Robust Efficient Robot Planning through Varying Model Fidelity

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

Abstract

Long-term robot autonomy requires reliable execution success. Execution success improves by modeling the real world accurately during planning time. Unfortunately, computing a full plan in highly accurate models is often intractable since accurate world models increase the dimensionality of the planning space. Additionally, planning in the highest model is unnecessary for simple uncluttered parts of the environment. Current planners achieve tractable planning times by using approximate models. However, these approximations reduce accuracy resulting in decreased execution success. We introduce an approach that balances planning time efficiency while maintaining execution success. This is accomplished by leveraging a varying fidelity model hierarchy. The approach identifies infeasible execution locations and uses a model selection process to locally re-plan in the minimum fidelity model necessary to circumvent the infeasibility. This effectively generates a single, mixed-fidelity plan. We evaluate the approach on a simulated differential drive robot that navigates a constrained environment. The robot maintains its rate of successful task completion while conserving computation resources by switching from lower fidelity models only when needed.

1 Introduction

Autonomous robots leverage planning and models to intelligently operate in the real world. Furthermore, autonomous robots deployed for long periods need to be robust to failure. They can minimize runtime failures by reasoning about models that capture execution feasibility at plan-time. Models can be arbitrarily complex to better represent the interaction between the robot and its environment. These high fidelity models often include higher dimensions, dynamics and differential constraints. However, planning with such complex models is computationally expensive. Therefore, model approximation may be necessary to achieve tractable global planning times for complex tasks. The loss of model fidelity from these approximations leads to increased failure rates in complex locales of the environment where lower fidelity models are insufficient.

Model approximations sacrifice fidelity for computation feasibility. These approximations are sufficient for simple regions in the task space. For example, a model that considers detailed terrain vehicle interactions is unnecessary for large flat open spaces. They are insufficient, however, for complex portions of the environment. For that reason, the robot's non-uniform execution space suggests plan savings by leveraging multiple models. Our work attempts to stay as long as possible in the lowest fidelity model applicable in order to decrease plan computation time without sacrificing execution quality.

Our approach organizes a given set of planning models into a directed hierarchical graph. These varying fidelity models, when combined, approximate the continuous highest possible real-world model. The approach attempts to detect the parts of the lower fidelity plans that are infeasible for execution and repair them using re-planning through higher fidelity model selection. The model selection process uses prior plans to autonomously select 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.

Our approach increases robot robustness by giving robots the ability to autonomously decide to re-plan in higher fidelity model representations that inherently incorporate more information. This will improve performance over only using a lower fidelity model, and it will reduce planning times over only using the highest fidelity model. 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.

We ran a series of experiments, in simulation, with a differential drive robot, and used wheeled mobile robot motion planning models for our testing. 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.

2 Related Work

Model generation is prevalent in the robotics community. For instance, there are models for humanoid balancing (Stephens 2007), physics based models for manipulation planning (Dogar et al. 2012), and models for wheeled robots that more accurately capture real world trajectories (Seegmiller and Kelly 2014). These previous works demonstrate different model types that increase robustness as the fidelity increases. These works inspire the use of a multi-model approach, but they use hard coded rules for knowing what model to use, and do not include a switching strategy.

Adjusting the resolution in a plan could be considered a form of model fidelity 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 and changes the resolution of both the state and action spaces.

Our use of hierarchical models is inspired by other hierarchical approaches (Fernández-Madrigal and González 2002), (Barbehenn and Hutchinson 1995) as well as hierarchical approaches in task motion planning that recognize the need to consider lower-level motion plans in task space for plan success (Wolfe, Marthi, and Russell 2010), (Kaelbling and Lozano-Pérez 2011).

In addition to a model hierarchy our work uses re-planning to switch between varying fidelity models. This is motivated by previous work in re-planning. Re-planning is a technique that has been in symbolic planning since early mobile robots (Fikes, Hart, and Nilsson 1972). The execution monitoring portion of the planner PLANEX included ways to regenerate unsuccessful plan actions with different arguments. Re-planning is also prevalent in the motion planning community. Motion planning uses re-planning in incremental search (Stentz 1995) (Koenig and Likhachev 2002), and sampling based algorithms (Bruce and Veloso 2002) (Ferguson, Kalra, and Stentz 2006). Our approach re-plans between different hierarchical levels of the configuration space and workspace including reasoning about differential constraints

Many previous works 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.

Related work, such as (Howard and others 2009), uses fixed strategies by focusing on when to switch between levels of detail. This is apparent in work that only has two levels, such as a higher level global plan that guides a low level continuous planner (Knepper and Mason 2011). Similarly hybrid planning switches between a distinct discrete plan and more focused continuous planner (Plaku, Kavraki, and Vardi 2010) as occurs 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, that generates an overall strategy, and more expensive decision-theoretic planning for abstracted sub problems.

