.35, 0.14, 0.29, 12.0, 0.17,-0.08, 0.2, 0.2]

8. $XY\theta VA \rightarrow XY\theta\theta_1 VA$
   Finally, this translation also uses the propagation functions to add the trailer heading, Table 5.9.

In summary, for any translation that adds the trailer, θ_1 is added and updated by propagating towards the next waypoint along the path. For any translation that adds velocity: the linear velocity is set to a constant value, and the angular velocity is geometrically computed and constrained to circle arcs towards the next waypoint. Time, t, is computed by propagating these velocities along the full path. For any translation that adds acceleration, the linear and angular velocities

Table 5.9: $XY\theta VA \rightarrow XY\theta\theta_1 VA$ path waypoints.

Model	Waypoint 1	Waypoint 2	Waypoint 3
$[x, y, \theta,$ $t, v, w, u_a, u_\alpha]$	[-2.5, -2, 0.28, 0, 0, 0, 0.2, 0.2]	[-1.63,-1.73, 0.39, 5.8, 0.17, 0.03, 0.2, 0.2]	[-0.65,-1.35, 0.14, 12.0, 0.17,-0.08, 0.2, 0.2]
$[x, y, \theta, \theta_{trailer},$ $t, v, w, u_a, u_\alpha]$	[-2.5, -2, 0.28, 0, 0, 0, 0, 0.2, 0.2]	[-1.63,-1.73, 0.39, 0.21, 5.8, 0.17, 0.03, 0.2, 0.2]	[-0.65,-1.35, 0.14, 0.29, 12.0, 0.17,-0.08, 0.2, 0.2]

will change to be within proper acceleration limits that are initially set as a constant maximum acceleration value.

5.4 Reference Path Propagation Details

In this section, we want to make clear the distinction between propagating a lower reference path, collision checking, and translating a lower reference path into a higher model. Reference path propagation occurs during feasibility detection and model selection. When we say propagating along a path this means we are integrating the state variables at each time-step as defined by our motion equations, and also waypoint following the path. The motions equations are the same equations used for generating trajectories. The integration occurs between waypoint nodes that are previously generated in a particular model. These nodes are then used as reference target points for propagating between. Propagation between nodes along a reference path ends when we determine the node is achieved. This occurs using the same logic as our robot controller, previously described in Section 5.2.

When we refer to checking a path, this involves both propagating between target waypoints and collision checking. This means that for each time-step of the propagation we then check the validity of the current robot's state by collision checking between the robot mesh and applicable obstacles. This propagation and collision check continues until a collision is found or the goal waypoint is achieved.

It is also possible to propagate along a reference path without collision checking. This just updates the robot's state variables according to the given control inputs and motion equations. This is useful for translating a path into a higher model.

In the next subsections, we describe more details of when we propagate, translate, and collision check in our model selector, and when this is also done during feasibility detection.

5.4.1 Feasibility Detector and Model Selection

The major differences between the feasibility detector and the model selector are that they check the reference path in different models and they check different parts of the path. The feasibility detector always checks along the full multi-model reference path, and the model selector only checks partial parts of this reference path. The feasibility detector checks the model reference path in our highest possible model, and the model selector uses different models from the model hierarchy.

Both the feasibility detector and model selector are trying to follow the reference path similarly to how the robot executes in the environment. The feasibility detector is doing this most accurately by always using the highest model, and the model selector is being less accurate when using a model that is not the highest model. In both cases, the planner has a control law for waypoint following the path that is the same as the robot controller. This is because the model selector wants to choose a model that will both generate a path around the infeasibility and is likely to work in execution (pass feasibility detection of the entire path). Since it is not possible to know if a path will fail without executing it, checking the path is a cheap way of trying to guess if the model will be useful. Checking the path is cheap because there is not a new search of the space as occurs during planning. Therefore, to check the path there needs to be logic for staying along the lower path (using it as a reference path) and knowing when a waypoint is achieved in order to go towards the next waypoint. More explicitly, both the feasibility detector and the model selector use pure pursuit to guide the propagation along the reference path. At each time-step the planner compares the current position along the reference path with the next target waypoint. If the waypoint is within 0.1 meters of the current robot's x, y position and 0.09 radians of the robot's heading, or the waypoint has been crossed (previously described in section 5.2.1), then the waypoint is achieved. If the waypoint is not achieved, propagation along the path continues. First pure pursuit adjusts the angular velocity to stay along the path as necessary (refer to Section 5.2), then depending on the model, a propagation integration step occurs for the control inputs using that specific model's motion equations.

Models differ in terms of fidelity because the controls and state variables used for propagation and the state sent for collision checking can differ. We note that the only exception is our purely geometric model XY which does not use controls but interpolation between points. Higher models may need more configuration variables set. For example, models with velocity have an additional time variable and so they can now collision check indexing time sensitive obstacles, trailer models have the additional trailer variable which gets updated with propagation and is used to collision check with the trailer, and models with acceleration change the velocity values to respect those limits as well as using a double integrator for their ODE transition functions. Therefore, while waypoint following the path, the model used determines what motion equations are used for propagating the path, and what robot state is collision checked. We describe what is collision checked for different models in the collision checker Section 5.3.2, and we show the different motion equations propagated for our models when describing the trajectory generation in Section 5.3.3. We also note that for each node that is a connection (as described in Section 5.5), we propagate towards the top connection.

It is possible to translate the path to a specific model as a separate step, but in our implementation path translation actually occurs during path propagation. For example, the feasibility detector uses the highest model's motion equations to propagate along the path updating the passive trailer heading. After each integration step of the motion equations, the velocities are updated to be within proper acceleration limits (dependent on the next target waypoint), and the time variable is updated based on the number of control steps taken. Therefore, the only setup path translation requires is saving additional state variables necessary for higher model nodes. This is already done when saving our global path, discussed more in Section 5.5. Passive variables can be set to zero because they are updated through the propagation of the reference path. To be clear, for the highest model the velocities may also be augmented to stay within

acceleration limits at each time-step.

The model selector always propagates the reference path from the start state, but does not always collision check during this propagation. The feasibility detector both propagates the lower reference path and collision checks from the start to the goal in the highest model. However, the model selector only collision checks a partial portion of the path. After the feasibility detector determines which two waypoints the infeasibility is between, this information is sent to the model selector. The model selector propagates the path in the model it is currently testing from the start waypoint to the waypoint following the infeasibility. This is so the partial path around the infeasibility is properly translated. Then collision checking only occurs between the two waypoints that contained the infeasibility. This is because it is possible for models lower than the highest model to prematurely detect a collision along the lower model reference path, and the repair area should not change based on checking in a model that is not the same as the feasibility detector (the highest model). For example, if the model being tested is not the highest model then it is assumed to not be the *best* representation of how the actual robot moves in the environment. So even if a lower model (say $XY\theta$) deviates from the lower reference path (an XY portion) in a location before where the feasibility detector determines a collision, this deviation is a less informed evaluation of the robot moving in the world. The purpose of the model selector is to choose the lowest applicable model to cover the collision detected in the highest model (using the feasibility detector). Therefore, the model selector checks the reference path within lower models from the hierarchy that do not move exactly as the robot does. The first model that finds a collision in the same area as the feasibility detector is assumed to provide information beneficial for circumventing the infeasibility without needing as much information as the highest model.

Padding for Feasibility Detection

We seek a balance between successful execution rates and shorter planning times. To achieve this, it is important to balance between false positives (i.e. predicting collisions that do not actually occur during execution) and false negatives (i.e. not predicting collisions that do actually occur) that arise. These false positives and false negatives occur due to inconsistencies between the planner's feasibility detector, a model of how the robot interacts with the environment, and how the robot actually interacts in the real (simulated) world, subject to full physics. We prefer to minimize false negatives, as these represent unsuccessful executions of the algorithm.

The feasibility detector should match the real world as close as possible, meaning the highest model checker should match to how the robot controller moves. Differences between the checked path and execution path occur because our highest checking model does not model dynamics. Our highest model is kinematic and does not account for mass which better approximates trailer swing. Consequently, the trailer may graze an obstacle during execution which is undetected by the feasibility detector. Thus, additional padding decreases these cases reducing the false negative rate.

Robot footprint padding is added during feasibility detection. Additional padding is added with uncertainty models, described previously in Section 3.1.1. The feasibility detector should check with the least amount of padding necessary to minimize false negatives but also perform with similar success as the real robot. If the detector has too much padding, plans are errantly rejected and the false positive rate increases. To achieve this balance we chose padding values for

one gurney world, increasing the value until the false negative values were zero while keeping false positive values less than 15. This should be optimized independent of the environment in the future. The value was set to 4 centimeters which adds 2 cms of padding to the robot footprint in both the x and y directions. There is also a relationship between the nominal footprint padding at plan-time and the feasibility detector padding. The padding of the feasibility detector should be less than or equal to this value. Otherwise the false positive rate is unnecessarily high.

5.5 Path Merging

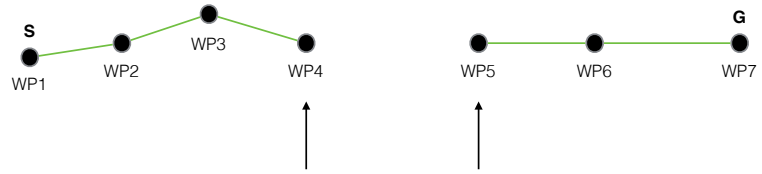
We maintain a global path through out the planning process. The feasibility detector determines areas along this path which need model switching, and the model selector determines the model for re-planning in those specific areas. When re-planning occurs, using multiple re-plan trees, a partial path repair is merged back into the global path. We first discuss this in Chapter 4.1.4. We provide more implementation details of this below.

5.5.1 Path Merging More Details

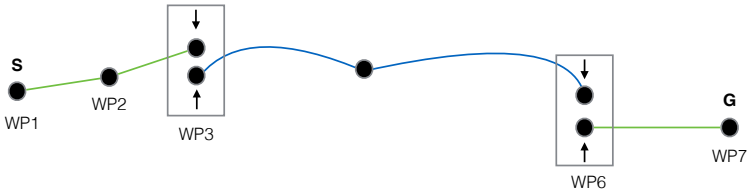
We first generate a plan in the XY model and this is our first global path. When our global path is saved we do a translation step that just updates waypoint state values to be equivalent to that of the highest model. This is a model translation step and is not translating the path. For the XY model, this implementation includes setting the θ value based on the next target waypoint, setting the linear velocity to a constant value, the acceleration limit to a constant value, and setting the angular velocity as described in the translation Section 5.3.5. The remaining values (trailer heading, and time) are set to zero and are updated when the path is propagated.

Every path that is planned gets saved into the global path and each node of that path is updated to include the same state as the highest model. During the re-planning phase the original global path waypoints are tracked, Figure 5.6 (a). This is so that if any waypoints are skipped during the re-plan process the original re-merge points are maintained. For example, a re-planning tree that starts before the waypoint immediately proceeding the infeasibility could connect to an intermediate goal after the waypoint directly following the infeasibility. The original path waypoints from where the partially re-planned portion started and ends become the connection points where the new partial path is re-merged in.

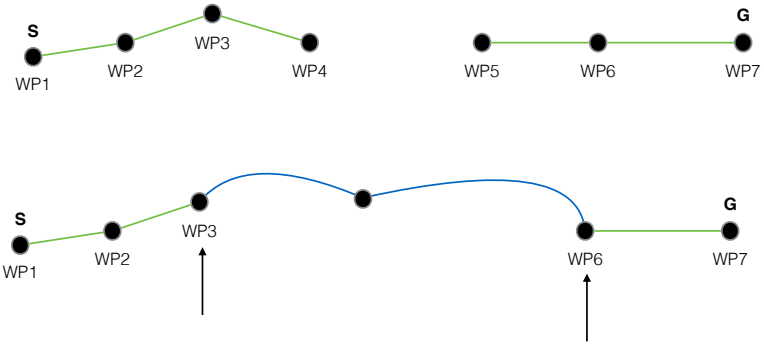
Merging partial paths into the global path requires the addition of a connection node. A connection node is created if the new partial path needs to be added to an old existing part of the global path. A connection node consists of a top part and a bottom part, Figure 5.6 (b). The bottom part of the connection node is the start of a model path (our new re-planned path). When adding the start connection for a newly merged path, we keep the node from the old global path as the top, and then add the start node of the new partial path as the bottom connection. We add the start node the same way as before where state variables are added to translate the node into the highest model regardless of what model was used for re-planning. This continues for every new node in the merged partial path until the end goal of the partial path. The goal from the new partial path is added as the top connection to the old path's bottom connection, Figure 5.6 (b). The merged path is shown in Figure 5.6(c). The values between the top and bottom connection



(a) Global path which needs repair between waypoints 4 and 5.



(b) Connection nodes are created for waypoint 3 and 6.



(c) A new partial re-planned path is merged back into the global path.

Figure 5.6: Path merging example.

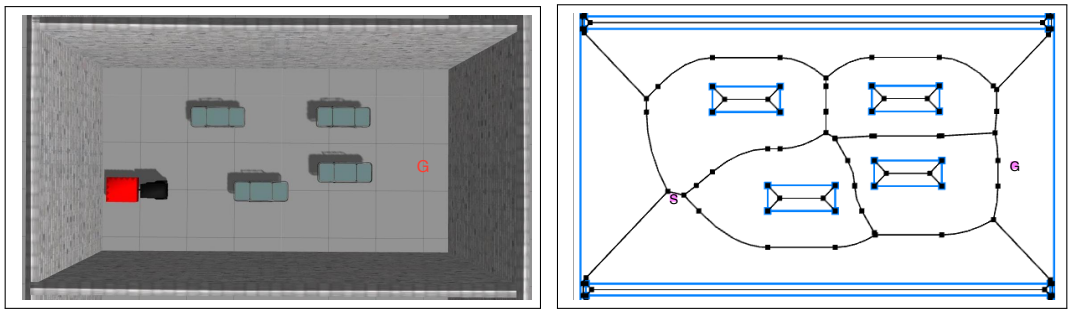
are then updated as necessary, and in our implementation the only additional step that is needed is XY paths require the old θ value for these connections to be updated appropriately to point towards the next target waypoint along the path. It is ok if top and bottom connection nodes do not match exactly, as achieving a waypoint during propagation is defined by a range, and the waypoint follower keeps propagation along the path as described in the next Section 5.4.

By maintaining the global path with state variables which include those of the highest model, the feasibility detector and model selector can take paths directly from the global path. The model selector can take a partial path from the global path that only includes the configuration variables necessary for the particular model it is checking in. Therefore, the configuration variables have been appropriately added for this model (model translation) and then the configuration variables are updated by propagating along the path (path translation).

5.6 Voronoi Implementation for Uncertainty Models

The Voronoi diagram uses distances between points to partition the space. These partitions are created based on cells, edges, and vertices. A cell is defined as the collection of points closest to this cell center over any other. Edges are placed equidistant between the center points of each cell, and a vertex is an equidistant point between three cell centers.

A generalized form of the Voronoi diagram is often used in path planning, [44], where obstacle edges are approximated with points such that cells contain the center of obstacles. One way of generating the generalized Voronoi is to approximate the edges of obstacles with points, generate the Voronoi edges and vertices, and then remove all edges that intersect an obstacle. Alternatively, we utilize the boost library (`www.boost.org/doc/libs/1_55_0/libs/polygon/doc/voronoi_main.htm`) to generate a Voronoi diagram based off poly line segments used to approximate the outside of obstacles. Any edges that intersect obstacles are ignored. Below we show a picture of a gurney environment Figure 5.7 (a) with its corresponding Voronoi diagram in Figure 5.7 (b).

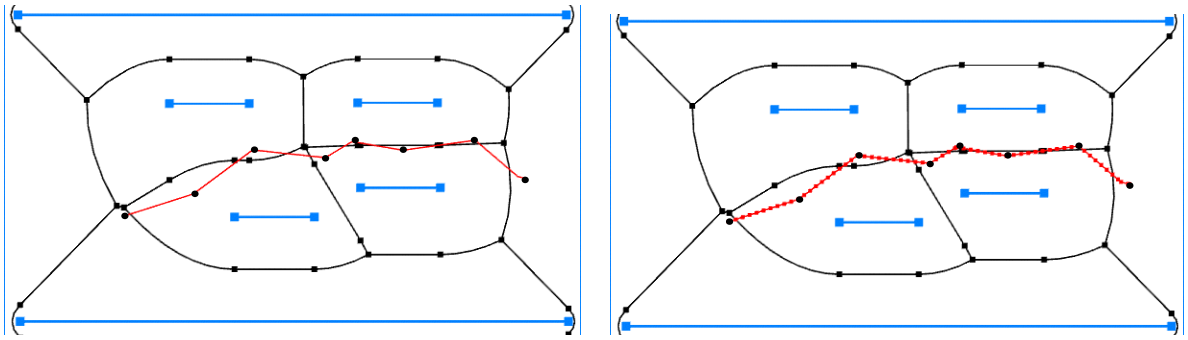


(a) Environment 0. (b) Voronoi Diagram for Environment 0.

Figure 5.7: Gurney world with corresponding Voronoi diagram.

We define helper functions for gathering information from the Voronoi Diagram to use in our implementation. One is finding the closest point perpendicular to a Voronoi edge from a path. This is used when creating the padding model as described in Chapter 4.2.1. Finding the closest

point along a path to a Voronoi edge is also useful when determining which homotopy class a current path matches in the Voronoi Diagram.



(a) Original path placed in the diagram.

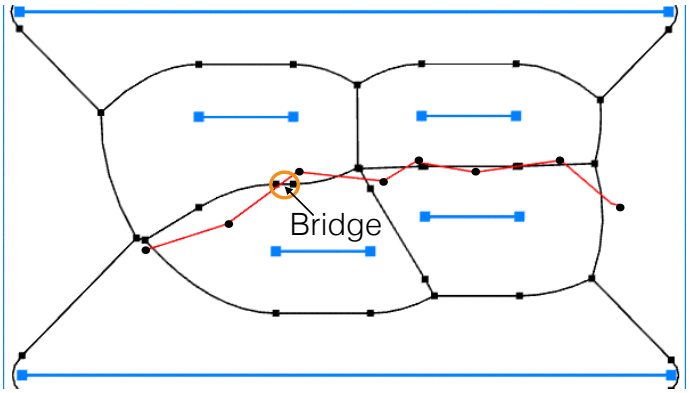
(b) Increasing the resolution along the path for better matching Voronoi edges.

Figure 5.8: Various Voronoi diagrams illustrating finding the best matching homotopy path through the diagram.

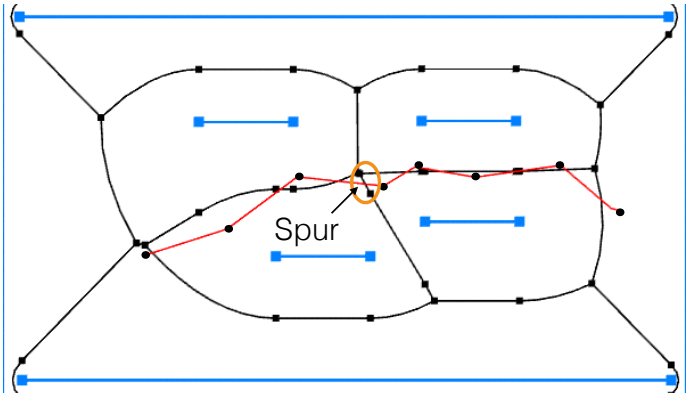
Homotopy partitions paths into equivalence classes [49]. The equivalence class of two continuous paths is homotopic if for a fixed start and end point one path can be deformed into another without going through an obstacle. Since homotopy classes can be obtained from a Voronoi diagram, [5], it is possible to determine the equivalent homotopy class that matches along Voronoi edges for a given planned path.

