Maxim Likhachev
Computer and Information Science,
University of Pennsylvania,
Philadelphia, PA 19104, USA
maximl@seas.upenn.edu

Dave Ferguson
Two Sigma Investments
379 West Broadway
New York, NY 10012, USA
dave.ian.ferguson@gmail.com

Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles

Abstract

In this paper, we present an algorithm for generating complex dynamically feasible maneuvers for autonomous vehicles traveling at high speeds over large distances. Our approach is based on performing anytime incremental search on a multi-resolution, dynamically feasible lattice state space. The resulting planner provides real-time performance and guarantees on and control of the suboptimality of its solution. We provide theoretical properties and experimental results from an implementation on an autonomous passenger vehicle that competed in, and won, the Urban Challenge competition.

1. Introduction

Autonomous vehicles navigating through cluttered, unstructured environments or parking in parking lots often need to perform complex maneuvers and reason over large distances. Furthermore, this reasoning usually needs to be performed very quickly so that the resulting maneuvers can be executed in a timely manner, particularly if the environment is inhabited, dynamic or dangerous. In particular, our current focus is planning for autonomous urban driving including both off-road scenarios and large unstructured parking lots such as the ones in front of malls and large stores (of the order of 200 m × 200 m). Maneuvering at human driving speeds ($\backsim$ 15 mph) through such areas requires very efficient planning, especially if they contain static obstacles or other moving vehicles.

Roboticists have concentrated on the problem of mobile robot navigation for several decades, providing a large body of research. Early approaches concentrated on *local* planning, where very short-term reasoning is performed to generate the next action for the vehicle. These include potential field-based techniques, where obstacles exert repulsive forces on the vehicle while the goal exerts an attractive force (Khatib 1986), and the curvature velocity (Simmons 1996) and dynamic window (Fox et al. 1997) approaches, where planning is performed in control space to generate dynamically feasible actions. One major limitation of these purely local approaches was their capacity to get the vehicle stuck in local minima en route to the goal (for instance, cul-de-sacs). Further, these approaches are unable to perform complex multi-stage maneuvers, such as three-point turns, as these maneuvers are not within the set of local actions considered by the planner.

To reduce the susceptibility to local minima of these approaches, algorithms were developed that incorporated *global* as well as local information (Kelly 1995; Thrun and others 1998; Brock and Khatib 1999; Philippsen and Siegwart 2003). Typically, these approaches generate a set of candidate simple local actions and evaluate each based on both their local traversability cost and the desirability of their endpoints based on a global value function (e.g. the expected distance to the goal based on known obstacle information). Although these approaches perform better with respect to local minima, their simple local planning can still cause the vehicle to get stuck or take highly suboptimal paths. Subsequent approaches have focused on improving this local planning by using more sophisticated local action sets that better follow the global value function (Thrun et al. 2006; Howard and Kelly 2007), and by generating sequences of actions to perform more complex lo-

The International Journal of Robotics Research
Vol. 28, No. 8, August 2009, pp. 933–945
DOI: 10.1177/0278364909340445

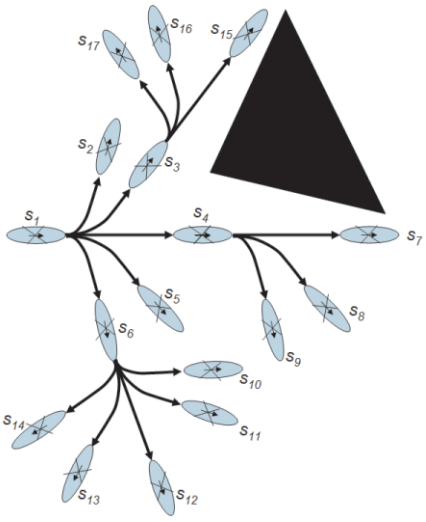
Figures 1, 3–12 appear in color online: http://ijr.sagepub.com

cal maneuvers (Urmson et al. 2006; Braid et al. 2006). The most complex of these approaches are able to perform very precise local maneuvering but are limited by the mismatch between their powerful local planning and their approximate global planning, resulting once more in a susceptibility to local minima.