Multi-modal and multi-stage work also contain fixed switching strategies. They switch between different planner types based on the modal discretization of the motion planning domain. Multi-modal motion planning for humanoid manipulation (Hauser, Ng-Thow-Hing, and Gonzalez-Baños 2011) generate trajectories within a single mode and reason about the motion transitions as targets. The levels of mode "classes" are pre-defined with such modes as: walking, reaching left, reaching right etc, each with their own constraints and dynamics. Work by (Hauser and Latombe 2009) also divides the space into modes based on the different dimensions in motion configuration space. They sample transitive connections between modes but do not reason explicitly about the detail they are switching to. Lastly, work by (Sucan and Kavraki 2011) for task motion multigraphs also contain predefined points where switching occurs. This work decides between left and right arm motion trajectories which could be viewed as different models. The decision of when to switch is predefined between tasks and the selection criteria is based on computation time. Our model selection criteria reasons about task success and the point when to switch to a new model is not predefined.

A work that is more similar to ours graduates motion primitive fidelity along a state lattice (Pivtoraiko and Kelly 2008). They change the fidelity between replans 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 amount of 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 also 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. In addition, our models incorporate more than just the state space dimensionality.

3 Approach

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

1. Plan generation.
   Here 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. [1]

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.

Using the motion planning domain as an example, 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 (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, 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 back to the highest available model. The re-planning model is selected by re-testing the partial plan segment, which needs repair, in 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.

The algorithm then cycles back to stage one. A new plan is generated in the selected model. 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. This creates a partial plan. The new partial plan is then merged back into the global plan. 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.

The next sections describes our definition of model as well as how the varying fidelity models are organized in a directed hierarchical graph. Section 3.3 provides more details about the robust plan generation algorithm.

3.1 Models

Note that the model space is separate from the underlying robot controller. To illustrate this separation, imagine a person racing a car 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 that is necessary for controlling the car. Similarly the robot's models provide decision making options through the generation of possible plans. The plan then needs to be translated into control inputs for the robot to execute [2] (see section 4.1).

When we use the word *model* we describe a three-tuple $\{S_r, S_e, A\}$ consisting of:

- S_r : set of robot states $\{s_0 \dots s_\infty\}$
  Example robot states: position, orientation, mass, wheel slip constraints, etc.
- S_e : set of environment states $\{s_0 \dots s_\infty\}$
  Example environment states: position and orientation of obstacles, 3D/2D representation, static or dynamic obstacles, etc.
- A : set of actions $\{a_0 \dots a_\infty\}$
  Example action set: planar motions, equations of motion that include control inputs and differential constraints, motion primitives, or other trajectory generation methods, etc.

Each model includes states and actions that allow the generation of a plan within that model space. The action space varies from geometric spatial actions to differential equations of motion that include input controls. The environment representation (2D vs 3D) also varies through the state space. Where the models come from is not relevant to this paper. We assume they are all provided to us (see, for instance, (LaValle 2006)).

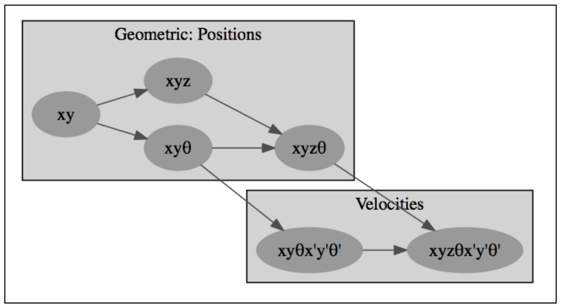


Figure 1: Chassis Directed Model Graph: The graph evolves from (roughly) left to right, with the root at [x,y]. Note: States listed do not encompass the entire model and exist only as labels for the model nodes.

3.2 Model Organization

Ordering the models hierarchically is important for model selection. For a model to be higher fidelity than another, it could be a direct superset by including more states in the space (higher dimensions), or containing more control inputs. The more correct a model is at representing the real

[1]For an execution time-only approach the robot can trigger a failure detection during runtime. This makes sense if failures are non-detrimental and easily detectable. For this work we focus on a plan-time approach since we are testing in simulation and not reasoning explicitly about uncertainty.

[2]Translation is only necessary for the geometric planar models for our specific plan following controller. This translation step can be avoided by planning with directly executable feasible motion primitives, as was demonstrated in previous state lattice motion planning work (Pivtoraiko and Kelly 2005). Additionally, some plan following controllers are robust enough to follow geometric plans without translation. This is demonstrated in Big Dog (Raibert et al. 2008) where the robot successfully executes plans that were generated on a two-dimensional grid.

world (such as modeling slip), or the larger the space considered, the higher fidelity the model is.