To determine the path in the Voronoi diagram that best matches a given path we use the following steps:

1. construct the Voronoi diagram from the environment
2. add additional waypoints along the path to increase the waypoint resolution, Figure 5.8(b) (this is used for determining edges in the Voronoi that are closest to points along this given path).
3. for every point on the path, find the shortest distance between each point and every edge (a line segment) in the Voronoi diagram.
4. maintain these closest edges. This results in edges that may contain gaps where there is not a continuous path yet from start to end, Figure 5.9(a).
5. bridge the gaps: for every close edge get all groups of connected edges. Starting with the group that contains the start edge find the closest vertices of the edges between groups. Create a bridge between the groups by adding edges, from the Voronoi diagram, to the group until the vertices are connected. Keep connecting groups until there is a connected group that contains both the start and end edges.
6. eliminate spurs: for all of these connected edges find the edges that are only connected by one vertex to another edge, Figure 5.9(b). If it is not the start or end edge eliminate it from the group. Keep doing this until the only edges connected by one vertex are the start and end, so there are no more spurs. This results in a continuous path in the Voronoi that is closest to the passed in path, and represents the homotopy equivalent class path for this



(a) An example of a bridge gap in the diagram.



(b) An example of a spur in the diagram.

Figure 5.9: Example gap and spur in the Voronoi diagram.

path, Figure 5.10.

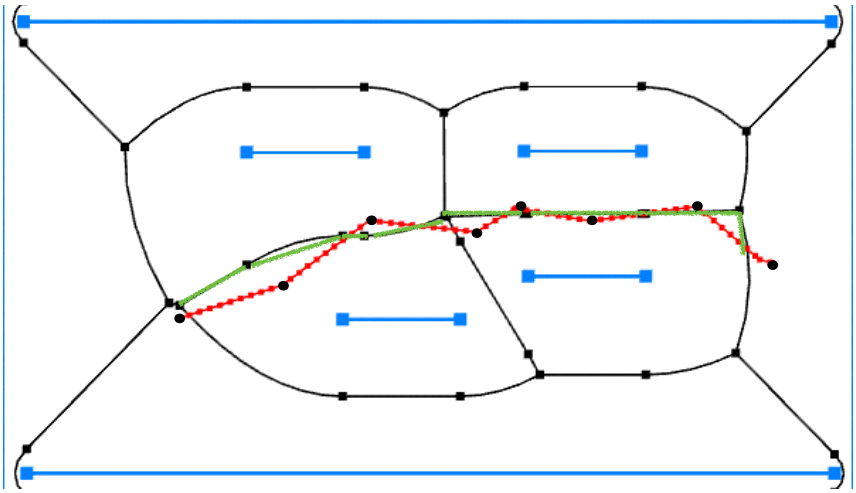


Figure 5.10: The final matching Voronoi homotopy path.

We also use the Voronoi diagram to determine if finding a plan with a desired padding amount is achievable. This is also first described in Chapter 4.2.1. First we determine source points for the cell used to create every Voronoi edge and compute the distance until we have found the minimum distance from the source point to any point on this edge. This is the minimum distance between the source and edge for a given edge. Then when checking if there is a traversable path for a given padding value, we compare the allowed padding amount to the minimum traversable distance for a given edge. If there is enough allowance for the padding amount with the robot to traverse for this particular edge, meaning it is within the minimum distance to a source point, then this edge is marked as traversable.

The last step is to find a traversable path among those edges marked as such. This is a traversable path which contains continuous connected edges that include the start and goal locations. We mark the start edge and the goal edge by projecting a point perpendicular to the closest edge for each start and goal location. Then we get all connected neighbors to the start edge by finding edges that share a vertex with the start edge. We repeat this process until we have found the goal edge. If we do not find a connection to the goal edge, and have exhausted all edges, then a traversable path for this padding value is not found.

Chapter 6

Testing Environments and Experiments

As first introduced in Section 5.1, experiments are run using a differential drive robot with a rear attached trailer in the high fidelity Gazebo simulator (http://gazebosim.org/). The simulator models dynamics by simulating rigid-body physics. Our testing environments are motivated from that of a hospital environment with automated swinging doors and gurneys in the hallways. Therefore, we showcase these obstacle types by running experiments using ten different environments with randomly placed gurneys and three environments which include an automatic door set to open for different amounts of time. We discuss how well our approach performs over these environments.

The specific models used in all of our experiments are described in detail in Section 5.3, and the model graph is shown again in Figure 6.4 for reference.

We present our results in three sections. First we discuss single model performance, then we present results for switching among the multiple models, and finally we evaluate switching with additional uncertainty models. For single models, we plan using each model from our graph in Figure 6.4, without any switching. Evaluating single model performance includes statistics on success rate, planning-time, and path lengths. We also show false positive and false negative rates for single model runs for different environment. We describe these values using a confusion matrix in Section 6.2.3. The confusion matrix for each model describes the accuracy between the feasibility checker and execution controller. For multi-model switching results, we present statistics for our three tree-weighting heuristics from Section 6.3.1, and further evaluate considerations for comparing switching to single model planning. Lastly, we discuss the addition of uncertainty models and how that affects overall switching results for a particular environment.

In each of our sections, we present results for our two environment types. We start with presenting results for our gurney environments. We first provide overall statistics for the different gurney worlds, then highlight one gurney environment, and lastly present interesting results for a subset of the remaining gurney environments. This includes any interesting aspects of single model evaluation, switching, and switching with uncertainty models for any particular gurney model. Then we show results for our automatic swinging door environments.

6.1 Testing Environments

6.1.1 Gurney Environments

The gurney environments contains multiple hospital gurney obstacles placed in a hallway that the robot must plan through. (Figure 6.1). The top center world we constructed by hand, and the remaining nine worlds contain randomly generated gurney positions. For each world the goal is indicated by the black 'G' and the environments are labelled E1 through E9. These environments contain examples that are more open (E1, E3), and progressively more constrained. For example qualitatively, E4 seems difficult due to the gurney close to the start, then E6 and E7 contain gurneys concentrated between the start and goal, with E9 having a gurney very close to the goal state. We hypothesize environments with more constrained areas for the robot to pass through will require more switching than those without.

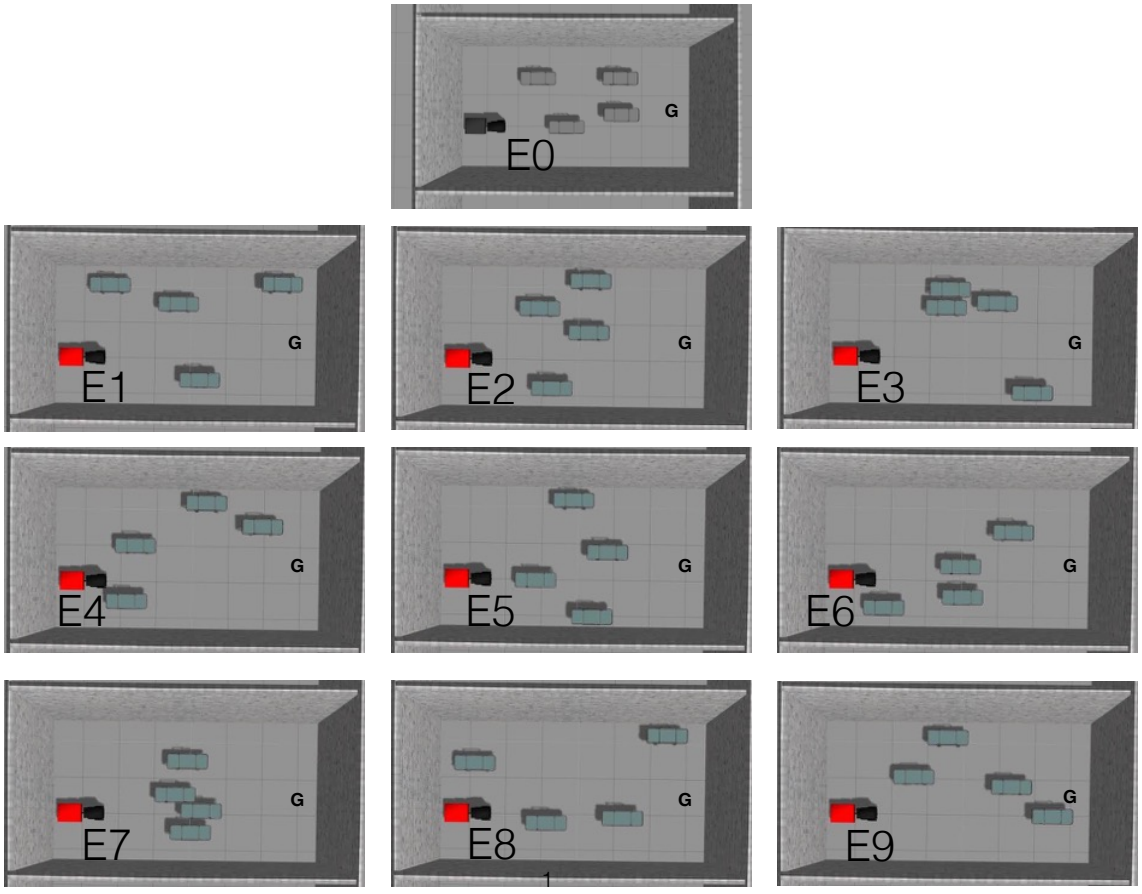
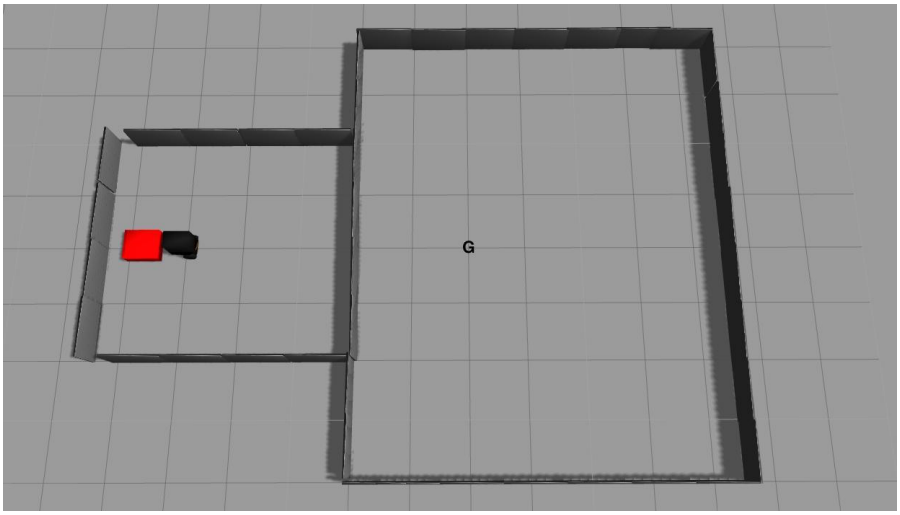


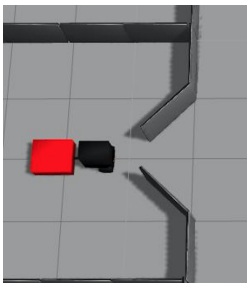
Figure 6.1: Many hospital environments with multiple gurneys.

6.1.2 Swinging Door Environment

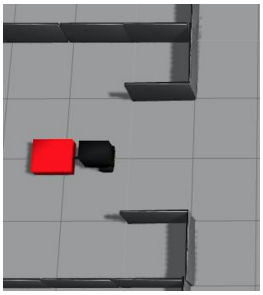
An automatic swinging door is displayed in Figure 6.2(a). The goal is displayed by a black 'G'. We ran experiments for three variants of the swinging door where the length the door stays open changes. The door is open for either 12, 14, or 16 seconds. The door swings open by applying a constant velocity to the hinge joints and takes 2.5 seconds to open, and then closes in the same time.



(a) The automatic swinging door world. The doors are closed before and after being opened.



(b) The doors partially opening or closing.



(c) The doors fully opened.

Figure 6.2: Swinging doors world and various stages of the door opening and closing.

The pictures in Figure 6.2 show the door as fully open, Figure 6.2 (b), or swinging open/closed, Figure 6.2 (c). The door is activated (to first open) when the robot is in front of the doors within 2m of the x-axis and 1m of the y-axis. The door stays open for a specified interval of time (12, 14, or 16 seconds) and then closes. The amount of time the door takes to close is fixed for all environments. The doors swing at a constant velocity and re-open as long as the robot is within the activation range.

The swinging doors are meant to show the importance of planning with velocity and acceleration. As an example, a path generated in this world using the highest acceleration model is

shown in Figure 6.3. The black rectangles highlight the door opening that must be collision checked at different time intervals. The waypoints are closer together before the door, then they become farther apart as the robot accelerates through the door opening.

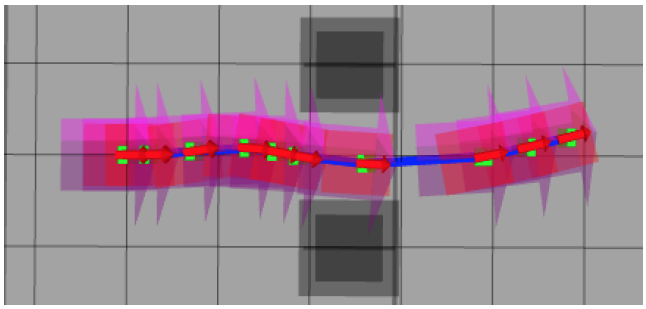


Figure 6.3: A path generated in the swinging doors world using model $XY\theta\theta_1VA$.

6.2 Single Model Performance for Various Environments

In this section, we display statistics over various environments for single models shown again in Figure 6.4, for reference. For our single model experiments, we run 100 trials for each model. To produce good plans, we generated 20 RRT plans for each trial and choose the best (shortest) one for execution. This is because we do not otherwise smooth our paths. After we generate the plan, we send it to the (simulated) robot for execution and record if the robot executed successfully.

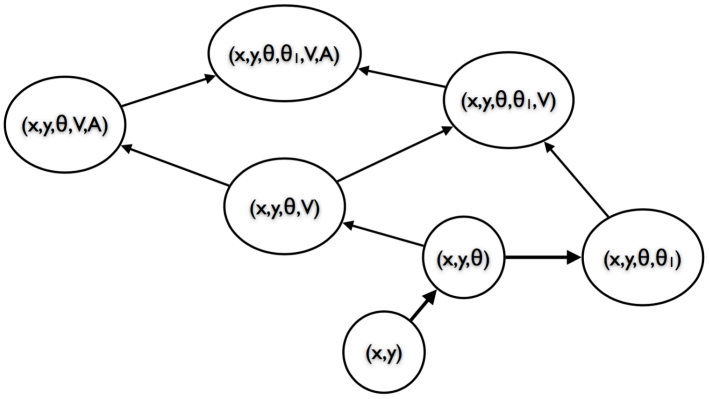


Figure 6.4: Full model graph used for experiments.

6.2.1 Single Model Performance for Gurney Environments

In the following subsections, we describe the single model performance for our gurney environments.

Average Performance

We start by presenting statistics for the average performance over all of the gurney environments.

Table 6.1: Average performance for single models over all gurney environments.

Model	Success (SD)	Plan Time (SD) (s)	Path Length (SD) (m)
[x, y]	0.62(0.40)	0.12 (0.23)	6.29(0.68)
[x, y, θ]	0.56(0.46)	65.19(178.22)	6.08(0.63)
[x, y, θ, θ_1]	0.92(0.12)	92.40(193.49)	6.23(0.80)
[x, y, θ, V]	0.62(0.38)	9.83(7.14)	6.33(0.84)
[x, y, θ, θ_1 ,V]	0.93(0.12)	26.29(36.34)	6.44(0.92)
[x, y, θ, V, A]	0.63(0.39)	12.38(5.49)	6.36(0.87)
[x, y, θ, θ_1, V, A]	0.97(0.06)	30.07(35.75)	6.42(0.89)

In Table 6.1 we see that models with the trailer are more successful across gurney environments than those without. But, planning times with the trailer take longer than the same models without the trailer. We also see that success rates for models without the trailer are comparable. The [x,y,θ] model performs worse in some gurney environments.

Planning times also do not increase monotonically with fidelity. Times for the [x, y, θ] and [x, y, θ, θ_1] models are higher than others, but with a very high variance. The high variance suggests that the environments where these models take longer are not the majority.

Performance for Specific Gurney Environments

In this section, we highlight interesting examples of other gurney environments for single model runs. We first describe our E0 gurney environment in Figure 6.1 and environments that gave similar results. Then we highlight an example environment from the subset of more open environments. Lastly, we give an example of an environment that did not preform well with our approach.

We first present singe model performance for an initial representative world with gurney obstacles, this is our E0 model in Figure 6.1. For this environment, we notice that success rates increase with models that include the trailer. Planning times increase for higher fidelity models, and models that include the trailer have higher planning times than those without. Similarly with path length, longer paths for models with the trailer than without.

In Table 6.2, We noticed that the [x,y,θ] model had lower success than the [x,y] model for this particular environment. This is due to implementation details for [x,yθ] as compared to [x,y]. For our [x,y] model we do not account for different θ values when collision checking and therefore are always collision checking the robot footprint with θ equal to zero. Since the robot footprint and our gurney obstacles are both rectangles, collision checking the robot footprint at θ equal to zero creates paths that are further from obstacles than paths generated for a robot whose θ value can vary. This is most evident for the gurney's corners. Due to this, we find that [x,y,θ] paths, for this environment, are closer to the gurney's corners which causes the robot's trailer to collide more with obstacles during execution.

Table 6.2: Single Model Results for the E0 Environment. Planning time increases with fidelity, and success and path length increases with the modeling of the trailer.

Model	Success	Plan Time (SD) (s)	Path Length (SD) (m)
[x, y]	0.48	0.05(0.01)	5.94(0.13)
[x, y, θ]	0.18	7.33(1.95)	5.78(0.09)
[x, y, θ, θ_1]	0.98	9.12(2.51)	5.94(0.08)
[x, y, θ, V]	0.60	10.53(2.95)	6.00(0.21)
[x, y, θ, θ_1 ,V]	0.99	17.69(6.78)	6.12(0.14)
[x, y, θ, V, A]	0.77	11.84(3.22)	6.05(0.17)
[x, y, θ, θ_1, V, A]	1	18.35(6.08)	6.13(0.13)

Environments that had very similar performance as our example gurney world include environment E6 and E9 as shown in Figure 6.1. These worlds also have higher success rates and planning times for models that include the trailer. They also have very low success rates for the [x,y,θ] model. Models without the trailer for both E6 and E9 do not have success rates over 20%. We believe this is due to trying to reach the goal at the finish of a turn. We also note that for world E9 our highest model only achieved a 98% rate.

We found that half of our environments were more open, and the lowest model actually fails less often. These environments include E1-E4 and E8 shown in Figure 6.1. They generate paths more directly to the goal with fewer turns. We focus on environment E8 as our representative example. These are the types of environments we would expect the robot to encounter most often, that is, environments where the majority of the time the current lowest model the robot plans in is sufficient.

Table 6.3: Single model performance for the more open environment E8.

