Plan-Time Multi-Model Switching for Motion Planning

Breelyn Kane Styler, Reid Simmons
Robotics Institute
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
{breelynk,reids}@cs.cmu.edu

Abstract

Robot navigation through non-uniform environments requires reliable motion plan generation. The choice of planning model fidelity can significantly impact performance. Prior research has shown that reducing model fidelity saves planning time, but sacrifices execution reliability. While current adaptive hierarchical motion planning techniques are promising, we present a framework that leverages a richer set of robot motion models at plan-time. The framework chooses when to switch models and what model is most applicable within a single trajectory. For instance, more complex environment locales require higher fidelity models, while lower fidelity models are sufficient for simpler parts of the planning space, thus saving plan time. Our algorithm continuously aims to pick the model that best handles the current local environment. This effectively generates a single, mixed-fidelity plan. We present results for a simulated mobile robot with attached trailer in a hospital domain. We compare using a single motion planning model to switching with our framework of multiple models. Our results demonstrate that multi-fidelity model switching increases plan-time efficiency without sacrificing execution reliability.

1 Introduction

Navigating robots must do so efficiently and reliably. Hospital robots are one example of this. Hospital robots navigate hallways to deliver medications and linens. These deliveries could be time sensitive requiring both efficient plan generation and reliability. Also, delivery robots save time for hospital workers to use towards other tasks. This time is not saved if employees constantly assist an unreliable robot to its destination. In this paper we aim to balance execution reliability while maintaining efficient planning time in motion planning.

Reliable execution requires planning models that accurately capture how the robot interacts with the environment. Complex environments require a more informed model. These models can be of arbitrary fidelity to better represent the interaction between the robot and its environment and often include higher dimensions, dynamics and differential constraints. However, planning with such models for the entirety of the environment is computationally expensive. Additionally, not all environments are equally complex, so for planning efficiency, model approximation may be necessary to achieve tractable global planning times.

Model approximations sacrifice fidelity for computation efficiency. These approximations are sufficient for simple regions in the task space. They are insufficient, however, for complex portions of the environment. For example, a model that considers detailed terrain vehicle interactions is unnecessary for flat open spaces, but needed for bumpy ground. This illustrates a non-uniform execution space where plan time is saved by leveraging multiple models. Model switching combines the robustness of a high-fidelity plan with the efficiency of an abstract representation by using higher fidelity models only when necessary.

Our work provides a framework for using a richer set of motion planning models than previous work. Our approach attempts to stay as long as possible in the lowest fidelity model applicable in order to decrease plan computation time without sacrificing execution quality. We do not introduce a new motion planner. We use existing motion planners in the context of switching between multiple robot models. We organize a given set of planning models into a directed hierarchical graph. An initial motion plan is generated in the lowest model. The approach then detects the parts of the lower fidelity plan that are infeasible for execution. The partial plans are repaired using re-planning through higher fidelity model selection. The model selection process autonomously selects the most applicable higher fidelity model in the hierarchy. This higher fidelity model is used to locally plan to an intermediate goal where the previous lower fidelity plan is resumed. This approach creates a mixed model plan.

We ran experiments in simulated hospital world with a differential drive robot. Our testing used 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. Our approach creates plans that maintain robustness, and take significantly less time to plan, overall, than planning from the start using the highest fidelity model. This improves performance over only using a lower fidelity model, and reduces planning times over only using the highest fidelity model.

2 Related Work

Models generated in the robotics community increase their robustness with fidelity. They inspire the use of a multi-model approach. For instance, there are models for humanoid balancing (Stephens 2007), physics based manipulation models (Dogar et al. 2012), and models for wheeled robots that more accurately capture real world trajectories (Seegmiller and Kelly 2014). Early work in motion planning also recognized that plan savings can be achieved by sacrificing execution feasibility at plan-time (Sekhavat et al. 1998). This work demonstrated motion plans for a multi-trailer system where lower model levels ignore more non-holonomic constraints than their higher planning levels.