Our work makes an assumption that lower fidelity models can be translated into higher models at each layer. This allows the lowest model to be translated into the highest model through each layer of the graph (see Section 4.1). This translation capability is necessary for 1) checking the full plan for errors, 2) model selection, and 3) connecting partial plans with intermediate goals.

As a concrete example, Figure 1 shows a model graph with models from the motion planning community for a two-wheel differential drive robot. The node names are just indicative - the actual models incorporate more information than just a description of state.

As the models change fidelity the environment and/or the action space is changed. For example, the added z-dimension changes only the collision checker to consider three dimensional objects, an environment change. This is because the chassis cannot actuate in the z-dimension, so the addition does not effect the action space. Adding the θ dimension changes both the state and action space. Collision checks are now done in θ and the underlying controller is constrained to s-curve motions. The specific models used in the experiments are described in more detail in Section 4.1.

3.3 Robust Plan Generation

We start by planning in a default model space, typically [x,y], and check the plan in the highest fidelity model. If the plan is feasible, it is sent to the robot for execution. If the plan is not feasible, the infeasible plan segment is sent to the model selector. The model space used for the plan segment that needs repair (initially the [x,y] model) is then the first node to search from in the model graph.

Algorithm 1 Robust Plan Generation

```
1: setupPlanSpace()
2: [p , planResult] = generatePlan(m);
3: globalPlan = SAVEPLAN(p, globalPlan);
4: if planResult == success then
5:     tm = translateToModel(p, highest);
6:     [planCheck, b4repair, afterrepair] = propagateWhileValid(tm, highest)
7:     if planCheck == infeasible then
8:         m = MODELSELECTOR(globalPlan, b4repair, afterrepair);
9:         Goto line 1
10:    else
11:        executeResult = sendToRobot(globalPlan);
12:        if executeResult == failure then Robot failed in execution.
13:        else
14:            Robot made it to goal!
15:        end if
16:    end if
17: else
18:     Failed to find plan.
19: end if
```

Our model selection process 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. Re-planning from the segment start to intermediate goals (waypoints) is then performed in that model. If a successful plan is found, it is merged back into the global plan (Algorithm 4) to be re-checked. This strategy is detailed in Algorithm 1.

As an example of how the model selector works (Algorithm 2), assume that it starts with the first child of the [x,y] model which is the [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 [x,y,z], and again test the previous plan in this higher fidelity model. If the plan again succeeds, we then try the [x,y,z,θ] model. If testing the previous plan finally does not succeed, we assume this model captures the space of the impediment and we select it as the model to use for re-planning from the start of the infeasible segment. Based on this approach, it is possible to produce a final plan that can effectively skip between model tree levels when choosing the next model (in this example, we skipped from [x,y] to [x,y,z,θ]).

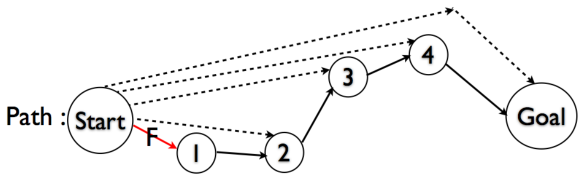
Algorithm 2 The model selector does a Breadth First Search by testing old infeasible plan segments in higher fidelity model spaces until the old plan fails. If the old plan fails, this indicates the model contains information that may be relevant to the impediment and that is the model chosen for re-planning.

```
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
```

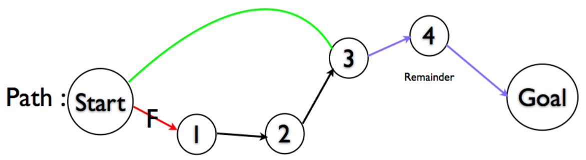
Intermediate Goals Instead of re-planning in a model all the way to the original goal, the previous plan is potentially reused by planning to intermediate goals. In particular, the remaining nodes after the infeasible area are translated to the currently selected model and set as possible goals. This effectively creates a goal set to plan to (Algorithm 3). If we successfully plan to an intermediate goal we can switch back to using the original plan for the remainder of the plan.

The idea of using waypoints as a cache was also done in work by (Bruce and Veloso 2002). We expand this for multi-fidelity nodes and plan reuse. For instance, in Figure 2 a goal set is created for all unachieved waypoints (1 through goal). The planner then re-plans to this goal set and finds waypoint 3 is successful. The remaining plan, with this new partial plan, is sent for the robot to execute.

Planning to an intermediate goal localizes around the infeasible location in order to plan in higher fidelity models for less time. The resulting plan to be executed can contain nodes from different models, creating a mixed model plan.