Model	Success	Plan Time (SD) (s)	Path Length (SD) (m)
[x, y]	0.92	0.02(0.01)	5.81(0.14)
[x, y, θ]	0.97	6.46(1.67)	5.67(0.09)
[x, y, θ, θ_1]	1	6.91(1.88)	5.67(0.07)
[x, y, θ, V]	0.98	8.72(2.57)	5.94(0.32)
[x, y, θ, θ_1 ,V]	1	8.34(2.32)	5.90(0.25)
[x, y, θ, V, A]	1	9.45(3.03)	5.92(0.30)
[x, y, θ, θ_1, V, A]	1	10.12(2.90)	5.97(0.30)

Table 6.3, shows an environment where most all of the models are successful. Again, models with the trailer are more successful than those without. We also see planning time increases with fidelity with the exception of the velocity model with and without the trailer being very similar.

The two remaining gurney environments, E5 and E7, have obstacles more concentrated in the center. Both of these environments caused higher planning times for [x,y,θ,θ_1] than the higher

models. We believe this is due to the position of the gurneys. We highlight success and planning times for single model runs in environment E5 in Table 6.4.

Table 6.4: Single model performance for a more constrained environment E5.

Model	Success	Plan Time (SD) (s)	Path Length (SD) (m)
[x, y]	0	6.10(60.30)	6.60(0.43)
[x, y, θ]	0.36	582.0(134.0)	6.67(0.15)
[x, y, θ, θ_1]	0.83	609.0(144.0)	6.70(0.16)
[x, y, θ, V]	0.26	18.10(2.61)	7.63(1.01)
[x, y, θ, θ_1 ,V]	0.69	29.7(5.65)	7.30(0.86)
[x, y, θ, V, A]	0.19	20.10(3.77)	7.78(.0.97)
[x, y, θ, θ_1, V, A]	0.84	32.80(5.87)	11.30(10.10)

Planning times were longer overall for both E5 and E7. We believe the poor performance for the trailer model without velocity is because it assumes a constant linear velocity. The close gurneys create small openings that are difficult for the planner to determine how to have the robot get through those openings.. It is possible planning stalls due to constraining the curvature which makes it harder for the robot to take tight turns through the space.

.

6.2.2 Single Model Performance for Automatic Swinging Door Environments

We have three swinging door environments where the door open times are 12, 14, or 16 seconds, and the time it takes for the doors to swing open is 2.5 seconds and then close is 2.5 seconds. When planning without velocity, the door is always assumed to be open. Models with velocity consider the time dimension and index the time with the door opening and closing during collision checking.

Average Performance

We first present a table, Table 6.5, of the average performance for single models over all swinging door environments.

We see that higher models which include velocity and acceleration are always successful regardless of the door opening times. Planning times are also highest for models that include both the trailer and velocity. Including the trailer increases the time it takes for the RRT to find a plan through the door opening because both the robot and trailer footprints must clear the door opening. The success rates for lower models hovers around 50%. This is because when the door is open longer lower models are successful every time, and when the door is open shorter lower models are not successful. The path length for the [x,y] model is longer and with a higher variance due to the implementation for the geometric path generation. These paths have a maximum extension between waypoints which creates intermediary waypoints near the door

Table 6.5: Average performance for single models over the automatic swinging door environments.

Model	Success (SD)	Plan Time (SD) (s)	Path Length (SD) (m)
[x, y]	0.46(0.48)	0.03(0.05)	5.30(0.21)
[x, y, θ]	0.57(0.51)	12.04(4.39)	4.99(0.04)
[x, y, θ, θ_1]	0.57(0.52)	13.46(4.88)	4.99(0.04)
[x, y, θ, V]	1(0)	26.25(12.81)	5.11(0.09)
[x, y, θ, θ_1 ,V]	1(0)	75.88(48.74)	5.10(0.09)
[x, y, θ, V, A]	1(0)	29.50(14.84)	5.11(0.10)
[x, y, θ, θ_1, V, A]	1(0)	77.72(48.73)	5.10(0.10)

opening. The path target waypoints are sampled randomly, and since the robot heading is always assumed to be zero, in some cases the connections between waypoints are not very straight which contributes to the larger variance.

Performance for Specific Swing Door Environments

The table below gives statistics for single model runs when the doors are open for 14 seconds. Then we discuss trends that we see for door opening times that are longer or shorter.

Table 6.6: Single model performance for an automatic swinging door set to open for 14 seconds.

Model	Success	Plan Time (SD) (s)	Path Length (SD) (m)
[x, y]	0.43	0.04(0.08)	5.28(0.18)
[x, y, θ]	0.70	13.78(5.20)	4.99(0.04)
[x, y, θ, θ_1]	0.72	16.09(4.84)	4.99(0.04)
[x, y, θ, V]	0.98	23.15(11.99)	5.09(0.09)
[x, y, θ, θ_1 ,V]	1	49.2(20.6)	5.11(0.09)
[x, y, θ, V, A]	0.99	25.80(12.37)	5.11(0.09)
[x, y, θ, θ_1, V, A]	1	49.13(21.60)	5.11(0.09)

In Table 6.6, we observe higher success rates for models that include both velocity and the trailer. The success rate for lower models increases with fidelity. We believe this is just due to chance. All models without velocity assume a constant linear velocity for execution. For this particular door opening time, the robot makes it through the doors more often for $[x, y, \theta]$ models than the $[x, y]$ model. The $[x, y]$ model paths do not always go straight through the door opening, and sometimes deviate slightly due to random sampling next target nodes in the RRT at a maximum allowed distance. The s-curves generated for $[x, y, \theta]$ paths tend to be slightly more direct through the doors, so it's possible that's why execution success is greater.

When the door timing changes, we found that the success rates did not change for the higher models, but do for lower models. When the doors are open longer, the constant linear velocity

is enough for all models without velocity to successfully clear the door opening. When the door open time is less, these models fail more often.

Higher models are equally successful regardless of how long the doors are open because they increase their velocity to get through the doors in time. The acceleration models were also equally successful. We believe this is because the acceleration limit during execution for velocity models ($0.2m/s^2$), is enough to increase velocity to the amount required to clear the door opening.

In general, planning times are longer for models that contain the trailer. The trailer causes the RRT to fail more often when trying to extend nodes near the door opening. Plan times for models without velocity did not vary with the door times because models without velocity always assume the doors are open.

For models that contain velocity, we would expect the planning time to increase as the door time decreases. As the door opening time decreases there are more node expansions necessary for generating paths which causes longer planning times. This could be due to the planner failing to expand nodes more often in cases where it, by chance, chooses a slower velocity. The more often this happens, the longer it takes to find a plan. These occurrences increase when the door is open for a shorter time because the robot is required to go faster for a longer duration of the plan to be successful. We observed that the planning times for the doors open for 14 and 16 seconds were comparable, and the environment with the door open for 12 seconds had higher planning times for the higher models (and more so for those with acceleration). We believe the planning time is comparable for the 14 and 16 second doors because there is a minimum door open time where the robot's average linear velocity is enough to always be successful. If the door times continued to be less than 12 seconds the planning time would continue to increase.

6.2.3 Confusion Matrix for Evaluating Checker Versus Controller Performance

In table 6.7, we define the confusion matrix for evaluating the feasibility checker's performance. The columns show ground truth which equates to collisions for the simulated robot during execution. The rows are the feasibility checker's prediction on if the robot will collide with the environment. The false positive rate is when the feasibility checker incorrectly predicts a collision and there is no collision. The false negative rate is when the feasibility checker incorrectly predicts no collision but one occurs.

Adding the top row (True Positive + False Positive) gives the number of trials that will switch during an experiment. This answers the question of *when* our algorithm decides to switch models. The number of trials that do not switch at all is the total across the bottom row. We note that the number of switches within a given trial can be greater than one. This means that the feasibility checker may repair a part of the path, and may still detect an infeasibility further along the same reference path. Additionally, the feasibility checker could determine that the area previously repaired requires another re-plan in a higher model. Therefore, this number gives us the value of trials that will switch, but does not indicate the total number of overall switches. We will present histograms for showing the switching numbers over multiple trials.

The experiment success rate is the accumulation of values in the last column (False Positive

+ True Negative) divided by the total number of trials.

Table 6.7: The confusion matrix for comparing the performance between the feasibility checker and the robot controller. The false positive rate is when the feasibility checker incorrectly predicts a collision and there is no collision. The false negative rate is when the feasibility checker incorrectly predicts no collision but one occurs.

		Ground Truth		
		Collision	No Collision	
Prediction	Collision	True Positive	False Positive	= Number of re-plan trials.
	No Collision	False Negative	True Negative	
			= Number of successful trials.	

Recording the False Positive and False Negative Rates

To generate the false positive and false negative rate, we first generate a plan in a model. Then we use the feasibility checker to check it in the highest model and record if the check is successful. Regardless of what the feasibility checker determines, the plan is sent to the (simulated) robot for execution. We then compare the success of the executed plan to the checked plan. Their outcomes fall under four categories: both are successful (a true negative), both fail (a true positive), the checker fails when the robot executes successfully (a false positive), or lastly, the checker succeeds and the execution of the path fails (a false negative).

Confusion Matrix Results for Single Model Performance

Here we present the false positive and false negative rates averaged over all gurney environments and then a separate table for average rates for the three automatic swinging door environments.

Table 6.8: False positive and negative rates for single model performance averaged over all gurney environments.

Model	False Positive (SD)	True Negative (SD)	False Negative (SD)	True Positive (SD)
[x, y]	0.08(0.08)	0.54(0.40)	0.01(0.01)	0.37(0.39)
[x, y, θ]	0.07(0.06)	0.50(0.45)	0.01(0.03)	0.44(0.46)
[x, y, θ, θ_1]	0.14(0.20)	0.78(0.31)	0.02(0.05)	0.06(0.11)
[x, y, θ, V]	0.08(0.06)	0.53(0.40)	0.01(0.01)	0.38(0.38)
[x, y, θ, θ_1 ,V]	0.10(0.15)	0.83(0.23)	0.02(0.04)	0.05(0.08)
[x, y, θ, V, A]	0.07(0.06)	0.56(0.42)	0(0)	0.37(0.40)
[x, y, θ, θ_1, V, A]	0(0)	0.97(0.06)	0.03(0.06)	0(0)

We describe our overall false positive and negative rates shown in Table 6.8 by discussing specific gurney environments. We found that more open worlds, such as E4 and E2, had higher false positive rates for models that do not model the trailer. For E4, we believe this is because the feasibility checker thinks that the robot is going to hit the gurney that is close to the start, but it successfully clears it during execution. This could be an issue with how the feasibility checker propagates the acceleration or trailer swing from a stopped state along a lower model path that did not initially consider the trailer. One possibility being that the mass of the trailer allows the executing robot to follow the path more exactly than the feasibility checker predicts.

We also believe that false positive rates are higher for more open worlds for models without the trailer because the trailer swings less in these environments. Therefore, the trailer is directly behind the robot for much of the planning environment and most often faces close to the theta zero direction. Since the trailer is wider than the robot, it has the effect of adding padding to create more distance from obstacles. We believe paths generated without the trailer in open environments are closer to obstacles than paths generated with the trailer. The additional padding added to the feasibility checker would cause paths closer to obstacles to have higher false positive rates than those that are not.

For more constrained environments, the opposite case is true. We found for worlds E6 and E7 false positive rates were higher for models with the trailer rather than without. This could have something to do with the over arching turn the robot takes to get to the goal in these environments. Since we do not properly model mass, it is possible that the actual trailer tracks the path better than our feasibility checker because the trailer may swing less when approaching the turn (avoiding collisions) and then swings away faster than predicting for rounding the turn.

We observed that E9 has high false positives for all models and more so for models with the trailer being higher than those without. Again, the trailer seems to track better for the actual robot than what the feasibility checker predicts. This could be caused by not properly modeling the trailer's mass.

Table 6.9: False positive and false negative rates for single model performance averaged over all the swinging door environments.

Model	False Positive (SD)	True Negative (SD)	False Negative (SD)	True Positive (SD)
[x, y]	0.46(0.47)	0(0.01)	0(0)	0.54(0.48)
[x, y, θ]	0.56(0.51)	0(0.01)	0(0)	0.43(0.51)
[x, y, θ, θ_1]	0.57(0.51)	0.01(0.01)	0(0)	0.43(0.52)
[x, y, θ, V]	0.08(0.10)	0.92(0.10)	0(0)	0(0)
[x, y, θ, θ_1 ,V]	0(0)	1(0)	0(0)	0(0)
[x, y, θ, V, A]	0.10(0.15)	0.90(0.15)	0(0)	0(0)
[x, y, θ, θ_1, V, A]	0(0)	1(0)	0(0)	0(0)

Even though the lower models are more successful when the door is open for longer, the total number of trials where switching will occur (False Positives + True Negatives) does not change. The higher false positive rates for models without velocity, Table 6.9, are due to our overly conservative way of collision checking the full door swing area when the doors are determined to

be opening or closing. This is discussed in Chapter 5.3.2. Models that do not contain velocity do not properly index door swing with time. Without time, the doors are always assumed to be open. Therefore, even though the door timing changes, these lower models do not change how they plan paths, but the probability the robot successfully makes it through does. The constant linear velocity from the lower model is sent to the executing robot which successfully gets through the doors more or less often based on how long they are open. Alternatively, the feasibility detector properly indexes the time dimension (knows the doors close), but since collision checking is overly conservative (the entire door swing area is considered an obstacle) it is always predicting that the lower model will collide. The doors are collision checked as open for the door open time, then the entire swing area of the doors are collision checked as an obstacle when the doors are opening or closing, and the doors are collision checked as closed when the robot is outside the activation area. This means there is a very small window when the doors are collision checked as open for indexing by time. Since lower models do not vary their velocity and assume the doors are always open, they do not generate paths that the feasibility detector determines successful. All of this leads to very high false positive rates for models without velocity and do not consider the time dimension.

For models with velocity and without the trailer we saw the false positive rate increase as the door timing decreased. Therefore, the feasibility detector predicts more collisions than actually occur. We believe this also has to do with how a partially open door is modeled for collision checking.

6.2.4 Single Model Performance Summary

It is important note that while we organize our models hierarchically by fidelity the notion of higher fidelity does not always mean higher success rate. This is shown in the gurney environments where the lowest [x,y] model can perform better than other higher models without the trailer. It is also the case that models with added velocity and acceleration, for those without the trailer, are not strictly better than those without. The trailer model was most beneficial for execution success in the gurney environments. For the swinging door environments, models with added velocity and acceleration are more successful than those without.

Both the swinging door environments and gurney environments show higher planning times for models which include the trailer.

We expect our switching results to have plan times consistent with choosing the most successful models for the environments. For the gurney environments, this suggests that the [x, y, θ, θ_1], [x, y, θ, θ_1, V], and [x, y, θ, θ_1, V, A] will be chosen most often for switching. There could be issues with planning times for the gurney switching results as compared to the highest model because the [x, y, θ, θ_1] has a high success rate, but also seems to have much higher planning times than the highest model. For the swinging door environment, the high success rates for models with velocity and acceleration suggest that these models will be chosen most often during switching.

Planning times are also affected by our false positive rates. The false positive rates for models that also have high success rates imply that the switching algorithm could choose to switch unnecessarily to other models. For example, the gurney environment has a false positive rate of 14% for model [x, y, θ, θ_1] which suggests that there are environments where even when

choosing to re-plan with [x, y, θ, θ_1] the feasibility detector may determine that the re-planned portion of the path needs to switch to an additional model. This is also the case in the swinging door environments where both [x, y, θ, V] and [x, y, θ, V, A] have false positives which may cause unnecessary switching.

For the swinging door environments, we also see that based on the false positive rates of lower models, the feasibility detector will decide to always switch. This should in general lead to higher planning times for our switching results.

Switching success rates are affected by our false negative rates. For the gurney environment, there are false negatives present for the highest model which could lead to lower overall switching success rates as compared to the highest model. For the swinging door environment, the lack of false negatives suggests that the success rates should be consistent with the highest model.

6.3 Plan-Time Model Switching

In this section, we discuss our switching results. First, we provide results for the heuristic used to weight our re-planning tree expansions. Then we organize the section, as before, first highlighting results for average performance with our gurney environments, then highlighting some individual environment cases, and lastly presenting results for our automatic swinging door environment.

For each environment, we present results for switching among the model hierarchy. We also note that the false positive and false negative rates discussed starting in Section 6.2.3 directly affect the success and planning times of our switching results. False negatives affect our switching success rates. Our goal is to produce robust plans so preference is to minimize the false negative rate at the expense of false positives. False positives affect our switching planning times, as it indicates times our approach chooses to unnecessarily switch to a different model.

We present tables for execution success, plan time, path length, and number of switches. The number of switches gives an indication of the number of times the checker signals that the current underlying reference path is insufficient for execution. We also include analysis on how our planning times compare with single model runs.

We note that the choice of RRT re-plan iterations per trial is a subject of future work. As stated, we generate 20 RRT path iterations, and choose the shortest to execute for each single model trial. This is for path consistency over trials, and for an overall smoother reference path for our switching algorithm. For example, we found that using two RRT iterations for each single model trial resulted in longer paths, and increased false negative rates for higher acceleration models. We also observed that switching success rates were lower when the initial path, generated in the lower model, was chosen only after two iterations. Therefore, we believe paths which use a low number of initial iterations for path consistency produce more trailer swing and are more difficult for the robot to follow.

We did experiments for model switching each consisting of 100 trials. Each trial starts with generating a path in the initial [x,y] model for 20 iterations and then choosing the shortest. Then for each model switch the re-planning cycle multi-tree re-plans for 2 iterations, and keeping the shortest. The resulting mean and standard deviations are recorded and displayed for different environments. The choice of re-plan iterations was chosen because it was determined sufficient.

In other words, we used two re-plan iterations for every detected model switch within a trial because increasing the re-plan iteration did not improve success rates.

Since single model runs use 20 planning iterations for each trial, and each switching trial has 20 initial RRT iterations and then 2 re-plan iterations per switch, we did not think it was fair to compare these plan times directly. Therefore, we normalize the plan time to better compare single model trials with our switching trials. We normalize the times by dividing each single model trial by 20, and recording the mean and standard deviation. Then for each switching trial we divide the number of initial RRT iterations by 20 and each re-plan, required for switching, by 2. We retain the running total per trial and record the mean and standard deviation. These results are displayed in the respective normalized planning time tables. For each table, we also discuss if the time differences are statistically significant

6.3.1 Multi-Tree Weighting Heuristic for Switch Tests

The path is repaired by re-planning from multiple start trees towards a corresponding set of intermediate goals. It is possible to probabilistically weight the node expansion of each start tree and the sampling for which intermediate goal the tree is re-planning towards. This is useful for more efficiently searching the planning space. The different tree weighting heuristics are described in Chapter 5. The first heuristic weights all re-plan trees by an inverse distance metric. This encourages the expansion of re-plan trees closer to the detected infeasibility for favoring re-use of the original reference path. The second variation weights the re-plan tree expansion by an inverse distance metric and then adds planning time until the weight for expanding all trees becomes uniform. This is to help prevent cases where node expansions may have difficulty for re-plan trees that are too close to the infeasibility such as in a tight hallway. The third switching heuristic uses a logistic function to favor re-plan tree start waypoints and intermediate goals to be further from obstacles. This is based on the observation that it is not known how close the infeasibility is to other reference path waypoints, but also it is unknown if other reference path waypoints are close to obstacles. Node expansion is difficult for waypoints closer to obstacles. These heuristics build into a combined final heuristic. Our initial testing included varying each model switching trial by incrementally testing each tree weighting heuristic.