The model choice to plan in creates an inherent tradeoff between plan-time and execution robustness. Many previous works use different forms of model switching to address this balance. Previous work in Variable Level-Of-Detail Planning, by (Zickler and Veloso 2010), relax the local space to ignore more complex details that occur far in the future. 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 (Kambhampati and Davis 1986), (Steffens, Nieuwenhuisen, and Behnke 2010), (Behnke 2004). 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 (Howard and others 2009) and work that only has two levels, such as a higher level global plan that guides a low level continuous planner (Knepper and Mason 2011). Hybrid planning also switches between a distinct discrete plan and more focused continuous planner (Plaku, Kavraki, and Vardi 2010) as in the SyClop planning framework. This also occurs in (Choi, Zhu, and Latombe 1989) where a contingency "channel" with many possible motion plans guides a lower-level potential field controller. Other work (Göbelbecker, Gretton, and Dearden 2011) 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.

Multi-modal and multi-stage work also contain fixed switching strategies. Work in (Hauser, Ng-Thow-Hing, and Gonzalez-Baños 2011) and (Hauser and Latombe 2009) 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 (Sucan and Kavraki 2011) 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 (Pivtoraiko and Kelly 2008). 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 (Gochev, Safonova, and Likhachev 2013) (Gochev et al. 2011) (Gochev, Safonova, and Likhachev 2012). 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. Our novelty is we use a complete hierarchy of models. Therefore, our algorithm contains an additional model selection stage. Also the discrete domains require a search in the higher space during tracking that is computationally more expensive than our checking phase.

3 Planning Models and Organization

Our planning models include information for generating a continuous motion trajectory from a start to goal configuration subject to vehicle and obstacle constraints. It contains a robot model, environment model, and collision checker. The robot model contains vehicle constraints. The environment model represents obstacles in the planning workspace.

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 system of motion equations of the form $\dot{x} = f(x(t), u(t), t)$. These equations describe the mapping of motions from $u(t)$ to $x(t)$, subject to kinematic and dynamic constraints. Robot models also describe the robot geometry which can contain aspects such as shape, mass, material, and coefficient of friction. The models we use in this paper consider only the shape attribute and contain both first and second-order differential motion equations

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 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.

A final model property is that 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.

The ability to translate one model into another without losing information forms a partial ordering. The organization forms a finite acyclic directed model graph $G(\vec{q}, E)$ connecting low fidelity models to higher fidelity models which often contain more constraints. Once all edge pairs 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 (Aho, Garey, and Ullman 1972). This is also known as the minimum equivalent graph, (Moyles and Thompson 1969), 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 1 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 1. Redundant edges are then eliminated, the right side of Figure 1, to create the directed graph's transitive reduction which maintains the longest edge paths.

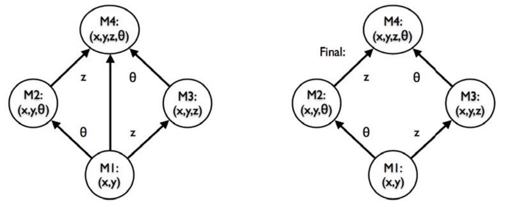


Figure 1: Transitive Reduction of original model graph.

4 Planning with Multiple Models

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.
   This is the plan checking stage where execution problems are determined. 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 graph to determine what model may be sufficient to repair the plan.
4. Multi tree re-planning
   For probabilistic completeness guarantees, re-planning starts from multiple previous points along the plan towards a goal set.