(a) After model selection, elevate remaining nodes (node after failure to original goal) to match model selected. Plan to goal set (1,2,3,4, and original goal)



(b) Successful re-plan to intermediate goal 3, remainder of plan sent from there to original goal

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

Algorithm 3 To plan to intermediate goals it is necessary to elevate the planning model of the remaining nodes in the plan, after the infeasibility, to be the same as the current model. This allows the planner to plan to a possible set of goals rather than a single goal.

```
1: procedure SETGOALS
2:     node = findNodeB4Repair(globalPlan);
3:     m = node.getModel();
4:     setGoals(translateRemainingNodes(globalPlan,m));
5: end procedure
```

4 Implementation

This section presents the models from the motion planning community that we used for our experiments. We also show translation between the models and how collision checking varies based on the model. Section 4.2 describes the low-level robot controller, and the collision monitoring techniques we applied for execution.

In our current implementation, the real world is represented by the Gazebo simulator (http://gazebosim.org/) which models dynamics by simulating rigid-body physics. Our test environment contains three dimensional overhangs of different heights as well as a sliding door that periodically opens and closes every 10 seconds. Pictures of the door open and closed are shown in Figure 4 (a) and Figure 4 (b). We are setting up an environment more complex than the lower models can handle for the sake of demonstrating the benefits of different model use.

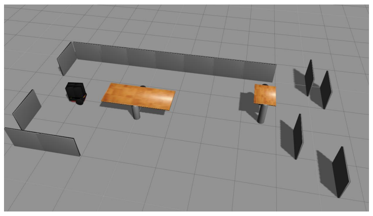
Plans are generated using the Open Motion Planning Library (OMPL) (Şucan, Moll, and Kavraki 2012). While the underlying motion planner is not so important, we use Rapidly Exploring RandomTrees (RRTs) to generate motion plans (LaValle and Kuffner 2001) since RRTs can easily handle various models, including those with differential constraints and dynamics. We defined the RRT distance function to primarily use the x and y components.

For intermediate goals, θ is added to the distance function to make sure mixed plans with connection points for θ-models align properly with previous plans. Also, as a heuristic in RRT planners, the goal is sampled with some probability. We probabilistically weight the remaining intermediate goals linearly based on the inverse of their distance from the repair segment start. This biases intermediate goals to those closest to the detected impediment in order to re-use more of the remaining plan when possible.

Figure 3 shows the plan-time algorithm generation for the navigation motion planning domain. Each generation stage of the plan is shown resulting in a final plan which merges