Table 6.10 shows results for different variants of our multi-tree weighting heuristic averaged over all of our automatic swinging door environments. This environment demonstrates statically significantly better performance for the combined heuristic over the other variants, ($t(300)=4.28$, $p<0.00001$, for a paired t-test between heuristic 1 and heuristic 3). Although, there was no statistical significant difference between the first two variations of the heuristic.

We believe this environment demonstrates the best use of this heuristic because the swinging door is centered in the environment. This weights the expansion for the start of re-plan trees further from the door (the environment's obstacle) towards intermediate goals, with a weighted sampling, also further from the door. Re-planning further from the door opening allowed for faster planning times because the RRT node expansions did not get stuck as often.

For these weighting heuristics we found that the final heuristic which combines the pieces from all previous heuristics had strictly better planning times than those without. We also found that using the combined heuristic never performed worse over environments. It always achieved planning times that either showed no statistical significant difference in the environment or pro-

Table 6.10: Planning times for different heuristic weighting averaged over all the swinging door environments.

Experiment Type	Plan Time (SD) (s)
Heuristic 1	16.93(16.38)
Heuristic 2	15.41(14.34)
Heuristic 3	12.28(11.79)

duced a lower average mean planning time that was statistically significant. Therefore, we chose to run all of our tests that involved model switching with the combined heuristic.

6.3.2 Switching for Gurney Environments

We explain results for average switching performance over all gurney environments and then highlight results for specific environments.

Average Performance

Table 6.11: Switching Results for all gurney environments.

Models Switched	Success (SD)	Plan Time (SD) (s)	Path Length (SD) (m)
All Models	0.94(0.12)	9.88(26.21)	6.39(1.19)
Switching Lowest and Highest Model	0.93(0.11)	2.14(5.53)	6.48(0.96)

The statistic for the number of trials that switch describes the mean over all gurney environments for the percentage of trials that switch. The standard deviation for the trials that switch is the standard deviation over the average number of trials that switch, so for the gurney environments this is divided by the number of worlds which is 10.

For Table 6.11, we see statistically significant improvement in planning time when switching between just the lowest and highest model versus all models. ($t(1000) = 9.33$, $p<0.00001$) This is due to gurney environments (E5 and E7) that cause large planning times for lower models, and because our model selector uses Breadth First Search it spends lots of time planning in these lower models. This is an example of where it would be beneficial to have a more informed search which switches more directly to the model which has the most efficient plan time and execution success for a given environment.

If we re-analyze the results without environments E5 and E7, shown in Table 6.12, we see that as hypothesized switching with the complete model hierarchy is better. The average planning time for switching with all models is now statistically significantly better than that of switching between just the lowest and highest model where $t(800) = 2.30$, $p<0.021$. Note this t-test was for 200 less trials because environments E5 and E7 are removed.

Table 6.12: Switching Results for all gurney environments except E5 and E7.

Models Switched	Success (SD)	Plan Time (SD) (s)
All Models	0.99(0.01)	0.86(1.80)
Switching Lowest and Highest Model	0.98(0.03)	1.06(2.20)

The success rates over all gurney environments for switching as compared to that of the highest model is not statistically significantly different. We see that the average success rate for model switching is lower than the highest model but has a large standard deviation, in Table 6.11. We believe this is due to the high false negative rates that are observed in two of the gurney environments. We have confirmed this hypothesis by showing in Table 6.12 that success rates are better and the variance is lower without environments E5 and E7.

Table 6.13: Normalized plan times for gurney environments.

Experiment Type	Plan Time (SD) (s)
[x, y, θ, θ_1, V, A]	1.45(1.77)
Switching all Models	4.12(11.87)
Switching Lowest and Highest Model	0.94(2.52)

In Table 6.13, we believe our normalized planning time for switching among all models was higher on average for all gurney environments due to outlier environments (E5 and E7) where lower models take higher to plan than the highest model causing a very large variance. This is further supported by our results where we switch only between the highest and lowest model. The mean plan time for switching between these two models is statistically significantly lower than both planning in the highest model alone and switching between all models. This is because exclusively switching between these two models avoids being overly conservative with switching to a lower model with longer planning times first.

Table 6.14: Normalized plan times for gurney environments without E5 and E7.

Experiment Type	Plan Time (SD) (s)
[x, y, θ, θ_1, V, A]	0.92(0.68)
Switching all Models	0.37(0.86)
Switching Lowest and Highest Model	0.48(1.09)

Once again we removed environments E5 and E7 to see if planning times would be consistent with the hypothesis that these outlier environments contribute to higher than expected

planning times for the full hierarchy switching results. These two environments are removed when averaging the normalized planning time results for the highest model, switching among all models, and switching between just the lowest and highest model. These results are shown in Table 6.14. We see that planning times for switching with the full model hierarchy is statistically significantly lower than the highest model, ($t(800) = 15.2$, $p<0.000001$), and also lower than just switching between the lowest and highest model ($t(800) = 2.59$, $p<0.01$). Switching between the lowest and highest model switches to the highest model unnecessarily because for most of our tested environments switching among the full model hierarchy does better. We also note that the plan time for the lowest and highest model is cut in half when removing those two environments which also suggests that these environments are more difficult for even the highest model.

Performance for Specific Gurney Environments

We discuss additional switching results by highlighting our E0 gurney environment from Figure 6.1, an open gurney environment E4, and an environment that performed less well E5.

We found that for our E0 gurney world we achieved lower on average planning times with our switching algorithm as compared to the highest model. The difference is also statistically significant ($t(100) = 3.97$, $p<0.0001$). These normalized values are shown in Table 6.15.

Table 6.15: Normalized plan times for gurney environment E0.

Model	Plan Time(SD) (s)
[x, y, θ, θ_1, V, A]	0.91(0.29)
All Models	0.59(0.74)
Switching Lowest and Highest Model	0.83(1.14)

More open environments, such as E8 and E4, saw significant improvements in planning time for switching between multiple models compared to the highest model. This is because the lower models were very successful and the approach did not choose to switch very often. When it did switch models, it often did so only once. Planning times for many of the environments were comparable to planning in the lowest model from start to goal. Table 6.16 shows a more open environment, E4, where there is a significantly reduced planning time for switching among the model hierarchy as compared to just switching between the lowest and highest model ($t(100) = 4.91$, $p<0.000003$).

An example of where switching does not improve planning times over the highest model is shown for environment E5. Planning times are shown in Table 6.17 for comparing the highest model with switching experiments. While planning times for switching are larger than the highest model, we did observe that time for switching between just the lowest and highest model was statistically significantly lower than the highest model. ($t(100) = 4.12$, $p<0.0001$). E5 is an environment where a lower model with high success rates, and high planning times compared to the highest model, increases overall switching time. This is because the model selector chooses this lower fidelity model, [x,y,θ,θ_1], before the highest model causing higher planning times.

Table 6.16: Normalized plan times for gurney environment E4.

Model	Plan Time(SD) (s)
[x, y, θ, θ_1, V, A]	2.21(0.67)
All Models	0.05(0.12)
Switching Lowest and Highest Model	0.94(1.82)

Table 6.17: Normalized plan times for gurney environment E5.

Model	Plan Time(SD) (s)
[x, y, θ, θ_1, V, A]	1.67(0.27)
All Models	37.23(18.04)
Switching Lowest and Highest Model	1.20(0.89)

Figure 6.5 shows the number of switches for all gurney environments. About half the trials do not switch at all. Then most of the remaining trials choose to switch just once where in most cases the model chosen to switch to was the first model with trailer heading.

6.3.3 Switching for Automatic Swinging Door Environments

Average Performance

Our switching results for the swinging door environment show larger planning times than switching between the lowest and highest model, and a very high variance, Table 6.18. All trials switch due to the feasibility detector always predicting a collision in the lowest model. This is due to extremely high false positive rates for the swinging door environment that has doors open the longest, and the high true positive rate for the swinging door environment that has the doors open the shortest.

Table 6.18: Average Switching Results over all swinging door environments.

Models Switched	Success (SD)	Plan Time (SD) (s)	Path Length (SD) (m)	Trials That Switch
All Models	0.98(0.2)	12.28(11.79)	5.64(1.41)	1(0)
Switching Lowest and Highest Model	1(0)	3.32(3.93)	5.27(0.65)	1(0)

We re-iterate here, as in Section 6.2.3, that these false positive rates are high for our lowest models because of how we collision check in our highest model. We are being too conservative with how we determine how much of the door swing area to collision check against for when the

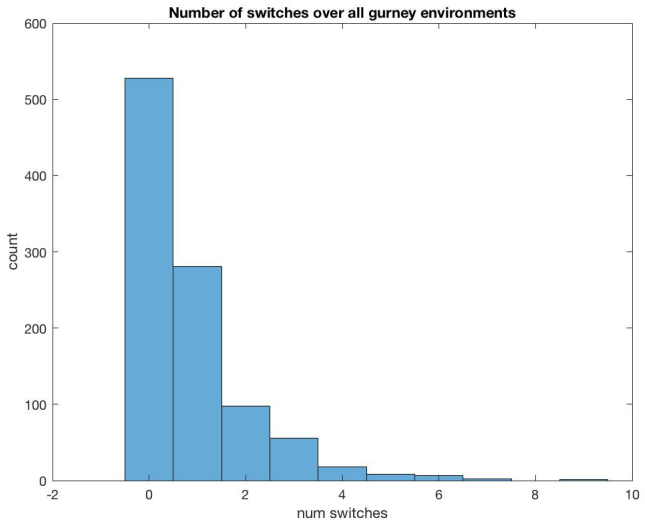


Figure 6.5: A histogram showing the number of switches for 100 trials of each gurney environment.

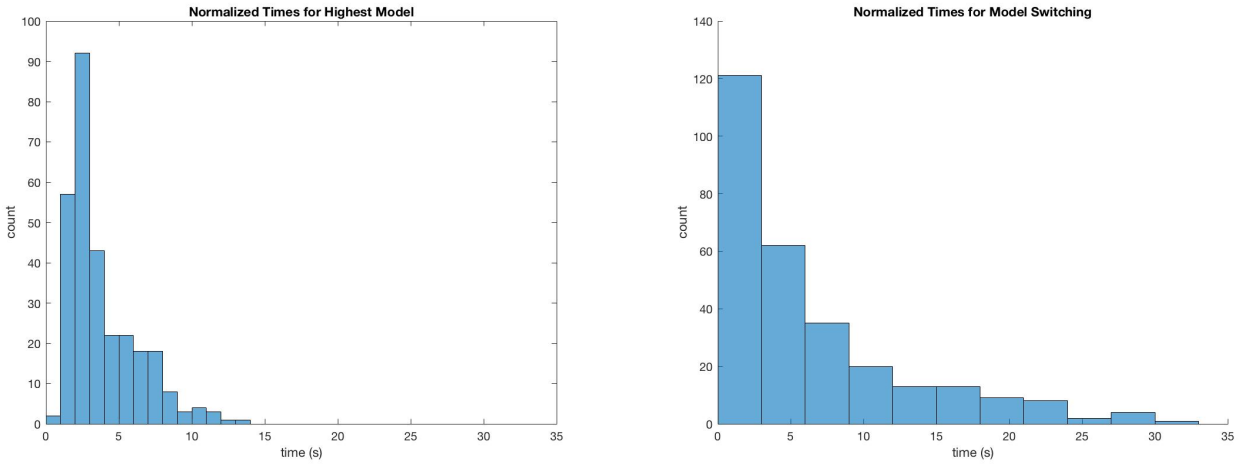
doors are opening and closing. An alternative approach is to discretize the door swing but we found this lead to a higher false negative rate where the highest model does not properly detect that the door can swing through an area. So, invalid paths are created where the robot can teleport around partially open doors which fail when followed during execution.

Table 6.19: Normalized plan times for the automatic swinging doors environment.

Model	Plan Time(SD) (s)
[x, y, θ, θ_1, V, A]	3.88(2.44)
Switching all Models	6.35(6.62)
Switching Lowest and Highest Model	1.64(1.98)

In Table 6.19, we see that while the planning times for switching among all models is larger than the normalized time of the highest model, the standard deviation is also high. This large difference in variance is show in Figure 6.6. Again this is due to our switching algorithm switching higher than necessary because of high false positive rates. More specifically, this is caused by the false positive values that exist for both [x, y, θ, V] and [x, y, θ, V, A]. These two models have high success rates so they are chosen often by the switching algorithm, and if they also have false positives it is possible that a re-planned portion of the path with this model will be detected as needing to switch again which causes a higher model (with even longer planning times) to be re-planned unnecessarily. It is also the case that false positive rates increase for these two models, causing our switching algorithm to switch more often per trial, as the door time decreases. This causes switching to occur for more models than necessary. This can be seen in the histograms for each door opening times in Figure 6.7(c). We do see improvement in the average plan time

for switching between the highest and lowest model. This shows that if we did not switch unnecessarily between two models with equivalent success rates, in our 12-second door environment, our average planning time would have been less.



(a) The normalized plan times for the highest model are more concentrated.

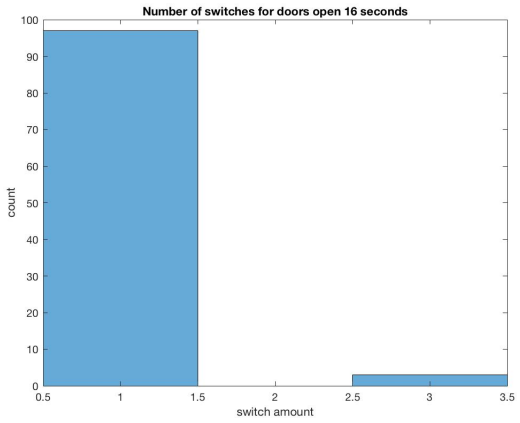
(b) The normalized plan time for switching between all models contains more outliers.

Figure 6.6: The two histograms show that the planning times for switching have a greater spread than the highest model.

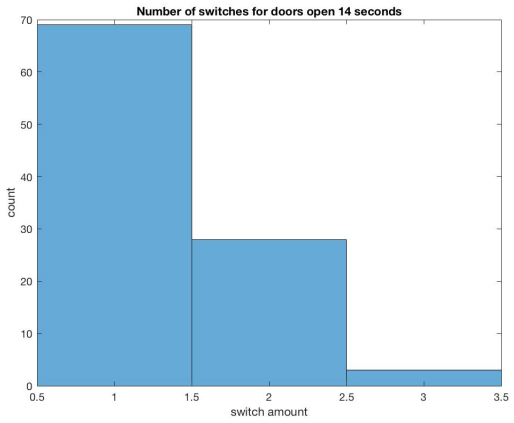
We also include histograms for the number of switches per trial for each of our automatic swinging door environments as shown in Figure 6.7. We observed that when the door time gets less the number of switches per trial actually increases. This is due to a high false positive rate which emerges for velocity models in the environment with the shortest door open time (12 seconds). This false positive rate indicates that partial paths repaired in these models are incorrectly detected as having a collision which causes an additional unnecessary switch to an even higher model. A false positive rate also exists for these models in the 14 second door environment but it was much less. The 16 second door environment had no false positives for higher models and properly chose to switch only once more than 95% of the trials, Figure 6.7 (a). We believe the false positive rate increases for these models because the feasibility detector indexes the door time and overly conservatively collision checks against the entire door swing area which occurs more often when the RRT has trouble extending nodes in this narrow passage when the doors are open less.

6.4 Plan-Time Model Switching with Uncertainty

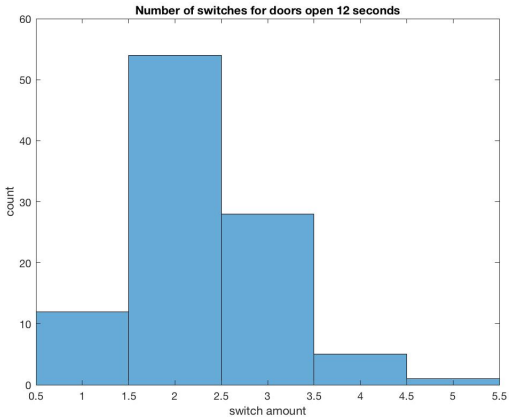
In this section we compare switching results with and without additional uncertainty models. We expect that adding uncertainty to models encourages the switching approach to choose to plan with lower models more often further reducing planning times. We show that this is the case for the majority of environments.



(a) Switches swinging environment 16 (100 trials).



(b) Switches swinging environment 14 (100 trials).



(c) Switches swinging door environment 12 (100 trials).

Figure 6.7: The number of switches increases as the door time decreases due to an increase in false positives with higher models.

6.4.1 Switching for Gurney Environments

Average Performance

As expected, Table 6.20 shows the difference between success rates with adding uncertainty models was not statistically significant. The difference between path length is statistically significant with paths for switching with uncertainty models being greater than without ($t(1000) = 5.97$, $p<0.00001$). This is because adding padding moves the robot further from obstacles, so when it turns it cannot cut corners as much and generates longer paths. Unexpectedly, switching among all models with uncertainty has higher planning time than without. We believe this is due to some outlier environments where the robot can get stuck planning in a lower [x,y,θ] model. Switching just between the highest and lowest model demonstrate lower planning times as seen previously.

Table 6.21 shows the normalized planning times for the highest model averaged over all

Table 6.20: Switching Results for all gurney environments with added uncertainty models.

Models Switched	Success (SD)	Plan Time (SD) (s)	Path Length (SD) (m)	Trials That Switch
All Models	0.94(0.12)	9.88(26.21)	6.39(1.19)	0.48(0.38)
All Models with Uncertainty	0.93(0.13)	12.42(34.72)	6.58(1.24)	0.42(0.41)
Switching Lowest and Highest Model	0.93(0.11)	2.14(5.53)	6.48(0.96)	0.43(0.38)

gurney environments with switching and added uncertainty. The normalized planning times show a similar trend as previously described. We compare this table directly to the following table which removes environments E5 and E7 (Table 6.22). In Table 6.21, we hypothesize that adding uncertainty when switching in the full model hierarchy increases planning time (with a higher variance) due to outlier environments which cause the switching algorithm to select lower models (with higher planning times than the highest model) even more often. Table 6.22 confirms this result, as we see statistically significantly lower planning times for switching among all models with uncertainty than both full switching, t(800)=6.12,$p<0.000001$, and switching of just the lowest and highest model, t(800) = 7.42, $p<0.000001$, when these outlier environments are removed.

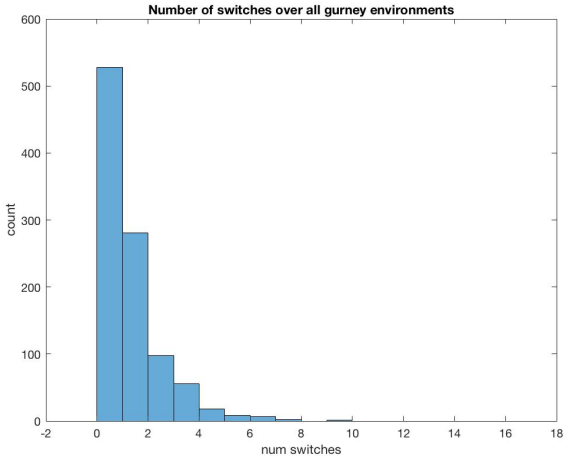
Table 6.21: Switching Results for all gurney environments with added uncertainty models and normalized planning times.