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
```

Motion Plan Generation

The purpose of this work is not to construct a new motion planner. We use Rapidly Exploring Random Trees (RRTs) from the sample based motion planning community to generate motion plans (LaValle and Kuffner 2001) since RRTs can handle various models, including those with differential constraints and dynamics.

We start by planning in a default model space, typically [x,y] (Algorithm 1, line 1). The robot's task is to navigate from a start to goal state. Our system uses an environment map to generate a plan from start to goal in the lowest applicable fidelity model. This is the lowest possible model that successfully generates an initial global plan given the current environment. This produces a plan in the form of a series of waypoints, such as that shown in Figure 2 (a).

Feasibility Detection

The plan is then checked in the highest fidelity model to determine parts that might need repairing (Algorithm 1, line 6). Even though the high 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. If the plan is not feasible, the infeasible plan segment is sent to the model selector. In Figure 2 (b), an infeasibility is detected between waypoints four and five.

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

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.

Model Selection

The model selector then determines the appropriate model to re-plan in. For efficiency, we plan from waypoint four towards waypoint five to get back on the previous path as early as possible, Figure 2 (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 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 back to the highest available model.

Our model selection process (Algorithm 2) uses Breadth First Search to explore the model graph. For each model, we first test the infeasible plan segment in that new model space. If the testing is unsuccessful, it means the model has information not present in the original model used to generate the plan, and is then selected as the re-plan model.

It is important to note 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.

As an example of how the model selector works (Algorithm 2), assume that it starts with the first child of the M_1 = [x,y] model which is the M_2 = [x,y,θ] model. 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 child of [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 select it as the model to use for re-planning. In this approach it is possible to produce a final plan that effectively skips between model tree levels when choosing the next model (in this example, we skipped from [x,y] to [x,y,z,θ]).

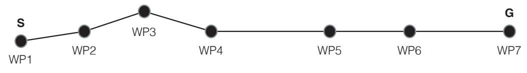
Multi Tree Re-planning and Intermediate Goals

Infeasibilities 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 guaranteeing probabilistic completeness. To accomplish this, waypoints along the globally maintained plan, Figure 2 (a), before the infeasibility are considered the starts of re-plan trees. Similarly, remaining waypoint nodes after the infeasibility are considered intermediate goals.

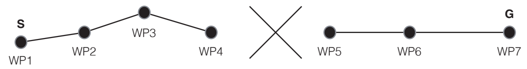
The remaining waypoints along the path, Figure 2 (c), form a set of intermediate goals. We sample from this multi-goal set based on an inverse distance metric from the previous infeasibility to encourage returning to re-using the previous path. This allows the algorithm to concurrently plan to all remaining intermediate goals, and be more efficient with path re-use. Using waypoints as a cache was also done pre-

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.

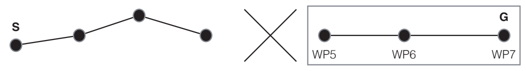
```
 1: function MODELSELECTOR(p, b4failIndex, afterfailIndex)
 2:     mLast = findModelFor(p.getNode(b4failIndex));
 3:     m = mLast;
 4:     p = resizePlan(p.getNode(b4failIndex), p.getNode(afterfailIndex));
 5:     setUsedModel(m);                    ▷ Last model used becomes root node.
 6:     m = 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
```



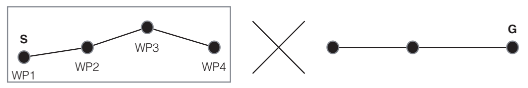
(a) Example global path.



(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.

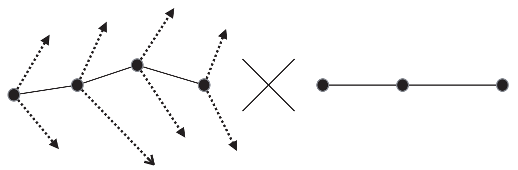

(e) Re-planning concurrently plans from multiple start trees towards a set of intermediate goals.