Algorithm 4 A global plan saves the proper model with each plan node. Partial paths are merged back into the global plan. When planning to intermediate goals the remainder of the old plan must also be added to the partial 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
```


(a) Open door

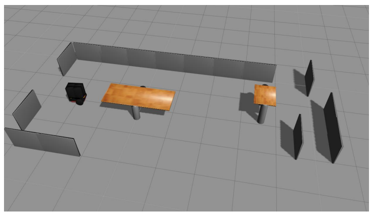

(b) Closed door

Figure 4: The world with a door that opens periodically on the right side (every 10 seconds). A large center overhang the robot cannot go under, and a smaller overhang the robot can go under.

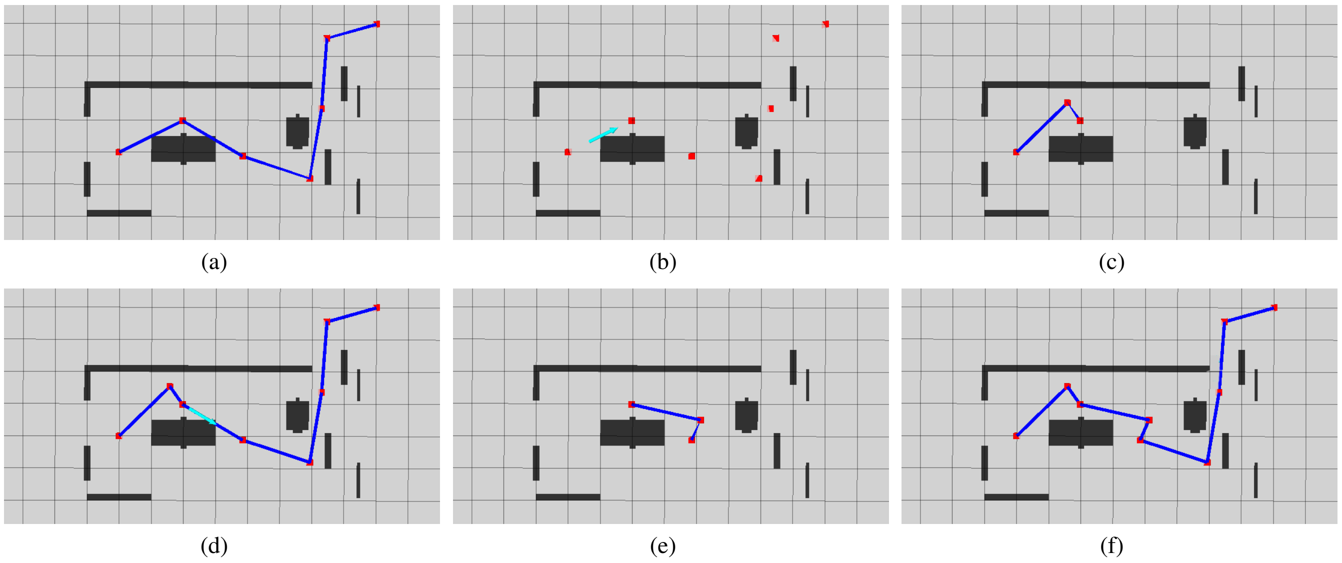


Figure 3: The dark blue lines connect the waypoints in the path. A plan is generated in the lowest fidelity space (a). The plan is translated into the highest model and a collision is detected in the plan run in the highest model (b). The light blue arrow shows the detected collision. The model selector re-plans in a higher selected model (this case [x,y,z]) (c). This partial plan is merged back into the global plan (d), where another collision is detected when checked in the highest model. The model selector re-plans a partial plan in the selected model[x,y,z] (e). The final mixed fidelity plan with no more detected repairs is constructed (f). Note: overhangs projected for illustration purposes.

partial plans from different models.

4.1 Models Used

Our robot has only one subsystem, the chassis, which is differential drive and can translate (in 2D x, y space where x and y are coupled) and rotate in theta (x, y, θ space). The model state focuses on positions in x, y, z space and SO2 space. Time is added to the state space for models that have velocities, which allows us to represent dynamic obstacles that can change position periodically.

For the action space, the underlying ordinary differential equations specific to a non-holonomic two wheeled vehicle were constructed based on the unicycle model (LaValle 2006). Models with proper velocity space sample the right and left wheel velocities separately rather than just sample a linear velocity. Table 1 lists the state, controls, and action spaces for the models we implemented. The models we selected are chosen explicitly to illustrate the benefits of switching. Models for real world robot use are expected to be more sophisticated. Figure 5 shows different plans generated using different models.

The robot's model fidelity changes in two ways. The fidelity of the physical robot changes for collision checking during planning, and the fidelity of robot motions vary to better capture the underlying controller. In particular, models without z treat the environment as 2D, in terms of collision checking, while models with z do full 3D collision checking. [3]

Collision Checker We check collisions only if obstacles inflated by the robot's bounding sphere intersect the current state. Table 2 describes how collision checking changes for different models.

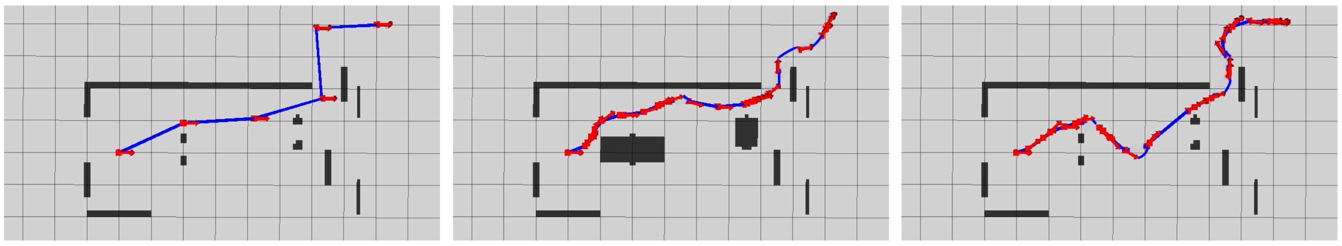
For models with velocity, we use an additional time variable in the state to properly index the environment we are checking against (doors that are closed or open). Models without velocity assume the doors are open.

To help account for uncertainty in the robot's motions, we create a small buffer around the robot by increasing the robot footprint 6% in the x and y directions (3% on each side).

Translating Between Models Translation functions exist to convert states and actions in lower fidelity models to higher fidelity (see Table 3 for details). For models that generate actions that are not directly executable by the controller, additional controls are added. This is necessary for the purely geometric models that have planar motions. For instance, the [x,y] model space generates plans that have the robot turn-in-place and follow a straight line to the next waypoint. The translation function adds extra waypoints to enable the controller to perform turn-in-place actions.

For checking translated plans in higher models we use the same propagation function used in the RRT to plan between waypoints. This forces the use of the same motion equations and collision checker as the model that is being checked in.

[3]Note: For our experiments we cut the z-dimension at a particular low height. The overhangs are not seen in the planner for the [x,y] model in Figure 3 during step (a). An alternative is to project the z-dimension into 2D. Projection is a more conservative approach since the robot would never consider going under the overhangs in [x,y]. In both cases, there is information lost to models that do not consider the z-dimension.



(a) x, y (2d environment, planar motions) (b) x, y, z, θ (3d environment, constrained s-curve motions) (c) $x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}$ (2d environment + time, smoother unconstrained s-curve motions)

Figure 5: Examples of plan generation for three different models.

Table 1: Models Used

Model label	States	Control Inputs	Action Space
[x, y]	x and y	none	straight line motions and interpolation.
[x, y, z]	x, y, and z	none	straight line motions and interpolation.