Models Switched	Success (SD)	Plan Time (SD) (s)
[x, y, θ, θ_1, V, A]	0.97(0.06)	1.45(1.77)
All Models	0.94(0.12)	4.12(11.87)
All Models with Uncertainty	0.93(0.13)	5.03(17.91)
Switching Lowest and Highest Model	0.93(0.11)	0.94(2.52)

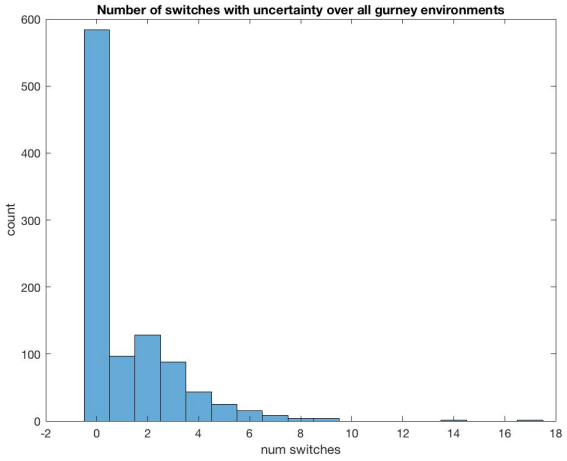
We also see how the number of switches increases per trial with adding uncertainty as shown in Figure 6.8. This is expected because the ability to add uncertainty models causes lower models with additional padding to be used more often and it is possible that for particular environments these lower models are not sufficient for going through particular parts of the space. For example, perhaps an $[x, y]$ model with additional padding is often chose for planning, but in particular areas with gurneys the robot will only be successful when considering trailer heading. This is a case where the algorithm may always choose a lower padding model before considering the trailer heading model which creates more switches on average.

Table 6.22: Switching Results for all gurney environments with added uncertainty models and normalized planning times except E4 and E5.

Models Switched	Success (SD)	Plan Time (SD) (s)
[x, y, θ, θ_1, V, A]	0.99(0.01)	0.92(0.68)
All Models	0.99(0.01)	0.37(0.86)
All Models with Uncertainty	0.99(0.01)	0.17(0.63)
Switching Lowest and Highest Model	0.98(0.03)	0.49(1.09)



(a) The number of switches for all gurney environments.



(b) The number of switches for adding uncertainty for all gurney environments.

Figure 6.8: The two histograms show that the spread for the number of switches increases when considering additional uncertainty models.

Performance for Specific Environments

Our E0 world demonstrates even more efficient plan-time performance over the highest model for switching with models that also include uncertainty as shown in Table 6.23. For this particular environment, lower $[x, y]$ models with added padding were often sufficient for planning a feasible path, so higher models were rarely needed. We found that average planning time for switching with uncertainty was statistically significantly lower than switching without uncertainty models. ($t(100) = 5.92$, $p<0.00001$).

As was the case for our switching experiments without uncertainty, more open environments, such as E8, saw significant improvements in planning time for switching between multiple models compared to the highest model. Again this is because of the high success rates for the lowest model, so the approach chose to switch rarely. Therefore, there was no added benefit to adding models with uncertainty. Results were comparable for switching with and without uncertainty

Table 6.23: Normalized plan times for gurney environment E0.

Model	Plan Time(SD) (s)
[x, y, θ, θ_1, V, A]	0.91(0.29)
All Models	0.59(0.74)
All Models with Uncertainty	0.11(1.13)
Switching Lowest and Highest Model	0.83(1.14)

for these environments.

In general, we expected that planning times would be lower for switching among all model and uncertainty as compared to just switching between the lowest and the highest model. The caveat is for environments like E5, where normalized times are shown in Table 6.24. This environment is an example where the implementation of the [x,y,θ] and [x, y, θ, θ_1] models for this environment cause these lower models to have longer planning times than the highest model. The high success rates for the [x, y, θ, θ_1] model also cause it to be chosen by the model selector more often. This is why experiments which only use the lowest model and highest model have lower average planning times than the full set of models because [x, y, θ, θ_1] is no longer a possible model to choose from (It is not the lower or highest model). This further motivates the need for a more informed search so that the [x, y, θ, θ_1] model is skipped due to it having larger planning times than an equivalently successful higher fidelity model with lower planning time.

Table 6.24: Normalized plan times for gurney environment E5.

Model	Plan Time(SD) (s)
[x, y, θ, θ_1, V, A]	1.67(0.27)
All Models	37.23(18.04)
All Models with Uncertainty	41.45(22.02)
Switching Lowest and Highest Model	1.20(0.89)

6.4.2 Switching for Automatic Swinging Door Environment

The swinging door environments see no benefit from adding robot footprint padding. This demonstrates correct usage of the algorithm which chooses the correct model, and does not switch to added uncertainty if it's not useful. This result also suggests the type of uncertainty matters. For example, adding padding to the robot's footprint addresses uncertainty in the robot's motion and obstacle placement. (spatial uncertainty). Footprint padding uncertainty is not as useful in (have as much impact) in a world that is more concerned about uncertainty in the

time dimension. Having uncertainty in the time dimension for adapting the duration of collision checking door swing would have been better for this type of world.

6.4.3 Plan-Time Model Switching with Uncertainty Discussion

The algorithm correctly chooses to not use models with uncertainty when they do not provide added benefit, but our approach is still overly conservative by favoring models with uncertainty in our Breadth First Search. This is most evident in environments where adding padding made it difficult to find paths. We believe there is benefit to including uncertainty models but there should be further evaluation to determine the point at which too much padding for a model actually negates its benefit for a specific environment. For example, one example is environment E9 where a gurney is close to the goal. Our approach limits the padding to only the amount that ensures a valid path can be found, therefore this particular environment did not benefit from additional padding. This is because either uncertainty models were not chosen, due to no valid re-plan trees existing when added padding invalidates the goal, or the few uncertainty models that were valid actually caused increased planning times. We believe outliers in Figure 6.8 (b) are caused by these types of environments. Even though our model selector correctly goes to the most appropriate model, the model selection is overly conservative by model selecting uncertainty models before higher fidelity models even in these cases. A more informed model selection would be useful in this scenario as well.

Another example of cases where using uncertainty is an overly conservative approach, is when models with uncertainty create paths in areas of the environment where higher models have difficulty finding a plan. This already occurs when switching between models without uncertainty, but additional uncertainty models can exacerbate the situation by planning more in lower fidelity models which create these scenarios. We do note that this is highly dependent on the environment. For example, [x,y] paths that have been checked and fail in a tight hallway may find a successful re-planned path with uncertainty that creates a longer route around the top gurneys. Yet, this is another location that plans generated in higher models rarely considers. Unfortunately, for gurney environment E0 this will never be successful since the area above the gurneys is too tight for a turning trailer to fit. If a path above the gurney is chosen as the next partial plan repair it will always have to be repaired with a model that contains the trailer. This also causes the variance for plan times to be higher for switching with uncertainty in this particular environment. This is an example of how planning times could improve if there was a way to go directly to the appropriate repair model (with informed model selection) rather than testing a lower uncertainty model first.

6.5 Results Summary

We discuss single model performance and false positive and false negative rates for our single models which help evaluate our switching results. We then present switching results for both the gurney and swinging door environments. We evaluate results averaging over all gurney environments and all swinging door environments. We also highlight specific environments to further explain our results.

The experiments show how our approach generalizes over non-uniform environments. We show that the majority of our environments benefit from switching with the entire model hierarchy, but that for particular environments planning times are longer due to implementation details and an overly conservative switching design. For the gurney environments, we show how adding uncertainty padding models can improve planning times and discuss cases where more padding can cause longer planning times. For our swinging door environments, we conclude that adding uncertainty is not beneficial for these particular environments.

Chapter 7

Future Work

In this chapter we discuss improvements with our approach which include improving the highest model used during feasibility detection, Section 7.1, improving planning search with adaptive sampling, Section 7.2, and better representations for uncertainty, Section 7.3. We also discuss some ideas for establishing more formal guarantees for our approach in Section 7.4. Our complete system would also need to consider execution time extensions shown in Section 7.6. The system would start with our work for generating an initial executable path. Then the system would close the loop by also re-planning during execution. Model selection for both plan-time and execution time could start the same. Extensions to the model selection process would include a more informed approach that depends on previous experience, Section 7.5. During execution, it may be possible to also anticipate dynamic obstacles as described in Section 7.7.

To introduce these future directions, we illustrate an example for a hospital robot which must prioritize recovery. For example, a hospital robot traveling through the environment delivering medication may commonly access models for navigating a long hallway and then entering an elevator. There could be an unanticipated day where the janitor leaves a mop bucket near the elevator entrance. The robot may try to approach the entrance in the same way it always has and constantly fail to enter the elevator. The amount of time the robot spends trying to correct itself is important. If the recovery process takes too long the state of the robot might be so bad that no recovery is possible. This could occur when recovery actions taken by the robot causes the robot to become more stuck. This could also occur from conditions outside the robot's control. This includes scenarios like: repeating the recovery process for so long it runs out of battery, or prompting a person to intervene because they assume it's broken. In the worst case scenario, a robot that consistently fails could cause a person to place it in an unrecoverable state, such as being turned off and stored.

Recovery is more expensive as recovery attempts increase because the robot could re-iterate the same task causing it to get more stuck than it was initially, or if the robot is doing something unusual for long enough it may cause human intervention. Based on the example we imagine that multiple models could help for allowing the robot to try increasingly complex models. This requires using models at execution time. Additionally, the time it takes to recognize a reliable recovery strategy and generate that recovery strategy is paramount. We believe our initial plan-time work which focuses on efficient planning is helpful for future work that re-plans recovery paths quickly. As the recovery time of the robot gets progressively worse the recovery strategy

would change. Initially the robot could try something with simple model switching in which it is less informed, the next step would be to have a more informed intuition of what model is applicable based on past experience which requires previous data Section 7.5, and if all these recoveries fail the robot would resort to human intervention.

Human intervention and teleoperation are a very expensive recovery option. The ability to explain and properly diagnose how the robot got into a situation that disrupted autonomy is important for the robot's ability to adapt and not repeat the same failure. Even in this undesirable scenario we believe that models are important for explaining to a user the robot's current state. Models are also important for more informed actions for the robot to know not only when to give up in its own recovery process but to also be able to signal to a user that it requires help. For recognizing recovery issues quickly, we believe that the use of multiple models is helpful for diagnosing unexplained robot behaviors for operators and assisting in the robot recovery process. We touch on this idea in Section 7.8.

7.1 Improving Feasibility Checking in the Highest Model

Our work has shown that how well the feasibility checker matches how the robot moves in the world directly impacts the effectiveness of our switching algorithm. As stated previously, false positive rates can cause our approach to switch more than necessary adding unnecessary planning time, and false negatives affect the execution success rates. This is why the model used for feasibility checking needs improving. One way to better match the robot's real world interactions is by increasing the model fidelity used for feasibility checking. This would include modeling trailer dynamics. We refer to work done in [12], where a system of equations for the trailer now include mass and inertia parameters. An improved model could also allow the padding for feasibility checking to vary. For example, the padding for checking along the path could change as uncertainty grows with time.

Alternatively, it may be very difficult to hand-tune a model that properly accounts for the robot's execution motion. Therefore, it may be possible to better approximate the highest model for feasibility checking through unsupervised learning. In one case, a very good optimized trajectory could act as a guide for feasibility checking and represent the most accurate model. Another technique would be to combine the model with learning. The hand-tuned model could provide the learner with initial assumptions of the world, and learning techniques could be applied to improve upon it.

7.2 Adaptive Sampling for Smarter Planning Search

Future work could be done for investigating how to improve planning time by better sampling target points. Adaptive sampling in configuration space has been done to speed up planning times [106]. Similarly, we could consider adaptive sampling that changes based on the model fidelity we are planning in. For example, certain models may have more difficulty in particular types of environments, and it would be useful to change how targets are sampled in these spaces to save planning time. This includes favoring the expansion of nodes to areas where it is easier for

RRTs to find paths (outside of tight spaces). Related work in multi-robot planning [56] favors the expansion of different nodes such that different "classes" of paths are selected based on a weighted cost. Weighting the edges allows higher penalties to be placed on expansions that cause the search to stall which encourages an exploration of more open parts of the environment. One possible future direction is to penalize edge expansions during the planning search that cause models to go to undesirable locations of the search space. This includes areas of the search space that may be difficult for higher models.

Another possibility is to use previous search information to improve the algorithm such as using the waypoints of previous searches as target points. Our current work generates plans, and uses multi-tree re-planning to generate partial path repairs. After we find a plan, we throw away previous searches done during the node expansion process. Previous search information could be utilized for providing additional target waypoints to guide re-plan repairs. For example, re-planning in higher models could also consider connecting to the previous search space of lower models. This would allow a finer resolution of re-planning start trees and intermediate goals to plan with rather than forcing re-plans just along the lower model path. This may be beneficial in a highly complex environment where having many more waypoints to plan from and towards gives a more thorough search of the space than the model path provides. Forcing higher models to generate plans guided along lower model reference paths could increase planning time by guiding higher models to plan in difficult parts of the environment it would not otherwise sample. Maintaining previous higher model searches could help guide future sampling for knowing when particular higher models are more or less useful for repairing a path in a particular area of the environment.

Further research could also investigate when searching in a higher model is no longer beneficial. This means there could be a point during the re-planning process where continuing to search in the higher model is not contributing more information and is actually increasing computation resource. For example, it may be useful to search in a higher model until the search has switched to a different homotopy class, and once this has occurred go back to searching in a lower model. By considering previous tree searches, we can reason more intelligently about the benefits of searching in models to preemptively end the search process and further save planning time.

7.3 Representing Uncertainty

It would be possible to represent uncertainty in different forms. This includes the ability to add uncertainty to different dimensions of the configuration state. Three possible forms include: as a uniform set-bounded distribution over states or particular dimensions, as a parametric Gaussian distribution, or as a non-parametric particle distribution. Note that each form increases the previous representation's complexity.

As an extension, the amount of padding could also vary along a particular dimension. If a robot's straight-line motions are more accurate than turns, uncertainty can exist as a bounding space only in the theta direction. One disadvantage of representing uncertainty uniformly is the generality in reasoning about uncertainty in this way. For example, representing uncertainty as padding covers both state and execution information, acting as a catch all, making it difficult to know exactly what information is known.

Related work has represented uncertainty as a parametric Gaussian distribution [80], [24], [67]. Again, for tasks that need to save computation time marginalizing hypothesis space (belief) as a Gaussian is more tractable than reasoning about separate hypotheses. This is effective in domains with simple unimodal observation dynamics. For instance, one might assume that the object a robot is going to retrieve has a normally distributed location. This implies that the object is only slightly shifted around some mean location.

In more complex domains, uncertainty can also be reasoned about probabilistically as a non-parametric distribution. For instance, [79] represents uncertainty using sample particles that act as possible hypotheses in the space. Similarly, [65] uses a particle filter to represent a state's belief at particular times. When uncertainty is reasoned about explicitly, in a distributed belief space, it increases computation time but is more constrained to the correct underlying model. One such example is with localization. If the robot's localization information becomes so bad that there are multiple distinct areas where the robot could be, a single Gaussian or uniform rectangle would be unable to capture this information.

Related work in capturing uncertainty in the belief space is often formulated as a Partially Observable Markov Decision Process (POMDP) problem. Here the state space observability is represented as a belief over a distribution of states, and action uncertainty is captured in the model's state transition probabilities. Solving POMDPs exactly is an intractable problem [74]. Rather than use a POMDP and assume the entire state space is unobservable [70] uses a mixed observability approach. They assume some components of the state are observable while others are not. This idea motivates applying uncertainty to different parts of the model space (like single dimensions or specific state features).

The organization of models with different types of uncertainty would grow from uniform to parametric to nonparametric. In statistics, the probability integral transformation states that data values, modeled as random variables, from any given continuous distribution can be converted to random variables having a uniform distribution. This suggests that under certain assumptions bounded sets may be translated into Gaussian distributions. Similarly, parametric distributions could transform into a higher fidelity nonparametric form.

7.4 Expansions for Theoretical Guarantees

We use this section to briefly describe ideas for expanding the provable guarantees of our approach. The following subsections touch briefly on ideas for discussing guarantees related to a model's success rates, a bound on the system's planning time, and lastly a more formal definition for model translation.

7.4.1 Guarantees for a Best Model

In this section we discuss ideas for providing more formal guarantees on how specific models may be successful over areas of the environment. This may be able to be represented as a notion of regret with respect to being able to evaluate if a model chosen for re-planning an infeasible portion of a path was the best model to use in hindsight. Regret is often characterized in machine learning domains. The discussion of minimizing regret refers to reducing the number of actions,

in our cases which models were necessary for re-planning, by deciding in hindsight if there was a better model choice. Minimizing regret minimizes the idea of sub-optimal actions. It is difficult to formalize a notion of optimality in our framework, but we believe characterizing regret could be a first step to discussing a best model choice.

We note that the *cost* of an optimal plan that reaches a goal region can be characterized in many ways. Often it is described as a function based on the plan's length and the vehicle's speed [47]. In our case, we would like to assess the cost of the use of a model when it is chosen for re-planning. While it may be possible to discuss the cost in terms of path length, planning time, and execution time, we are most concerned with robustness and how successful a planned path will be for allowing the robot to execute collision free through the environment. Therefore, the notion of success is a bit more difficult to capture. If it is possible to discuss the cost of a model's contribution towards an executable path, it may also be possible to discuss the inverse,1-cost, which represents the reward. If we are trying to estimate an unknown reward distribution between multiple models to determine which one would be best for re-planning with next it may be possible to formulate this as a multi-armed bandit problem `en.wikipedia.org/wiki/Multi-armed_bandit`. The idea would be to maximize the reward of an unknown distribution by performing a tradeoff between exploiting the model with the highest payoff versus exploring to get more information about the expected payoff. This could be another learning approach for estimating which models are most beneficial for a particular area of the environment. This also could be a way to frame the model selection part of the approach where it is possible to discuss bounds on the payoff for particular models based on what previous models have been selected (bandit arm selection) or optimality in terms of converging with model choice towards a maximum expected reward.

Lastly, it may be possible to discuss a bound on a model's success by leveraging its position in the model hierarchy. Although, we have seen that sometimes higher fidelity does not always equate to better model success it may be possible to still hypothesize that neighboring models in the model graph will perform with similar success. This may allow discussing bounds on success where lower or higher models that are in a pre-defined neighborhood of the current model in the hierarchy limit how successful the model is in a particular environment.

7.4.2 Guarantees for System Planning Time

The planning time for the system could initially be described by considering what is the worst case. We recognize that this is very difficult to conjecture since it is possible for models to re-plan with an infinite amount of node expansions. For discrete space planners, it my be possible to discuss the bound on the search of the space based on a branching factor for that particular model. Randomized algorithms, such as sample based planners, bound the computation time in order to avoid infinite loops. Therefore, perhaps it is sufficient to discuss the worst case in terms of the max node expansions, for a particular model, before a planner times out. Assuming that it is possible to determine the maximum number of node expansions generated during a re-plan, then we can discuss planning times with respect to the highest model. This assumes that the highest fidelity model takes the longest time to plan in. We will refer to these max expansions as q.

The worst case for planning time would then be if the lowest fidelity model takes q_{lowest} to

plan an initial path, and each edge (between waypoints) gets re-planned in every model with each taking the max time $q_{highest}$ up to the highest model. The planning time for planning between only two waypoint would need to be discussed for each model. If planning between two waypoints represents the time for planning an edge, then each model would include an edge planning time e_x where x is the number of the model used for planning an edge and we will define m as the total number of models. If there are n waypoints, then there are $n-1$ edges. One model would then take $t_x = e_x \times n-1$ time to re-plan for one expansion. In the worst case, the planning time is then $t_{worst} = \sum_0^m e_x \times n - 1 \times q_x$.