Figure 2: Localizing around the infeasible area by generating multiple re-plan trees towards intermediate goals, in the new selected model, rather than re-planning to the original goal.

viously, (Bruce and Veloso 2002). We expand this for multi-fidelity nodes and plan reuse.

For completeness guarantees, we also plan from all previous waypoints. In Figure 2 (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 2 (e), also uses an inverse distance metric from the infeasibility to probabilistically sample what tree to expand next.

The initial 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. Since all models have at least one common ancestor (such as M4 in Figure 1), previous waypoints along the global path are elevated to this common model before re-planning occurs. Additionally, the remaining nodes after the infeasible area are translated to the currently selected model and set as possible goals.

The first re-plan tree that connects to an intermediate goal (Motion Plan Generation) creates a partial plan. The new partial plan is then 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 process repeats until the feasibility detection stage does not find any more impediments. If this is the case, the full global plan is executable

Probabilistic Completeness Guarantees We assume a model hierarchy exists where every model has at least one common ancestor of higher fidelity. Therefore, there exists a highest fidelity model that is an ancestor of all models. Additionally, we assume the model implementations use steering functions for the control input (Kunz and Stilman 2015).

Definition:
For probabilistic completeness, the probability that the highest fidelity model tree will find a solution approaches one as the number of tree states approaches ∞. To illustrate this, we must show in the worst case our algorithm generates a planning tree from the start of the space in the highest fidelity model in finite time.

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 (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. If the partially re-planned path (Algorithm 1, Line 8) continues to detect infeasibilities, then by BFS (Algorithm 2, line 6), the re-plan tree will continue to get elevated until reaching M_h. 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). 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.

5 Implementation

Differential Drive Robot Models with Trailer

We use seven different wheeled robot models in our testing.

1. $\vec{q} = [x, y] = XY$
   The first model is geometric with no control inputs. It is collision checked in a 2D environment model.
2. $\vec{q} = [x, y, \theta] = XY\theta$
   This model includes θ and constrains motions to s-curves. The linear velocity is held constant. It is collision checked in x, y, and θ environment model.
3. $\vec{q} = [x, y, \theta, \theta_{trailer}] = XY\theta\theta_1$
   This model generates the same motions as model 2, but also includes calculating motion for a trailer. The trailer's motion is calculated using $\theta_{trailer}$. The collision checker checks x, y, θ, and the robot 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. Collision checking now includes a time variable used to index 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 4, but now includes additional collision checking and motion for a 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 (u) variables become part of the state. It collision checks in x, y, θ, and a time state which indexes 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 an extension of model 6 which includes the trailer.

Models with sampled linear and angular velocities use the standard differential drive motion equations.

$$\dot{x} = u_v \cos(\theta)$$
$$\dot{y} = u_v \sin(\theta)$$
$$\dot{\theta} = u_w$$

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.

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

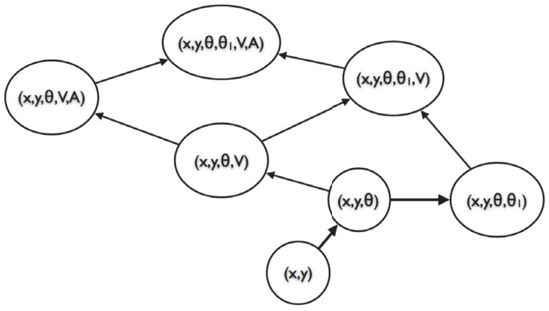


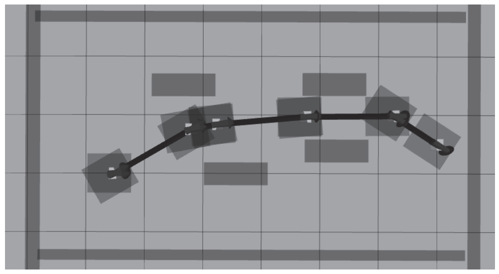
Figure 3: Transitive reduction of full model graph.

Different models create different motion trajectories as shown in Figure 4. The XY path (Figure 4(a)) contains straight line geometric motions. The XY$\theta\theta_1$ path (Figure 4(b)) models the trailer and follows s-curve motions. Finally, the XYθVA path (Figure 4(c)) varies accelerations creating a smoother trajectory.