[x, y, θ]	x, y, and theta	sample linear velocity, u[0], rotational velocity u[1] found by dividing by radius.	Equations of motion. $\dot{x} = u[0]\cos(\theta)$ $\dot{y} = u[0]\sin(\theta)$ $\dot{\theta} = u[1]$
[x, y, z, θ]	x, y, z, and theta	sample linear velocity, u[0], rotational velocity u[1] found by dividing by radius.	Equations of motion. $\dot{x} = u[0]\cos(\theta)$ $\dot{y} = u[0]\sin(\theta)$ $\dot{\theta} = u[1]$
[x, y, θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]	x, y, theta, and time	sample left and right wheel velocities u[0], u[1].	Equations of motion. $\dot{x} = \frac{(u[0]+u[1])}{2\cos(\theta)}$ $\dot{y} = \frac{(u[0]+u[1])}{2\sin(\theta)}$ $\dot{\theta} = u[1] - u[0]$
[x, y, z, θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]	x, y, z, theta, and time	sample left and right wheel velocities u[0], u[1].	Equations of motion. $\dot{x} = \frac{(u[0]+u[1])}{2\cos(\theta)}$ $\dot{y} = \frac{(u[0]+u[1])}{2\sin(\theta)}$ $\dot{\theta} = u[1] - u[0]$

Since the global plan contains a mix of models, the translation function loops through the global plan translating partial plans between connection points.

4.2 Execution

Robot 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 linear velocity to be a function of the angular error when the angular error is greater than some epsilon.

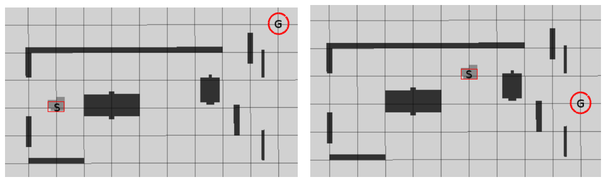
The controller also uses a correction in angular velocity

Table 2: Collision Checking

Model label	States	Collision Checking
[x, y]	x and y	in x and y, with 2D obstacles.
[x, y, z]	x, y, and z	in x, y, and z with 3D obstacles.
[x, y, θ]	x, y, and theta	in x, y, and theta with 2D obstacles.
[x, y, θ]	x, y, z, and theta	in x, y, z, and theta with 3D obstacles.
[x, y, θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]	x, y, theta, and time	in x, y, and theta with 2D obstacles indexed by time.
[x, y, z,θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]	x, y, z, theta, and time	in x, y, z, and theta with 3D obstacles and indexed by time.

for better path following. 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).

Execution Monitor We use a Gazebo contact sensor to simulate the robot's bump sensor. If the robot bumps into anything a failure message is sent to indicate failure during execution. This failure signal can be used to initiate replanning, using a variant of Algorithm 1.



(a) Test 1 start and end (b) Test 2 start and end

Figure 6: The start and end positions for the simulation runs.

5 Experimental Results

We ran experiments in the simulator world for two different start and end states as shown for Test1 and Test2 in Figure 6. Test1 requires the robot to traverse through more obstacles, take more turns, and navigate more overhangs than Test2. Test2 focuses on the dynamic sliding door. It places the robot closer to the doors and changes the goal to be directly behind the doors.

Table 3: Translation

Model label	Translate to Label	Translation
[x,y]	[x,y,z]	add z
[x,y]	[x,y,θ]	add theta by finding arctan from next waypoint and get linear and angular velocity from distance to next waypoint.
[x,y,z]	[x,y,z,θ]	add theta by finding arctan from next waypoint and get linear and angular velocity from distance to next waypoint.
[x,y,θ]	[x,y,z,θ]	add z.
[x,y,θ]	[x,y,θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]	convert linear and angular velocity to left and right wheel velocities. Time variable calculated through state propagation.
[x,y,z,θ]	[x,y,z,θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]	convert linear and angular velocity to left and right wheel velocities. Time variable calculated through state propagation.
[x,y,θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]	[x,y,z,θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]	add z.

We ran three test types. First we ran all six models independently without allowing switching, as shown in Table 4. Then we demonstrate our approach using Algorithm1 and always starting in the lowest model. We refer to these results as 'switch all' shown in Table 5. Lastly, for comparison, we ran tests that only switch between the lowest and highest model, again always starting in the lowest model.

For all tests we recorded the planing time mean and std deviation for 100 runs. We also display the percentage of successful executions that occurred for running the final produced plan in simulation, without execution time replanning.

To produce good plans, we generated 30 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 fidelity models to show the efficacy of our approach. In normal practice we would expect the initial model (in this case [x,y]) to fail rarely.