It may be possible to also assume that in an adversarial designed (worst case) system each model takes as long as the highest model. This would represent the upper bound. We note that if a model takes longer than the highest model it would be removed since we will assume that the highest model is the worst case for model planning time. Therefore, the upper bound could be, $t_{upper} = m \times n - 1 \times q_{highest}$, the number of models times the number of edges times the max expansions for the highest model. This value would potentially also need to be multiplied by some constant number of threads to represent the time lost to other threads when doing concurrent multi-tree re-planning.

7.4.3 Guarantees for Path Translation

It may be possible to more formally define the translation between models with differential constraints by using Lie algebra as described in [55]. We discussed the topology of the state space in Chapter 4.5, as defined for our planning models, we can also discuss the set of all velocity vectors belonging to a vector space [23] as curves on manifolds. This can be referred to as the tangent space. By discussing the tangent space we can discuss differentiable manifolds and describe how the tangent space can also be described as a manifold. Therefore, we can imagine the motions the robot takes as actions between states as a differentiable mapping. This mapping is better described as a Lie group that contains a collection of vector fields. Lie groups are models for configuration spaces of mechanical systems where each point of the space allows continuous motions as defined by a differentiable manifold [92]. A Lie group forms a differentiable manifold. Any Lie group gives rise to a Lie algebra (`https://en.wikipedia.org/wiki/Lie_algebra`).

There could also be guarantees for translation between model fidelity by characterizing Lie algebra as Lie brackets for a specific model. Formalizing the definition of state reachability, in higher models when tracking lower fidelity reference paths, would give guarantees on the ability to translate between particular models. One possibility is to define reachability for configuration states in higher models as controllability. Controllability describes the ability to move a system in its entire configuration space as defined in `https://en.wikipedia.org/wiki/Controllability`. The Lie brackets of vector fields describe controllability. This describes if a robot can reach any given point in the configuration space. It is possible to determine the Lie bracket of the Lie algebra, and this is used to determine if a system is integrable. From Lavalle's book [55], the Lie brackets can characterize all possible directions for each configuration state. Therefore, it could be useful in more formally characterizing movement in translation and propagation of lower model paths in higher model configuration space. For formal guarantees, it may be possible to discuss if translation is even possible between particular configuration spaces by proving if a particular higher model's configuration space is even reachable.

7.5 Model Selection Extension: Informed Search

While a model's order in the hierarchy is often a good indication of its performance (both in terms of planning time and success), it does not necessarily give any indication of how well it will do in a particular environment. While we would expect higher models to perform closer to how the real robot interacts with the world, it is unclear if what that model captures is useful for a particular environment.

This section discusses ideas for directly selecting the next applicable model in the hierarchy rather than using a Breadth First Search. Breadth First Search is flawed in not distinguishing between the choice of models when they have comparable fidelity. We define comparable fidelity as those models that are present along the same level of the graph hierarchy. These are models that are not translatable between and therefore are not comparable in a BFS selection strategy. The ordering of the models is arbitrary and therefore the model selection may not be as good for a particular environment when it is uninformed. Direct selection requires more information to be stored with each model based on past data of model usage in similar environments. We would like to be able to have a more informed model selection which skips levels of the hierarchy to directly choose the most appropriate model.

7.5.1 Model Selection for Base Models Using Informed Search

The lower model path is checked in the highest possible model. If the check fails, such that an impediment is found, a model must be selected to re-plan in. The model search starts with the next highest model of the current lower model path. If multiple higher models of the lower fidelity model are found to fail, and are along the same level of the tree hierarchy, a choice must be made between models. The choice in the original algorithm was through Breadth First Search (BFS). A more informed choice utilizes a utility function to decide between models by choosing the model with lower expected cost.

For estimating a utility function for model selection:

1. Organize the initial static planning models in a hierarchy.
2. Obtain a Probability of Success $P(s)$ within the current environment $P(s \mid m, e)$.
3. Obtain a path generation cost C within the current environment $C(m, e)$.

The $P(s \mid m, e)$ can be calculated from multiple simulations over a large variety of environments in order to generalize this value for new environments. The $C(m, e)$ would also need to be estimated in a way that allows it to be obtained even in unseen environments, perhaps there is a way to estimate this value by recording path lengths over varied environments. This path length is represented as an interval. It may also be possible to extrapolate the probability of execution success and potential path length of this model on future environments with a learning approach, as suggested in Section 7.5.3. Thoughts on how to compute these values are given in subsequent sections, but for now we assume they can be determined.

The informed utility function created for deciding which model to switch to is:

$$\min P(s \mid m, e) * C(m, e) + (1 - P(s \mid m, e)) * R \tag{7.1}$$

Where R is a constant recovery cost equivalent for all models.

In the following subsections, we describe considerations attributed to more informed selection, and how risk affects model choices.

7.5.2 Model Selection for Uncertainty Models Using Informed Search

While the general hypothesis is that adding a model with increased uncertainty should be cheaper than a higher order model with more explicit detailed information and constraints, it is not clear if plan generation actually will be. Plan generation can be more expensive with a model of higher uncertainty when the padding amount is large enough to make path finding difficult. This can also occur if it's difficult to find a path within narrow corridors or a tight turn where otherwise the padding amount is sufficiently small.

Since it is not clear that choosing to switch to a model with uncertainty is actually more beneficial and under what context of the environment it is truly mode useful, this makes a case for choosing models based on utility rather than a search order. Search order forces an ordering to what is believed is higher fidelity than other models which we have demonstrated can shift depending on the environment. The issue is that we may try to be too conservative by choosing a model with less padding first because of the search order (meaning we are checking more models than necessary). For example, if we choose a model to re-plan in that subsequently fails and then find a more useful which also addresses the failure, then both models contribute useful information, but the initial model chosen for re-planning did not cover all of the information in that failure. Or alternatively, by choosing to re-plan in a particular model we then move to higher models that we have labelled as 'higher' for this model, but this labelling for this particular environment is incorrect. In which case, we are effectively skipping lower models that could have been helpful for addressing the failure and settling on a model higher than would have been necessary if we had chosen using a utility.

Estimating Success of Uncertainty Model

The probability of success $P(s)$ of the uncertainty model for a given environment could be estimated using the lower model path, the probability of success of the lower base model, and a resulting Voronoi diagram which all contribute to a success curve. A curve is determined where minimal padding provides zero success and max padding is closer to 100% successful (minus some ϵ value to assume a path can be found). This estimates $P(s \mid m, e)$.

Uncertainty always increases the padding amount of the base model. Therefore, the success probability with additional uncertainty is hypothesized to be greater than that of the base model.

The max padding amount at the top of the probability curve is determined using the lower model path within the voronoi diagram. Every voronoi diagram cell that intersects with the lower model path is determined. Since voronoi edges are equidistant between obstacles they provide a value for a max possible padding amount between obstacles. The max amount necessary to guarantee success will be this max amount minus ϵ, and used as the top of value of the success curve. This curve for probability is fit between the minimum padding amount used in the base model and the max padding amount.

Estimating Cost of Uncertainty Model

This cost is an equivalent estimate for what the model's path length would be. To estimate this value, we need to determine how much longer the path would be if an uncertainty model is used. The value increases if the new path creates a bifurcation causing it to switch to a different homotopy class. For example, if the model with less uncertainty is able to find a particular path; the question is whether the model with more padding can possibly find the same path. If it is, the costs are comparable if the current path has enough space to fit the model with the larger padding and, if not, we try to quickly approximate the length of an alternate path that would be feasible.

The cost for the dynamic uncertainty model for a given environment is estimated using a distribution of lower model paths and the resulting voronoi diagram. Every voronoi diagram cell that intersects the lower model paths is determined. The minimum amount of padding that was added during the construction of the dynamic model is then compared to each edge for every intersecting cell. If the minimum padding fits within this path distribution the deviation is small and only a small path increase from the base model is determined. If the minimum padding requires a large deviation outside the path distribution then the deviation is large and requires a larger cost value.

When calculating the path length, calculations start from the point perpendicular to the start waypoint then follow along edges until the point perpendicular to the end point. This is because the end edge may be much longer than where the final end point actually is, and we want to keep the cost closer to the voronoi path length for similar homotopy classes.

The model choice now includes models with added uncertainty. Now switching must decide to go to a higher fidelity static model, or to go to a more conservative model which has increased uncertainty. Again the choice is to minimize the expected cost where the probability of success and costs of the uncertainty models are also included.

7.5.3 Selecting the Model Based on Environment Features

One possibility for obtaining $P(s \mid m, e)$ is through a classification approach. It might be possible to find this value based on experience with running models over a large set of training in environments, and then evaluating the algorithm on a separate environment test set. One caveat is that it could take a very long time to simulate enough examples with every model to obtain a good coverage that generalizes over many environments. The goal would be to learn an estimation of the probability given the environment features. The main question is what environment features would be best for predicting success.

It is important to allow the classification to be updated online because unanticipated failures could happen that were not captured during initial training. The robot can start with an informed prior based on simulation trials. Then the classification for success probabilities would change as the robot encounters more failures during execution. We imagine that as the robot travels to a part of the environment for which it does not currently have a good estimation of success, it could start by using the initial uninformed model selection algorithm and then update the correlation between model and environment feature set as it encounters more examples. As an example, a robot is going through the world and gets stuck (unable to find a plan in 1 second) then it goes offline or takes some time to optimize the correct parameters to be unstuck for that particular

scenario instance. Next time the robot gets stuck in the same area it uses nearest neighbor to try and match parameters similar to the current instance, if it gets stuck again it again tries to optimize for that particular instance and adds it to its database.

We would like to be able to answer the question: When given an arbitrary model, How do I know how well it will help in the given environment? (without knowing the environment) One way is by attempting to leverage environment features that indicate models that generate successfully executable paths for that part of the environment. Another could be to use the behavior of previous models in the model graph to give some indication of how a future model would perform based on its position or relationship to these previous models. This allows leverage of the hierarchical model graph to focus on a particular group of related models which perform better for a particular area of the environment. For example, an environment with different coefficient of friction on the terrain would do better with a model that took into account the robot's wheel contacts. This would also depend on how much the environment varied for differences in terrain interaction. If this value was constant this model would be less necessary. If this value varied greatly (maybe determined with an entropy parameter) then this variable could be more important. Another example is an environment with a time-dependent obstacle. If a model has configuration variables that relate to the robot's time- dependent trajectory than these would be important variables that could improve the model's success in environments where time is more constrained.

7.5.4 Switching with Risk

Incorporating risk attitudes captures the trade-off between variance and mean in the distribution of reward outcomes. For instance, risk-seeking policies tend to have a lower mean and larger variance (more upside, but also more downside) than risk-neutral policies. Utility theory provides structure for making decisions under varying risk attitudes [81]. A utility function maps an agent's value to a plan of wealth that represents the agent's rational choice. Linear utility functions maximize expected cumulative reward and represent risk-neutral decision makers, while exponential functions model risk-sensitive decisions. Concave utility functions reflect risk-averse decisions, and convex utility functions characterize risk-seeking decisions. The convexity of these functions changes for different risk factors.

The notion of risk can be considered during model selection. At any point during the task a robot can decide which model balances a high probability of success with its cost of use. Example costs associated with a model include recovery path length, recovery time, amount of uncertainty, and computation time. For example of a robot choosing between a model with an increased uncertainty footprint versus no uncertainty. This tradeoff between choosing a model that does or does not have uncertainty is a function of how much risk the robot should take. If the robot chooses to not consider uncertainty, it assumes a higher risk of failure, but with a possible reward payoff of a much shorter, faster path. The variance of the robot's utility is very large. It will either create a low cost path that accomplishes its task or it will fail trying and need to recover. If the robot does consider uncertainty, it acts much more risk-averse. Overall, it will tend to fail less often, but its expected path cost is lower than in the risk-seeking minimal uncertainty case. Adding uncertainty also brings in a different kind of risk associated with the ability to generate a plan. Even though plans will tend to be more successful when considering

uncertainty than without, there is a higher risk of not being able to find a plan at all. If the robot is able to find a plan, it has a higher chance of success without a lot of variance in the path cost.

Risk also plays a part in choosing which model to use in order to satisfy an overall task constraint. Since the choice of model is tied to a probability of success, the cost of failure may be adjusted to generate more or less risky decisions. This allows the robot to have a more risk-averse or risk-seeking strategy for task completion. Example task constraints include: time to complete the task, computation time, or throughput. The robot would choose which model at any given point in which to re-plan in to fulfill a constraint. For example, suppose the robot has a constraint on task throughput which is determined by the number of objects it can transport in a certain time limit. Let's say the robot is reaching the end of its task and it has only moved 3 of the 10 boxes. For the robot to try to maximize its throughput (move all 10 boxes) it may decide to risk carrying all the remaining boxes at a higher speed. The risk is if any boxes are dropped it may take more time for the robot to re-plan to pick them up then had it just moved two boxes at a time. The closer the robot gets to reaching the task's end the more the planner would choose models that increase throughput but have a higher probability of failure (much more risky). If the task was well within the throughput limits, the planner could be more conservative in choosing models that increase time costs while safely moving boxes. The architect could set limits on how much risk they are willing to allow the robot to take when trying to meet the constraint.

A constraint over the entire task ties choices made at each time in the planning process. As shown above, constraining the throughput affects which models are chosen to either meet or stay within the throughput limit. One approach is to formulate this as a problem that is solved through Dynamic Programming techniques. We can break up the choice of model recovery into maximizing the reward of successive subproblems. The current payoff $R(s, a)$ of choosing model m over a feature region f between states s_t to s_{t+1} transitions to the next state $T(s, m)$ based on a probability of success. The value of the cumulative discounted reward is: $V(s_0) = max \sum_{t=0}^{\infty} \beta^t R(s_t, m_t)$ subject to some constraint, c. This is the robot's utility function.

This then follows as an optimization of the Bellman Equation subject to some constraint, where β is a discount factor.

$$V(s) = max\{R(s, m) + \beta V(T(s, m))\} \tag{7.2}$$

Pervious work, described in [46], varies a risk parameter that generates different 'risky strategies' by transforming transition probabilities. These strategies are precomputed policies that choose more or less risky actions based on a risk parameter subject. An agent then switches between these policies at run time based on a constraint (specifically a reward threshold tied to task completion time). Risk seeking agents assume a higher variance for the chance of maximizing a threshold. We can extend this approach to allow generating more conservative policies. It would also be possible to augment the utility function used for informed model selection directly by incorporating a risk attitude Risk sensitive planning is further described in [51], [60].

7.6 Thoughts on Expansion to an Execution-Based Framework

For extending our approach to using the model selector at execution time, we would also want to consider how to detect the failure without actually colliding. This would require anticipating the failure by having the robot reason about failure symptoms before a failure occurs. It would also be useful to determine an initial recovery routine so that subsequent re-plans do not cause an immediate failure. A current (naive) approach is to have the robot move a small distance from the direction the failure occurs to give it enough space to generate a new plan. Movement of a robot to this initial safer state is what we mean by initial recovery.

Model selection occurs after a collision is detected and the robot has initially recovered to a safe state. A possible initial recovery could be to plan a recovery action in a higher fidelity model. A poor initial recovery strategy could actually cause the robot to be unrecoverable. For example, a robot could become wedged in a tight corner, become high-centered, or even flip over, which would be an unrecoverable state. Another example is traveling through a muddy area. Depending on the wheel direction and speed the robot could become more stuck.

The initial recovery strategy also should consider footprint padding when choosing a recovery location. For example, when uncertainty adds padding to the robot footprint it is larger than the robot's actual size. If the robot initially recovers to a space very close to a wall, this additional padding could cause the planner to think that all possible waypoint to re-plan from are invalid. A possible initial recovery, when reasoning about uncertainty, might be to try footprints that are less than this current padding amount and have the padding amount be reduced towards the actual robot's size to increase the chances of finding a successful plan. Or, one could plan the recovery action using the model with uncertainty, which would ensure that the robot's position would be valid for that model.

7.7 Thoughts on Dynamic Obstacles

In this section we discuss thoughts on extending our approach to handle dynamic obstacles not known at plan-time. Our current hierarchy only considers known obstacles. For dynamic obstacles discovered during execution-time, the feasibility detector could constantly re-check the plan against new information. If the plan fails feasibility checking, we could stop the robot and execute our normal model selection and re-planning process. The model hierarchy would have to be augmented with new models that could consider and model observed dynamic obstacles to varying fidelity. For environments without any dynamic obstacles, there should be little overhead as these models could be skipped at plan-time.

Example models might exclude dynamic obstacles altogether, model them as static at their last known position, or linearly extrapolate their observed motion. Even more expensive models could be constructed leveraging known semantics about how obstacles might move through a particular environment, such as a hospital.

In another case, we assume that we do not have a model of the dynamic obstacle and must learn to avoid it by exploring and re-planning in the environment. This is not a well formed

idea, but it is inspired by related work with real-time motion planning and re-planning for unpredictable obstacles as described in [71]. In this work, unpredictable obstacles are avoided by re-wiring the search path using RRT subtrees. We believe that it might be possible to do something similar with our multi-model approach. It is unclear which model would be selected for re-planning around the discovered dynamic obstacle, but one possibility is to create a uniform padding area that conservatively avoids the obstacle entirely and have the padding shrink or expand depending on the exploration of the space.

7.8 Notes on Explainability

The idea of multiple models with various amounts of information is something that might be beneficial for explaining to a user. This could be done at both plan-time and execution time. The model variables in our framework are known. The robot could explain the path generated at plan-time based on the models used to generate the initial execution path. During execution, the robot would have a notion of state based on the current model it plans with, and models it has tried that were not beneficial. The robot could provide information on what models it is currently using or even give updates. This includes tracking when the number of model switches seems higher than expectation or decisions are approaching the use of the highest fidelity model.

We believe combining learning with hand-tuned models is best, but there is more difficulty for explaining what is happening in a learned model. Hand-tuned models can start as the prior for which learning continues, and has the added benefit of using a model that is more explainable if things go awry. This includes a better understanding of how a path is generated.

Our model selection process provides knowledge about what models have been attempted at a particular point in the task. The hierarchical model graph is leveraged for deciding what model to use next. If the robot has a set order of recoveries to try that end when it reaches the highest model, then the robot knows at what point its possible model choices are about to be exhausted. If the robot is about to try re-planning in the highest model, it knows at this point that if a solution is not found for the highest model it may not be able to recovery. The robot will at this point either run out of options or could fail if it proceeds. At this point, the robot could alert the user knowing its next option is to wait and ask for help.

7.9 Future Work Summary

In summary, we believe there are many extensions and directions to our current work that would lead to the ultimate goal of increasing reliable robot autonomy. In this chapter we touch on some of the major ideas for being smarter about model selection and closing the loop of the system by allowing multi-model re-planning during execution time. We begin with extensions that would improve switching success rates and planning times. This includes thinking more about what model is used for feasibility checking the path Section 7.1, and reasoning more about re-using search during the re-planning process Section 7.2. We go on to address extensions for uncertainty representations, Section 7.3, as well as ideas for providing more formal guarantees in our approach, Section 7.4. We also believe that a more informed model selection process

would decrease planning times as we discuss in Section 7.5. Informed model selection would be even more impactful when moving towards an execution time approach because there could be situations where execution time operation highly depends on a fast recovery solution achieved by going straight to the applicable model rather than checking through multiple models in the hierarchy. Lastly, informed model selection can consider a risk parameter to change the utility for model selection to favor different types of recovery strategies. Therefore, informed selection could select models outside of expectation to reach other task objects rather than just selecting the next model which has the best expected value for both being robust and plan efficient. During execution, we could also consider dynamic obstacles, Section 7.7, and we finish with some thought on the benefits of using hand-tuned models for explainability, Section 7.8.