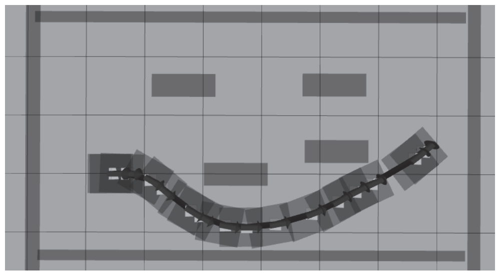
Translation

Plan translation is also 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, 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. This is one step of plan translation. Examples of passive states include the 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 plan fidelity to account for what the controller does when checking paths. For example, our [x y] model requires turn in place actions to be added for higher fidelity spaces where θ is actually modeled. This creates feasible controls for the planner to check. Additionally, the plan is translated to more closely match the controller by augmenting angular velocity to stay along the path during checking. This forces a check of the path that most closely resembles the robot's actual path during execution.



(a) Example XY path: notice that this lower model has turn-in-place waypoints that take it through a tight hallway.



(b) Example XYθVA: notice that this higher model chooses to plan around the gurneys.

Figure 4: Examples of paths generated using different models.

Controller

Angular and linear velocity controls are sent for the robot to execute. Controls are matched by having separate PID controllers for each wheel. The robot executes these controls until it is within epsilon of the next waypoint or it crosses a line segment that goes through the next waypoint perpendicular to the robot's heading. The robot follows smoother curves by modifying the commanded angular velocity to be a function of the angular error when the angular error is greater than some epsilon. It adjusts the robot to turn towards the next waypoint when within some small linear distance (0.6) and then turn towards the next planned theta within an even smaller linear distance (0.1).

6 Hospital Simulation Experiments

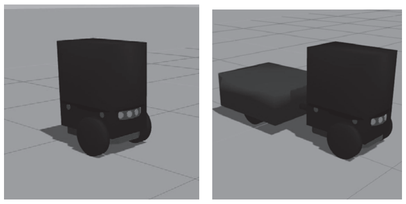
We ran single model runs over various environments. These single runs were compared for plan time and success rates with the multi-model switching strategy. Below, we describe the different environments for each experiment.

Robot

The robot geometry models are shown without (Figure 5 (a)) and with the trailer (Figure 5 (b))).

Environments

The real world is represented by the Gazebo simulator (http://gazebosim.org/) which models dynamics by simulating rigid-body physics. We model a typical hospital environment with automated swinging doors and gurneys in the



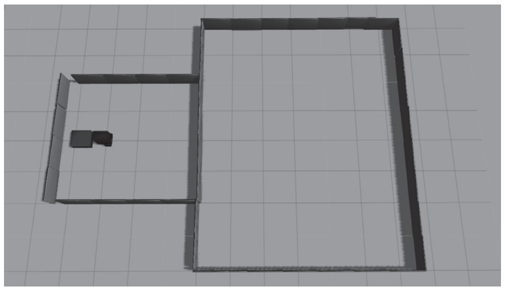
(a) Robot Model. (b) Robot Model with Trailer.

Figure 5: The robot model.

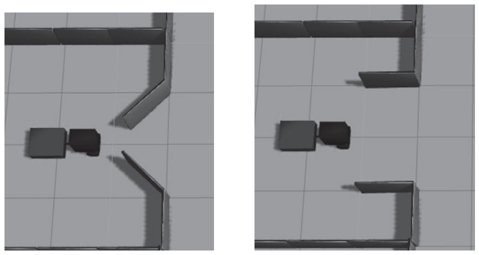
hallways. The swinging doors are meant to show the importance of planning with velocity and acceleration and the gurney world shows the importance of modeling the trailer through turns.

Swinging Doors The pictures in Figure 6 show the three swinging doors stages checked in the collision checker when time is part of the state space. The door is activated when the robot is in front of the doors within 2m of the x-axis and 1m of the y-axis.

An example path generated in this world using the highest acceleration model is shown in Figure 7. 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.



(a) The doors are closed before and after being opened.



(b) The doors partially opening or closing. (c) The doors fully opened.

Figure 6: Swinging doors world and various stages of the door opening and closing.

Gurney The gurney world contains multiple hospital gurney obstacles placed in a hallway that the robot must plan through. (Figure 8).

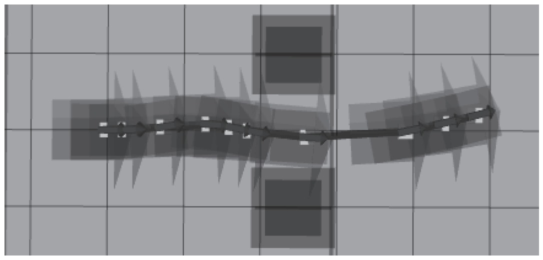
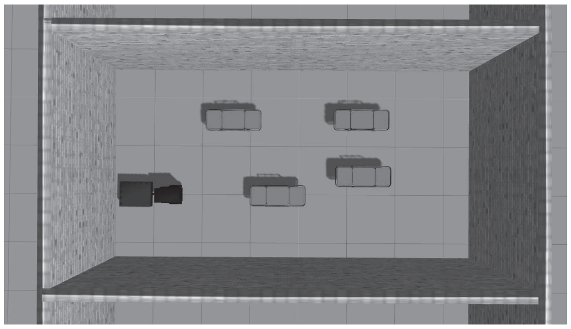
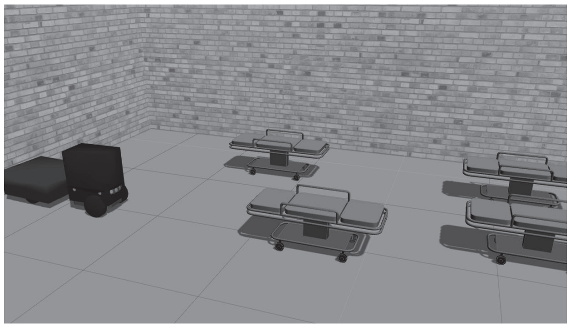


Figure 7: A path generated in the swinging doors world using model $XY\theta\theta_1VA$.



(a) Gurney world.



(b) Gurney world zoomed in.

Figure 8: A hospital environment with multiple gurneys.

7 Results

We ran experiments in the simulator world for each environment as shown in Table 1 and Table 2. This was done for all seven models, with and without switching, for 100 runs.