Table 4 shows the percentage of successful executions to the goal for each model (without switching) as well as their average planning times. We see that as the models increase in fidelity the mean plan-time over 30 runs increases. The time change in adding the z-component is less than that of theta since this is purely a geometric change which has collision checking in three-dimensions rather than two. The search space does not change. For models that incorporate theta, collision checking is done using theta. Additionally, theta is sampled and the linear velocity is sampled to constrain motion to s-curves. This increases the state search space for waypoint targets and the action space. The two highest fidelity models take the longest since they sample both the right and left wheel velocities as well as incorporating time in the state which adds additional collision checks for sliding doors. This also creates a larger search space which increases the branching factor causing higher planning times.

Test1 starts in front of a long overhang. Models that include the z-dimension are the most beneficial for this test. Additionally, since this test ends near the sliding door velocity models played a minor role in task success.

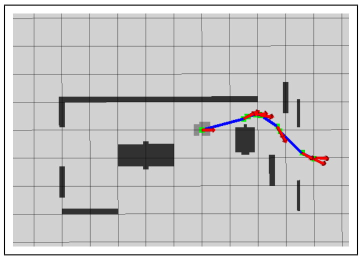


Figure 7: An example of a plan generated in model $[x, y, z, \theta, \dot{x}, \dot{y}, \dot{\theta}]$ for Test2.

In Test2, success rates are much higher for models with velocity since they do properly model the time space (note that models that do not consider time assume the sliding doors are always open). Figure 7, is an example of a path generated in the model [x,y,z,θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]. Note that the velocity decreases as the robot nears the doors (as evidenced by the smaller distance between waypoints), waiting for the doors to open, and then increases again to rush through the doors before they close.

Table 4: Results for single model runs with no switching.

Model	success %	plan-time mean(s)	plan-time std dev
Test 1 (From x=-2, y=-2 to x=6.0, y=2.0). Includes overhangs.			
[x,y]	19%	0.13	0.01
[x,y,θ]	20%	79.0	25.5
[x,y,z]	67%	1.79	0.52
[x,y,z,θ]	87%	93.0	19.1
[x,y,θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]	17%	110.2	30.2
[x,y,z,θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]	90%	158.5	43.3
Test 2 (From x=2.0 , y=-1.0 to x=6.0 to y=-2.0). Includes moving door.			
[x,y]	35%	0.10	0.01
[x,y,θ]	46%	56.1	14.6
[x,y,z]	44%	1.22	0.33
[x,y,z,θ]	52%	81.1	19.5
[x,y,θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]	73%	162.2	51.1
[x,y,z,θ, $\dot{x}$, $\dot{y}$, $\dot{\theta}$]	75%	194.2	58.2

Table 5: Switching results.

Model	success %	plan-time mean(s)	plan-time std dev
Test 1 (From x=-2, y=-2 to x=6.0, y=2.0). Includes overhangs.			
Switch all	90%	2.44	8.44
Switch lowest and highest	86%	17.4	23.0
Test 2 (From x=2.0 , y=-1.0 to x=6.0 to y=-2.0). Includes moving door.			
Switch all	85%	128.2	127.7
Switch lowest and highest	80%	149.4	146.3

Table 5 shows the results of our switching algorithm. Note that in these results mean planning time also includes the time it took for selecting the next appropriate model to replan in. However, model selection times are negligible. Our algorithm, 'switch all', produces success rates comparable to that of the highest fidelity model, but with much more efficient planning times. For comparison, we also evaluated a more traditional approach where the system switches between only two models (the lowest and highest fidelity models). The traditional approach produces success rates comparable to our full switching approach, but the plan times are higher.

The plan-time gains for switching were higher for Test1 than Test2. This is due to which models were selected during plan-time to switch to. The models selected for switching with Test2 were most often the velocity models since infeasibility was often detected in the time dimension. This causes the times to be higher on average. Additionally, plans with impediments detected directly in front of the sliding doors tend to take much longer to repair since the robot cannot travel backwards. Test1 switched more often to lower models with the 'z' dimension which gave a larger plan-time savings. This is most evident when compared with the switching between only the highest and lowest model. In many parts of the Test1 space the highest model is not necessary for task success.

In general, switching creates plans with mean times lower than the highest fidelity model and higher than the lowest fidelity model. Intermediate goals further improve planning times. This is evident in the results for switching between the low and high model in Test1. The mean planning time is much lower because the highest model is only necessary for a short detour in the space. Switching with all models created plans with mean times lower than just switching between the low and high models without sacrificing execution success. The switching tests also contained a handful of cases which failed to generate a plan. This suggests additional evaluations for completeness and guarantees.

The fact that the robot still fails in our highest model during execution 20-25% of the time means the framework can still be extended to additional models. For instance, assuming instantaneous velocity and acceleration caused the robot controller to vary from the plan. A model that can capture this would reduce the failure rates even more. In cases where it is difficult to model the information necessary to approximate the underlying controller, it would also be possible to add uncertainty.