Chapter 8

Conclusions

Robots often operate in non-uniform environments, and over time will encounter unexpected environment interactions. These unstructured environments may include large open areas appropriate for simple models as well as cluttered, constrained areas that require modeling more detailed environment interactions. Planning in these environments benefits from a multi-model approach.

In this thesis we presented a probabilistically complete plan-time model switching approach for path generation over a hierarchy of multiple models in continuous space. The framework determines *when* to switch between models and *what* model to switch to for producing a single mixed-model plan. A model includes the robot's configuration space, environment workspace, and motions that connect configurations. In continuous space, motion equations describe actions that include differential and dynamic constraints. Our model hierarchy includes multi-fidelity models and models with variable uncertainty in the form of additional footprint padding.

The planning framework switches models to minimize planning resources while maintaining robust execution success over varied environments. Our results show comparisons for single model performance versus switching among models with varying fidelity including those with variable uncertainty padding.

When switching between just the lowest and highest model, we see equivalent execution success with statistically significantly faster planning times on average. In the multi-model switching experiments, while we see comparable execution success, the average normalized plan time is worse than that of the highest model albeit with a very large variance. Therefore, while we make an effort to minimize planning times and maintain execution success with our switching approach, we saw that depending on the environment and implementation this is not always achievable.

We also found that higher fidelity models are generally better than lower fidelity models for success rates as well as planning times, but not always. We believe this is due to a combination of implementation (how different models sample and propagate their robot state through the space) and the environment (areas that are particularly more difficult for a model to plan through). We touch on specific cases of this in our results Chapter 6.

8.1 Execution Success

A higher fidelity model does not always have higher success rates, depending on the environment. Initially we were expecting execution success rates to monotonically grow with fidelity. Therefore, while we organize our models hierarchically by fidelity, the notion of higher fidelity does not always equate to a higher success rate. As shown in our single model performance results in Chapter 6, the implementation of the model may be directly influenced by the type of environment. Although, in most of our environments higher fidelity models were the most beneficial. For example, a robot traveling through very constrained areas, was most successful with paths generated from models that include the trailer. A robot traveling through an autonomous swinging door benefited from models that included velocity. For the few environment cases where this did not occur, it further argues for the ability to match environment features to models that perform well in those areas. We touch on this extension in future work, Chapter 7.

8.2 Plan-Time Efficiency

Average planning times across environments while switching models are not always better than planning from start to goal using just the highest model. Again we found that this is affected by a combination of the environment and how well a model's implementation performs in that particular environment.

In the worst case, switching between multiple models which eventually require the highest model have higher planning times than planning in just the highest model. This is because the number of re-plans required to plan from the start to the goal in the highest model would have to be at least one. Therefore the initial planning time for generating the lower model reference path plus re-planning in the highest model from start to goal will be greater than planning in the highest model alone. The plan savings for model switching comes from using lower fidelity models in areas of environments where higher fidelity models are a waste of resources. These are situations where comparable execution success rates are achieved with either the use of multiple lower models or a combination of lower and higher models.

Our approach is geared towards the average case. There will always be situations where just planning in the highest model is better, but on average, using the model hierarchy is preferred.

In general, we conclude that planning times for switching are greatly affected by the following things:

1. Discrepancies between the feasibility checker and how the robot executes leads to false positives which require more switching than necessary.
2. The environment could include tight corridors where partial path repairs stall due to RRTs' tendency to increase node expansion in constrained spaces.
3. Forcing higher models to generate plans guided along lower model reference paths may cause higher models to be re-planned in areas which are difficult for the RRT to find plans. Possible improvements are described in future work.

The last two items are examples where having a more informed way of knowing which models are more applicable to particular environment features could better guide model selection.

This would prevent the selection of re-plan models which could take longer or have difficulty generating paths in these particular parts of the environment.

Lastly, we also believe our longer planning times are largely attributed to making overly conservative design decisions. We preference robust paths (higher execution success and the reduction of false negatives) over efficient planning times. Due to this we always choose to be more conservative when our approach allows. We've included examples of this throughout the document. This includes the design of how we collision check against our automatic swinging door environment, the addition of padding to our feasibility checker, and how we choose to re-plan first in a lower model with uncertainty before trying a model with higher fidelity.

8.3 Contributions

Overall our approach describes when to switch models by feasibility checking lower reference paths in the highest model. We have discussed implementation of this as well as the performance described by false positive and false negative rates.

Our framework also explicitly reasons about what model to switch to and re-plan in. We include analysis for describing how we switch among models of varying fidelity and how we decide to add models with variable footprint padding. Once a model is selected, we also discuss details of path repair and the advantages of re-weighting different re-planning trees that expand towards a set of intermediate goals.

To re-iterate our contributions more explicitly:

1. We provide a framework for multi-fidelity plan-time model switching.
2. We incorporate uncertainty by automatically determining variable padding models in the model graph hierarchy.
3. We developed a model selection strategy for models with higher fidelity and variable padding.
4. We demonstrate advantages of the multi-fidelity model switching framework.

Overall, the results show that, in general, the use of a hierarchy of models produces comparable success rates but with improved planning times to that of the highest model. We also demonstrate examples where switching with the full model hierarchy is more beneficial than just planning between the lowest and highest model, but there are still issues that need to be better understood in order to increase the situations for which this approach is applicable.

Chapter 9

Appendix

9.1 Larger Demo Environment

Here we show screen captures of a full hospital environment with swinging doors, a nurse's station, and gurneys in the hallway. We show this environment as a demo and therefore only discuss what is occurring for one switching experiment to demonstrate the approach. The upper picture shows the plan-time path as it is generated, Figure 9.1, and the bottom picture shows the simulated robot in Gazebo. In this experiment, there are two places the approach chooses to switch to a higher model, Figure 9.1 and Figure 9.3. Figure 9.2 shows where the path is corrected to plan in $XY\theta V$ for going through the swinging doors, and Figure 9.4 shows where the path chooses to switch to $XY\theta\theta_1$ for going around the nurse's station and gurneys. The robot executes this path after it is determined feasible and in Figures 9.5 it goes through the doors, avoids the nurse station, and then goes through the gurneys.

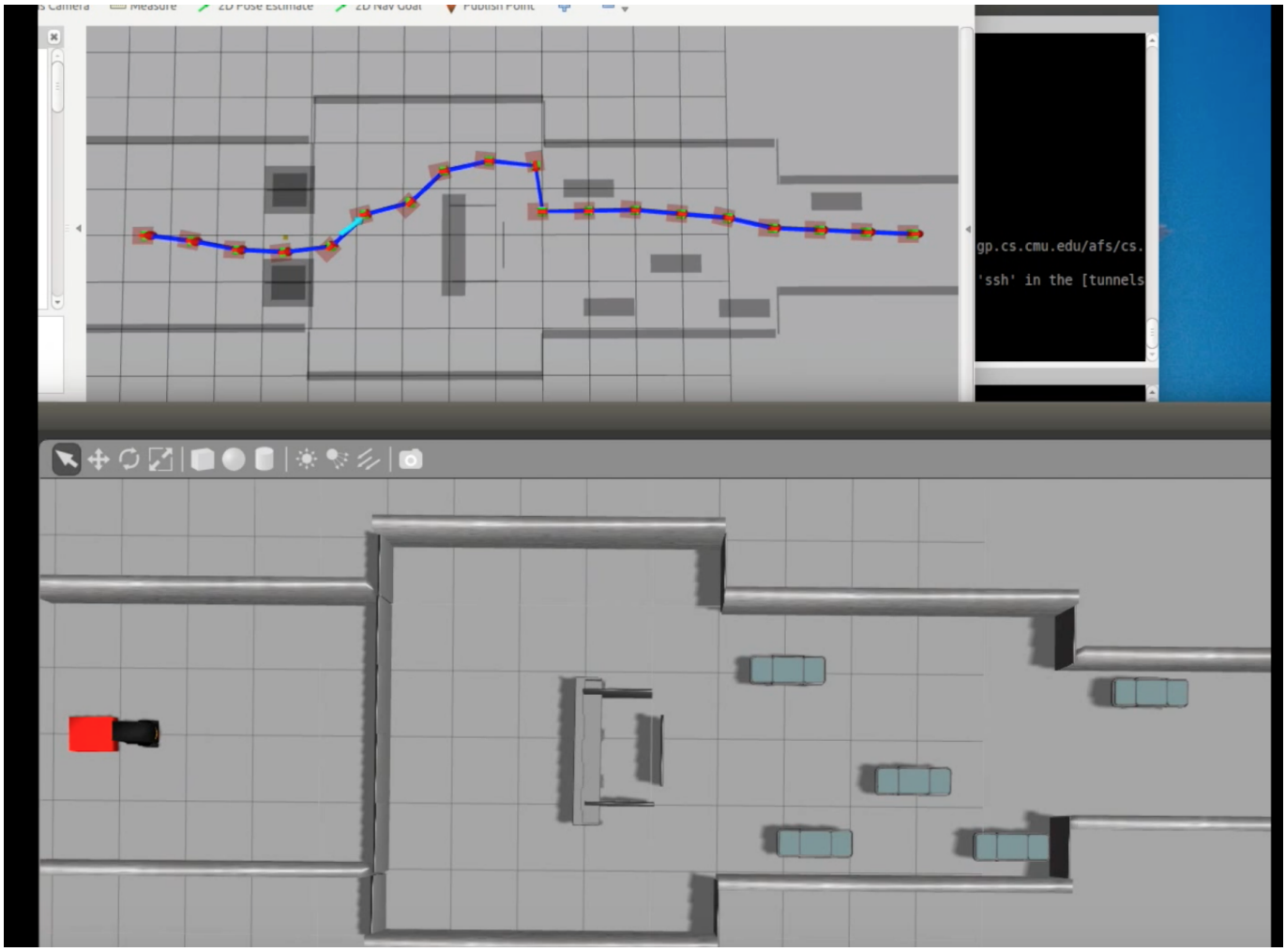


Figure 9.1: A collision is detected at plan-time near the swinging doors.

Figure 9.2: A model is selected and the path near the doors is repaired with model $XY\theta V$.

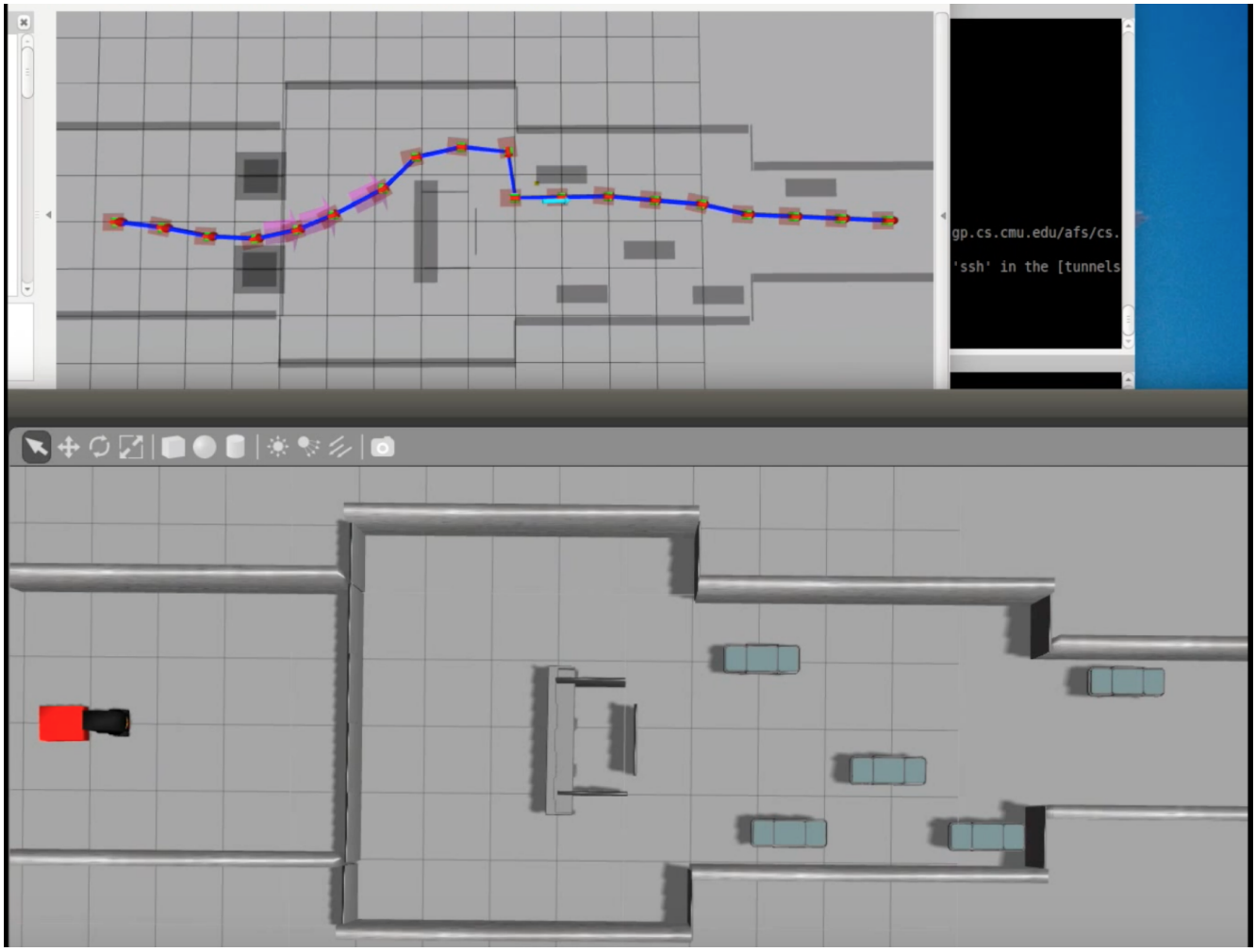


Figure 9.3: The feasibility detector detects a second collision near the a gurney.

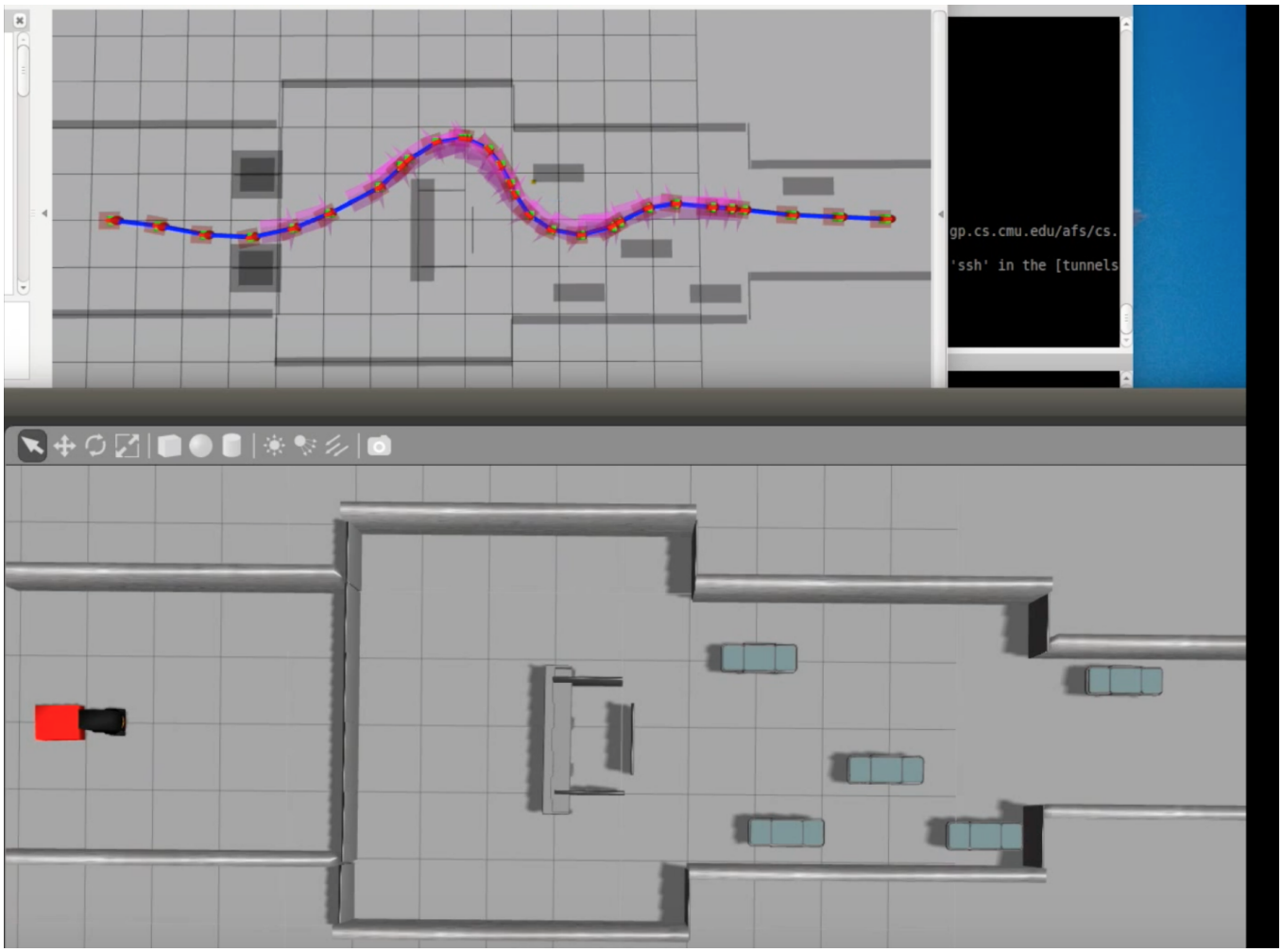
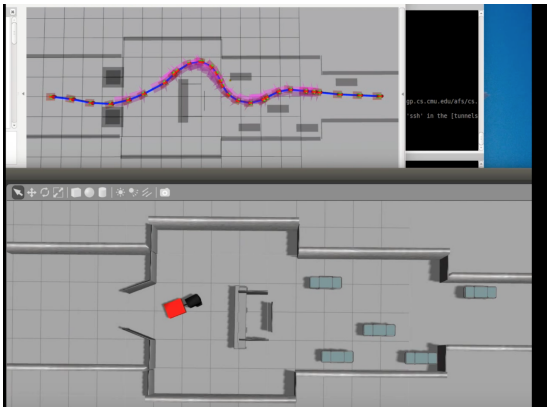


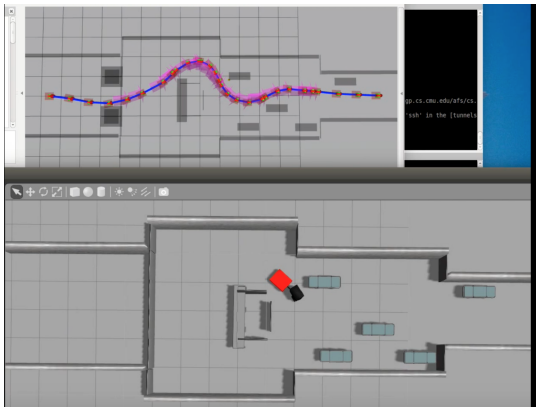
Figure 9.4: A model is selected and the path near the gurney is repaired with model $XY\theta\theta_1$.

(a) Robot goes through the automatic sliding doors.