To produce good plans, we generated 20 RRT plans and chose the best (shortest) one for execution. Note that we are constructing scenarios that are more likely to fail in the lower models to show the efficacy of our approach. In standard execution we would expect the initial model (in this case [x,y]) to fail rarely.

Swinging Door

Table 1 contains the swinging doors environment runs. It shows the percentage of successful executions to the goal for each model (without switching), switching among all models, and their average planning times. We see that as the models increase in fidelity the mean plan-time over 20 runs increases. Models for this world that sample velocity and acceleration controls incur higher plan-times. This is due to the additional collision checking necessary for the time states with the swinging doors in these higher models.

Table 1: Swinging doors results for no switching single model runs and switching in swinging doors world.

Model	success %	plan-time mean(s)	plan-time std dev
Test (From x=-2, y=-2 to x=3.0, y=-2.0).			
[x, y]	69%	0.08	0.03
[x, y, θ]	77%	2.76	0.7584
[x, y, θ, θ_1]	53%	2.86	0.9416
[x, y, θ, V]	98%	11.13	4.91
[x, y, θ, θ_1 ,V]	100%	44.65	14.29
[x, y, θ, V, A]	99%	14.39	5.97
[x, y, θ, θ_1, V, A]	100%	52.7	19.5
Test (From x=-2, y=-2 to x=3.0, y=-2.0). Swinging doors.			
Switch all	98%	6.89	12.27

Gurney

Table 2 contains the gurney environment runs. It also shows the percentage of successful executions to the goal for each model (without switching), switching among all models, and their average planning times.

The gurney switching results had higher average mean planning times than expected, but this is due to the breadth first order during model selection. Multiple models may be applicable to infeasibilities found, and since we use breadth first search, re-planning occurred first in the non-trailer models. This increased average planning time. This suggests future work on a more informed method for choosing model selection based on the environment rather than breadth first search.

Table 2: Gurney results for no switching single model runs and switching in gurney world.

Test (From x=-2.5, y=-2.0 to x=3.0, y=-1.5).			
[x, y]	53%	0.08	0.05
[x, y, θ]	42%	2.47	0.49
[x, y, θ, θ_1]	95%	4.21	0.62
[x, y, θ, V]	88%	6.98	2.31
[x, y, θ, θ_1 ,V]	98%	15.48	6.7
[x, y, θ, V, A]	87%	12.84	1.95
[x, y, θ, θ_1, V, A]	99%	27	6.38
Test (From x=-2.5 , y=-2.0 to x=3.0 to y=-1.5). Gurney World.			
Switch all	94%	8.8	10.36

We also found that seeding plans with the lowest model (with already initial poor success rates) sometimes created paths to areas where higher models infrequently planned towards. This caused more re-planning and higher average plan times. Re-planning in very constrained spaces increased plan time due to the difficulty sample based algorithms have in tight spaces such as hallways. This can be improved by varying the multi tree expansion sampling metric (currently an inverse distance).

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 the 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 in the gurney world where we are not effectively modeling trailer dynamics, such as mass, which could increase or decrease trailer swing 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 switching success rates. We are currently working to match the highest model checker as closely as possible to the underlying real world controller. We are confident that this would minimize the occurrence of these values aligning our results with the theory of properly balancing low plan-times with high execution success rates.

8 Conclusions and Future Work

We presented an approach which leverages a multi-fidelity model graph to produce a mixed-model plan. This plan finds a good balance between decreased planning time and increased robustness by giving the robot the ability to re-plan in a more detailed model to provide a more accurate representation of reality. Just as the robot's operation space is non-uniform, containing a mix of simple and complex areas, our algorithm tries to capture the space with a mix of low and high fidelity models. Our tests attempt to improve computation time and achieve similar success performance (of the higher models) in this example domain. Our switching tests also localize planning around the infeasible location.