6 Future Work

The architecture allows for expansion to additional models. The model graph can include other robot subsystems such as models for a manipulator, or those that combine manipulator and chassis motions. We would also like to include models with simple physics such as friction for object interaction, and forces for pushing. The architecture allows models that are intractable over the full plan generation to still be used locally which would increase robot robustness by enabling even higher success rates.

We observed that if the robot controller did not follow the generated plan exactly time state synchronization drifts. If this occurred when the robot was passing near the sliding doors, during a state change, it most certainly caused a failure. At other times it had little effect. The ability to add uncertainty as a buffer around time, would allow the robot to be more conservative when deciding to pass the doors. Therefore, we would like to investigate how uncertainty fits into the model hierarchy and selection process. We recognize increasing uncertainty, such as a very large robot footprint, can cause the robot to be conservative and think it cannot traverse through a narrow area. This is why we would also like to investigate different levels of uncertainty coverage. We believe varying uncertainty is another form of varying fidelity and will give the robot more choices to increase success when applicable.

Lastly, we would like to combine the execution-time version of the approach with our plan-time approach for real-robot application. Execution time is important for obtaining information that is not available during plan-time. This could be due to uncertainty in the world or sensing abilities. An execution time approach would focus the algorithm towards the failure recovery domain. Information acquired during execution is then provided to the planner allowing the robot to switch to models it previously did not have enough information for. This would continue to provide a more robust plan.

7 Conclusions

We present an approach that 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 generating an efficient plan that does not sacrifice execution success.

Our tests show that the algorithm improves computation time while obtaining comparable performance to planning always in the highest fidelity model. The average overall planning time (initial plus re-plans) is greater than the average time for [x,y] planning alone, but still less than the average for the higher fidelity spaces. Switching between multiple models is also cheaper than just switching between the lowest and highest model. Our switching tests localize planning around the infeasible location without sacrificing the probability of successful executions.

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). This is why it is important to have a separate model selection stage that can reason about which model should be used for repair. This is increasingly important as robots, their tasks, and their environments become more complex. We believe that, especially as these complexities increase, selecting the most appropriate model to plan in is important for robust, tractable planning.

References

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

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.

Ferguson, D.; Kalra, N.; and Stentz, A. 2006. Replanning with rrts. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, 1243–1248. IEEE.

Fernández-Madrigal, J.-A., and González, J. 2002. Multihierarchical graph search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24(1):103–113.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial intelligence* 3:251–288.

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.

Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 1470–1477. IEEE.

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.

Koenig, S., and Likhachev, M. 2002. D* lite. In *AAAI/IAAI*, 476–483.

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

LaValle, S. M. 2006. *Planning algorithms*. Cambridge university press.

Pivtoraiko, M., and Kelly, A. 2005. Efficient constrained path planning via search in state lattices. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*.

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.

Raibert, M.; Blankespoor, K.; Nelson, G.; Playter, R.; et al. 2008. Bigdog, the rough-terrain quadruped robot. In *Proceedings of the 17th World Congress*, volume 17, 10822–10825.

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

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*.

Stentz, A. 1995. The focussed dˆ* algorithm for real-time replanning. In *IJCAI*, volume 95, 1652–1659.

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.

Şucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 19(4):72–82. http://ompl.kavrakilab.org.

Wolfe, J.; Marthi, B.; and Russell, S. J. 2010. Combined task and motion planning for mobile manipulation. In *ICAPS*, 254–258.