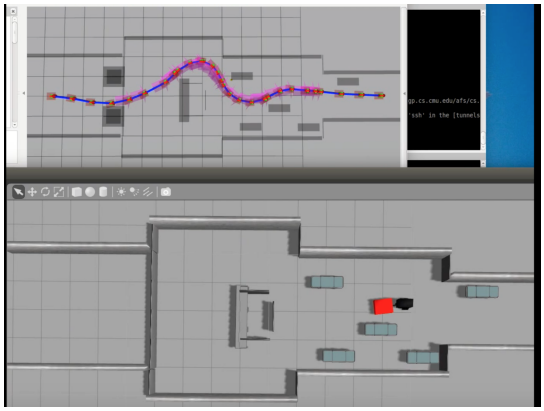


(b) Robot continues through the automatic doors.

(c) Robot drives around the nurse's station.



(d) Robot continues around the nurse's station and a gurney.



(e) Robot continues through the gurneys to the hallway.

Figure 9.5: If the path is determined feasible at plan-time it is then sent to the robot for execution in the real (simulated) world, with full physics.

Bibliography

[1] Alfred V. Aho, Michael R Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972. 3.2

[2] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-heuristic a. *The International Journal of Robotics Research*, 35(1-3):224–243, 2016. 2.3.1

[3] Brian Axelrod, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Provably safe robot navigation with obstacle uncertainty. *arXiv preprint arXiv:1705.10907*, 2017. 2.2.2

[4] Christopher Baker, Aaron Morris, David Ferguson, Scott Thayer, Chuck Whittaker, Zachary Omohundro, Carlos Reverte, William Whittaker, D Hahnel, and Sebastian Thrun. A campaign in autonomous mine mapping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 2. IEEE, 2004. 1

[5] Bonny Banerjee and B Chandrasekaran. A framework for planning multiple paths in free space. In *Proceedings of 25th Army Science Conference, Orlando, FL*, 2006. 5.6

[6] Michael Barbehenn and Seth Hutchinson. Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest paths trees. *Robotics and Automation, IEEE Transactions on*, 11(2):198–214, 1995. 2.3.1

[7] Jennifer Barry, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. A hierarchical approach to manipulation with diverse actions. In *IEEE Conference on Robotics and Automation (ICRA)*, 2013. 2.1

[8] Sven Behnke. Local multiresolution path planning. In *Robocup 2003: Robot Soccer World Cup VII*, pages 332–343. Springer, 2004. 2.1, 2.3.1

[9] Richard Bloss. Mobile hospital robots cure numerous logistic needs. *Industrial Robot: An International Journal*, 38(6):567–571, 2011. 1

[10] Jim Blythe. An overview of planning under uncertainty. In *Artificial intelligence today*, pages 85–110. Springer, 1999. 2.2.1

[11] Abdelbaki Bouguerra. Robust execution of robot task-plans: A knowledge-based approach. 2008. 2.2.1

[12] Alejandro Bravo-Doddoli and Luis C García-Naranjo. The dynamics of an articulated n-trailer vehicle. *Regular and Chaotic Dynamics*, 20(5):497–517, 2015. 7.1

[13] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*,

volume 3, pages 2383–2388. IEEE, 2002. 2.2.1, 4.1.4

[14] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 723–730. IEEE, 2011. 2.2.2

[15] Shih-Yi Chien, Michael Lewis, Siddharth Mehrotra, Nathan Brooks, and Katia Sycara. Scheduling operator attention for multi-robot control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 473–479. IEEE, 2012. 1

[16] Wonyun Choi, David Zhu, and J-C Latombe. Contingency-tolerant robot motion planning and control. In *Intelligent Robots and Systems' 89. The Autonomous Mobile Robots and Its Applications. IROS'89. Proceedings., IEEE/RSJ International Workshop on*, pages 78–86. IEEE, 1989. 2.3.1

[17] Shushman Choudhury, Oren Salzman, Sanjiban Choudhury, and Siddhartha S Srinivasa. Densification strategies for anytime motion planning over large dense roadmaps. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3770–3777. IEEE, 2017. 2.3.1

[18] R Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992. 5.2, 5.2.1

[19] Jacob W Crandall, Mary L Cummings, Mauro Della Penna, and Paul MA de Jong. Computing the effects of operator attention allocation in human control of multiple robots. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 41 (3):385–397, 2011. 1

[20] Richard Dearden and Chris Burbridge. An approach for efficient planning of robotic manipulation tasks. In *Twenty-Third International Conference on Automated Planning and Scheduling*, 2013. 2.1

[21] Mehmet Dogar, Ross A Knepper, Andrew Spielberg, Changhyun Choi, Henrik I Christensen, and Daniela Rus. Towards coordinated precision assembly with robot teams. In *Proceedings of the 2014 International Symposium on Experimental Robotics, page TBD*, 2014. 2.1

[22] Mehmet R Dogar, Kaijen Hsiao, Matei Ciocarlie, and Siddhartha S Srinivasa. Physics-based grasp planning through clutter. In *In RSS*. Citeseer, 2012. 2.1

[23] Silvia Ebetiuc and S Harald. Applying differential geometry to kinematic modeling in mobile robotics. In *WSEAS international conference on dynamical systems and control, Venice, Italy*, pages 2–4, 2005. 4.5, 7.4.3

[24] Tom Erez and William D Smart. A scalable method for solving high-dimensional continuous pomdps using local approximation. *arXiv preprint arXiv:1203.3477*, 2012. 3.1.1, 7.3

[25] Dave Ferguson, Nidhi Kalra, and Anthony Stentz. Replanning with rrts. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1243–1248. IEEE, 2006. 2.2.1

[26] J-A Fernández-Madrigal and Javier González. Multihierarchical graph search. *Pattern*

Analysis and Machine Intelligence, IEEE Transactions on, 24(1):103–113, 2002. 2.1, 2.3.1

[27] Richard E Fikes, Peter E Hart, and Nils J Nilsson. Learning and executing generalized robot plans. *Artificial intelligence*, 3:251–288, 1972. 2.2.1

[28] Paul A Fishwick and John A Miller. Ontologies for modeling and simulation: issues and approaches. In *Simulation Conference, 2004. Proceedings of the 2004 Winter*, volume 1. IEEE, 2004. 2.1

[29] Moritz Göbelbecker, Charles Gretton, and Richard Dearden. A switching planner for combined task and observation planning. In *AAAI*, 2011. 2.2.1, 2.3.1

[30] Kalin Gochev, Benjamin Cohen, Jonathan Butzke, Alla Safonova, and Maxim Likhachev. Path planning with adaptive dimensionality. In *Fourth Annual Symposium on Combinatorial Search*, 2011. 2.3.1

[31] Kalin Gochev, Alla Safonova, and Maxim Likhachev. Planning with adaptive dimensionality for mobile manipulation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2944–2951. IEEE, 2012. 2.3.1

[32] Kalin Gochev, Alla Safonova, and Maxim Likhachev. Incremental planning with adaptive dimensionality. In *Twenty-Third International Conference on Automated Planning and Scheduling*, 2013. 1, 2.3.1

[33] RONALD Goldman. Graphics gems. *Graphics gems*, page 304, 1990. 5.2.1

[34] Birgit Graf, Matthias Hans, and Rolf D Schraft. Care-o-bot iidevelopment of a next generation robotic home assistant. *Autonomous robots*, 16(2):193–205, 2004. 1

[35] Kris Hauser and Jean-Claude Latombe. Multi-modal motion planning in non-expansive spaces. *The International Journal of Robotics Research*, 2009. 2.1, 2.3.1

[36] Kris Hauser, Victor Ng-Thow-Hing, and Hector Gonzalez-Baños. Multi-modal motion planning for a humanoid robot manipulation task. In *Robotics Research*, pages 307–317. Springer, 2011. 2.3.1

[37] Frederik W Heger and Sanjiv Singh. Robust robotic assembly through contingencies, plan repair and re-planning. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3825–3830. IEEE, 2010. 1, 2.2

[38] Jörg Hoffmann. Ff: The fast-forward planning system. *AI magazine*, 22(3):57, 2001. 2.2.1

[39] Thomas M Howard et al. Adaptive model-predictive motion planning for navigation in complex environments. 2009. 2.3.1

[40] Adele E Howe. Improving the reliability of artificial intelligence planning systems by analyzing their failure recovery. *Knowledge and Data Engineering, IEEE Transactions on*, 7(1):14–25, 1995. 2.2

[41] Ajinkya Jain and Scott Niekmun. Efficient hierarchical robot motion planning under uncertainty and hybrid dynamics. *arXiv preprint arXiv:1802.04205*, 2018. 2.2.2

[42] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*,

pages 1470–1477. IEEE, 2011. 2.1, 2.2.1, 2.3.1

[43] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4569–4574. IEEE, 2011. 2.2.1

[44] Nidhi Kalra, Dave Ferguson, and Anthony Stentz. Incremental reconstruction of generalized voronoi diagrams on grids. *Robotics and Autonomous Systems*, 57(2):123–128, 2009. 5.6

[45] Subbarao Kambhampati and Larry Davis. Multiresolution path planning for mobile robots. *Robotics and Automation, IEEE Journal of*, 2(3):135–145, 1986. 2.1, 2.3.1

[46] Breelyn Melissa Kane and Reid Simmons. Risk-variant policy switching to exceed reward thresholds. In *ICAPS*, 2012. 7.5.4

[47] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1478–1483. IEEE, 2011. 7.4.1

[48] Ross A Knepper and Matthew T Mason. Improved hierarchical planner performance using local path equivalence. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3856–3861. IEEE, 2011. 1, 2.3.1

[49] Ross A Knepper, Siddhartha S Srinivasa, and Matthew T Mason. An equivalence relation for local path sets. In *Algorithmic Foundations of Robotics IX*, pages 19–35. Springer, 2010. 5.6

[50] Sven Koenig and Maxim Likhachev. D* lite. In *AAAI/IAAI*, pages 476–483, 2002. 2.2.1

[51] Sven Koenig and Reid G Simmons. Risk-sensitive planning with probabilistic decision graphs. In *Proceedings of the 4th international conference on principles of knowledge representation and reasoning*, page 363, 1994. 7.5.4

[52] Benjamin Kuipers. A hierarchy of qualitative representations for space. In *Spatial Cognition*, pages 337–350. Springer, 1998. 2.1

[53] Tobias Kunz and Mike Stilman. Kinodynamic rrts with fixed time step and best-input extension are not probabilistically complete. In *Algorithmic foundations of robotics XI*, pages 233–244. Springer, 2015. 4.5.1

[54] Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001. 2.2.1, 4.1.1, 4.5.1

[55] Steven Michael LaValle. *Planning algorithms*. Cambridge university press, 2006. 1.2, 3.1, 4.5, 7.4.3

[56] Duong Le and Erion Plaku. Cooperative multi-robot sampling-based motion planning with dynamics. 2017. 7.2

[57] Daniel Lecking, Oliver Wulf, and Bernardo Wagner. Variable pallet pick-up for automatic guided vehicles in industrial environments. In *2006 IEEE Conference on Emerging Technologies and Factory Automation*, pages 1169–1174. IEEE, 2006. 1

[58] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime search in dynamic graphs. *Artificial Intelligence*, 172(14):1613–1643, 2008. 2.3.1

[59] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 1

[60] Yaxin Liu and Sven Koenig. Risk-sensitive planning with one-switch utility functions: Value iteration. In *AAAI*, pages 993–999, 2005. 7.5.4

[61] Sara Ljungblad, Jirina Kotrbova, Mattias Jacobsson, Henriette Cramer, and Karol Niechwiadowicz. Hospital robot at work: something alien or an intelligent colleague? In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 177–186. ACM, 2012. 1

[62] Tomas Lozano-Perez. Spatial planning: A configuration space approach. *IEEE transactions on computers*, (2):108–120, 1983. 2.3

[63] Brian MacAllister, Jonathan Butzke, Alex Kushleyev, Harsh Pandey, and Maxim Likhachev. Path planning for non-circular micro aerial vehicles in constrained environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3933–3940. IEEE, 2013. 2.2.2

[64] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 300–307. IEEE, 2010. 2.2.2

[65] Nik A Melchior and Reid Simmons. Particle rrt for path planning with uncertainty. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1617–1624. IEEE, 2007. 3.1.1, 7.3

[66] Juan Pablo Mendoza, Manuela Veloso, and Reid Simmons. Plan execution monitoring through detection of unmet expectations about action outcomes. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3247–3252. IEEE, 2015. 2.2.1

[67] Scott A Miller, Zachary A Harris, and Edwin KP Chong. Coordinated guidance of autonomous uavs via nominal belief-state optimization. In *American Control Conference, 2009. ACC'09.*, pages 2811–2818. IEEE, 2009. 3.1.1, 7.3

[68] Dennis M Moyles and Gerald L Thompson. An algorithm for finding a minimum equivalent graph of a digraph. *Journal of the ACM (JACM)*, 16(3):455–460, 1969. 3.2

[69] Illah R Nourbakhsh and Michael R Genesereth. Assumptive planning and execution: a simple, working robot architecture. *Autonomous Robots*, 3(1):49–67, 1996. 2.2

[70] Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8):1053–1068, 2010. 7.3

[71] Michael Otte and Emilio Frazzoli. {\mathrm {RRTˆ{X}}} rrt x: Real-time motion

planning/replanning for environments with unpredictable obstacles. In *Algorithmic Foundations of Robotics XI*, pages 461–478. Springer, 2015. 7.7

[72] Ali Gurcan Ozkil, Zhun Fan, Steen Dawids, Henrik Aanes, Jens Klestrup Kristensen, and Kim Hardam Christensen. Service robots for hospitals: A case study of transportation tasks in a hospital. In *2009 IEEE International Conference on Automation and Logistics*, pages 289–294. IEEE, 2009. 1

[73] Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3859–3866. IEEE, 2012. http://gamma.cs.unc.edu/FCL/. 5.3.2

[74] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987. 7.3

[75] Liam Pedersen, David Kortenkamp, David Wettergreen, and I Nourbakhsh. A survey of space robotics. In *Proceedings of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, pages 19–23, 2003. 1

[76] Mihail Pivtoraiko and Alonzo Kelly. Differentially constrained motion replanning using state lattices with graduated fidelity. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2611–2616. IEEE, 2008. 2.3.1

[77] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. Discrete search leading continuous exploration for kinodynamic motion planning. In *Robotics: Science and Systems*, pages 326–333, 2007. 2.3.1

[78] Erion Plaku, EE Kavraki, and Moshe Y Vardi. Motion planning with dynamics by a synergistic combination of layers of planning. *Robotics, IEEE Transactions on*, 26(3): 469–482, 2010. 2.3.1

[79] Robert Platt, Leslie Kaelbling, Tomas Lozano-Perez, and Russ Tedrake. Non-gaussian belief space planning: Correctness and complexity. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4711–4717. IEEE, 2012. 3.1.1, 7.3

[80] Robert Platt Jr, Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-Perez. Belief space planning assuming maximum likelihood observations. 2010. 3.1.1, 7.3

[81] J.W. Pratt. Risk aversion in the small and in the large. *Econometrica: Journal of the Econometric Society*, pages 122–136, 1964. 7.5.4

[82] Nathan Ratliff, Matthew Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009. 2.2.1

[83] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011. 2.2.1

[84] Ricardo Samaniego, Joaquin Lopez, and Fernando Vazquez. Path planning for non-circular, non-holonomic robots in highly cluttered environments. *Sensors*, 17(8):1876,

2017. 2.2.2

[85] Neal Seegmiller and Alonzo Kelly. Enhanced 3d kinematic modeling of wheeled mobile robots. 2.1

[86] Neal Seegmiller, Jason Gassaway, Elliot Johnson, and Jerry Towler. The maverick planner: An efficient hierarchical planner for autonomous vehicles in unstructured environments. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2018–2023, 2017. 2.3.1

[87] Gregory A Silver, OA-H Hassan, and John A Miller. From domain ontologies to modeling ontologies to executable simulation models. In *Simulation Conference, 2007 Winter*, pages 1108–1117. IEEE, 2007. 2.1

[88] Siddhartha S Srinivasa, Dave Ferguson, Casey J Helfrich, Dmitry Berenson, Alvaro Collet, Rosen Diankov, Garratt Gallagher, Geoffrey Hollinger, James Kuffner, and Michael Vande Weghe. Herb: a home exploring robotic butler. *Autonomous Robots*, 28(1):5–20, 2010. 1

[89] Ricarda Steffens, Matthias Nieuwenhuisen, and Sven Behnke. Multiresolution path planning in dynamic environments for the standard platform league. In *Proceedings of 5th Workshop on Humanoid Soccer Robots at Humanoids*, 2010. 1, 2.3.1

[90] Anthony Stentz. The focussed dˆ* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995. 2.2.1

[91] Benjamin Stephens. Humanoid push recovery. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pages 589–595. IEEE, 2007. 2.1

[92] Herbert K Struemper. Motion control for nonholonomic systems on matrix lie groups. Technical report, MARYLAND UNIV BALTIMORE, 1998. 4.5, 7.4.3

[93] Ioan Alexandru Sucan and Lydia E Kavraki. Mobile manipulation: Encoding motion planning options using task motion multigraphs. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5492–5498. IEEE, 2011. 2.3.1

[94] Wen Sun, Luis G Torres, Jur Van Den Berg, and Ron Alterovitz. Safe motion planning for imprecise robotic manipulators by minimizing probability of collision. In *Robotics Research*, pages 685–701. Springer, 2016. 2.2.2

[95] Jur Van Den Berg, Pieter Abbeel, and Ken Goldberg. Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7):895–913, 2011. 2.2.2

[96] Jur Van Den Berg, Rajat Shah, Arthur Huang, and Ken Goldberg. Ana*: anytime nonparametric a*. In *Proceedings of Twenty-fifth AAAI Conference on Artificial Intelligence (AAAI-11)*, 2011. 2.3.1

[97] Glenn Wagner and Howie Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3260–3267. IEEE, 2011. 2.3.1

[98] Minlue Wang and Richard Dearden. Planning with state uncertainty via contingency planning and execution monitoring. In *SARA*, 2011. 1, 2.2.1

[99] Jason Wolfe, Bhaskara Marthi, and Stuart J Russell. Combined task and motion planning for mobile manipulation. In *ICAPS*, pages 254–258, 2010. 2.1, 2.3.1

[100] Sung Wook Yoon, Alan Fern, and Robert Givan. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, volume 7, pages 352–359, 2007. 2.2.1

[101] Stefan Zickler. Physics-based robot motion planning in dynamic multi-body environments. Technical report, DTIC Document, 2010. 2.3.1

[102] Stefan Zickler and Manuela Veloso. Efficient physics-based planning: sampling search via non-deterministic tactics and skills. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 27–33. International Foundation for Autonomous Agents and Multiagent Systems, 2009. 2.1

[103] Stefan Zickler and Manuela M Veloso. Variable level-of-detail motion planning in environments with poorly predictable bodies. In *ECAI*, pages 189–194, 2010. 2.3.1

[104] Shlomo Zilberstein and Stuart J Russell. Anytime sensing, planning and action: A practical model for robot control. In *IJCAI*, volume 93, pages 1402–1407, 1993. 2.3.1

[105] Matthew Zucker, James Kuffner, and Michael Branicky. Multipartite rrts for rapid replanning in dynamic environments. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1603–1609. IEEE, 2007. 2.2.1

[106] Matthew Zucker, James Kuffner, and James A Bagnell. Adaptive workspace biasing for sampling-based planners. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3757–3762. IEEE, 2008. 2.2.1, 7.2