The results also show there is not always a single best model to use, but rather that it depends on the situation. For example, even though modeling the differential constraints of the robot is higher fidelity than a purely geometric model, that model will not help if the real problem is not considering the z dimension (e.g. if overhangs exist in the world), or the trailer's shape in tight turns. This is why it is important to have a separate model selection stage that can reason about which model fidelity should be used for repair.

References

Aho, A. V.; Garey, M. R.; and Ullman, J. D. 1972. The transitive reduction of a directed graph. *SIAM Journal on Computing* 1(2):131–137.

Behnke, S. 2004. Local multiresolution path planning. In *Robocup 2003: Robot Soccer World Cup VII*. Springer. 332–343.

Bruce, J., and Veloso, M. 2002. Real-time randomized path planning for robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, 2383–2388. IEEE.

Choi, W.; Zhu, D.; and Latombe, J.-C. 1989. 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*, 78–86. IEEE.

Dogar, M. R.; Hsiao, K.; Ciocarlie, M.; and Srinivasa, S. S. 2012. Physics-based grasp planning through clutter. In *In RSS*. Citeseer.

Göbelbecker, M.; Gretton, C.; and Dearden, R. 2011. A switching planner for combined task and observation planning. In *AAAI*.

Gochev, K.; Cohen, B.; Butzke, J.; Safonova, A.; and Likhachev, M. 2011. Path planning with adaptive dimensionality. In *Fourth Annual Symposium on Combinatorial Search*.

Gochev, K.; Safonova, A.; and Likhachev, M. 2012. Planning with adaptive dimensionality for mobile manipulation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2944–2951. IEEE.

Gochev, K.; Safonova, A.; and Likhachev, M. 2013. Incremental planning with adaptive dimensionality. In *Twenty-Third International Conference on Automated Planning and Scheduling*.

Hauser, K., and Latombe, J.-C. 2009. Multi-modal motion planning in non-expansive spaces. *The International Journal of Robotics Research*.

Hauser, K.; Ng-Thow-Hing, V.; and Gonzalez-Baños, H. 2011. Multi-modal motion planning for a humanoid robot manipulation task. In *Robotics Research*. Springer. 307–317.

Howard, T. M., et al. 2009. Adaptive model-predictive motion planning for navigation in complex environments.

Kambhampati, S., and Davis, L. 1986. Multiresolution path planning for mobile robots. *Robotics and Automation, IEEE Journal of* 2(3):135–145.

Knepper, R. A., and Mason, M. T. 2011. Improved hierarchical planner performance using local path equivalence. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, 3856–3861. IEEE.

Kunz, T., and Stilman, M. 2015. Kinodynamic rrts with fixed time step and best-input extension are not probabilistically complete. In *Algorithmic foundations of robotics XI*. Springer. 233–244.

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

Moyles, D. M., and Thompson, G. L. 1969. An algorithm for finding a minimum equivalent graph of a digraph. *Journal of the ACM (JACM)* 16(3):455–460.

Pivtoraiko, M., and Kelly, A. 2008. Differentially constrained motion replanning using state lattices with graduated fidelity. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, 2611–2616. IEEE.

Plaku, E.; Kavraki, E.; and Vardi, M. Y. 2010. Motion planning with dynamics by a synergistic combination of layers of planning. *Robotics, IEEE Transactions on* 26(3):469–482.

Seegmiller, N., and Kelly, A. 2014. Enhanced 3d kinematic modeling of wheeled mobile robots.

Sekhavat, S.; Svestka, P.; Laumond, J.-P.; and Overmars, M. H. 1998. Multilevel path planning for nonholonomic robots using semiholonomic subsystems. *The international journal of robotics research* 17(8):840–857.

Steffens, R.; Nieuwenhuisen, M.; and Behnke, S. 2010. Multiresolution path planning in dynamic environments for the standard platform league. In *Proceedings of 5th Workshop on Humanoid Soccer Robots at Humanoids*.

Stephens, B. 2007. Humanoid push recovery. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, 589–595. IEEE.

Sucan, I. A., and Kavraki, L. E. 2011. Mobile manipulation: Encoding motion planning options using task motion multigraphs. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 5492–5498. IEEE.

Zickler, S., and Veloso, M. M. 2010. Variable level-of-detail motion planning in environments with poorly predictable bodies. In *ECAI*, 189–194.