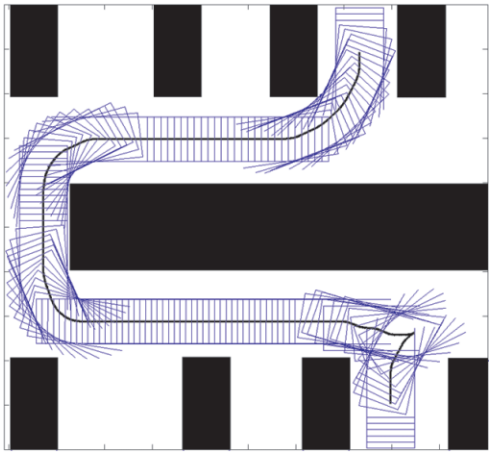
Recognizing this mismatch, other approaches have concentrated on improving the quality of the global planning, so that a global path can be easily tracked by the vehicle. In particular, some of the earlier approaches to global planning for nonholonomic vehicles are closely related to our work as they were based on constructing and searching graphs that contained dynamically feasible paths (Barraquand and Latombe 1993). Later, a number of approaches were proposed based on the idea of randomized motion planning (Svestka and Overmars 1997; LaValle and Kuffner 2001). Other methods were based on the idea of globally planning feasible paths within the vicinity of the paths generated by a geometric (i.e. 2D) planner (Laumond et al. 1994; Lamiraux et al. 1999; Stachniss and Burgard 2002). More recently, a number of methods have also been proposed based on systematic discretization of the environment and applying efficient graph searches or using highly informative heuristics to guide the search for feasible paths (Likhachev et al. 2003, 2005; Knepper and Kelly 2006). However, the computational expense of generating complex feasible plans over large distances has remained challenging, and current approaches are restricted to either small distances, fairly simple environments or highly suboptimal solutions.

In this paper, we present an efficient, global planning approach that attempts to overcome these challenges. First, we employ a multi-resolution lattice search space to reduce the complexity of the global search while still providing extremely high-quality solutions. Second, we use an efficient anytime, incremental search to quickly generate bounded suboptimal solutions, then improve these solutions while deliberation time allows and repair them when new information is received. The resulting approach is able to plan complex, dynamically feasible maneuvers over hundreds of meters and improve and repair them in real-time for vehicles traveling at high ($\backsim$ 15 mph) speeds.

This paper is an extended version of Likhachev and Ferguson (2008). The additional material we give in this paper includes the details of the most important optimizations used in the implementation of our approach, as well as additional experimental results. The paper is organized as follows. We first describe how the multi-resolution lattice is constructed, and then the search we use to plan paths in the lattice and the theoretical properties it can provide. We next describe some of the optimizations used in our implementation of the approach, and conclude with experimental results from both simulation and the Urban Challenge competition.



(a)

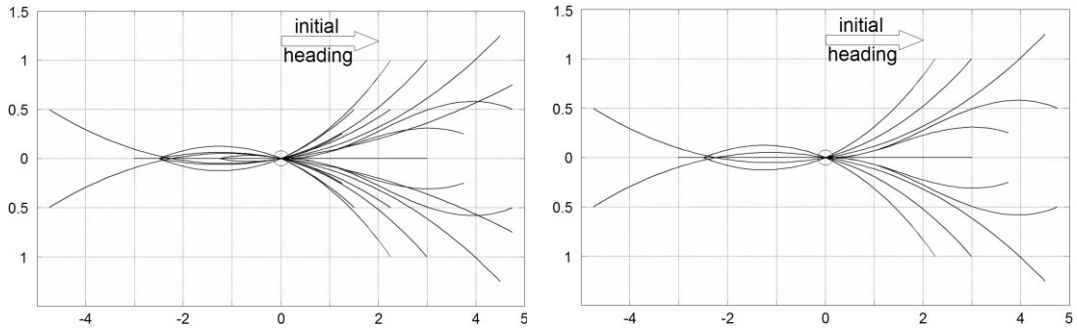


(b)

Fig. 1. (a) A 3D (x, y, θ) lattice with at most five forward actions for each state and no backward actions. A full set of actions is shown for state s_1. For every state, this set of actions is translated and rotated appropriately, and all actions intersecting obstacles are removed. (b) An example of a complex 3D path for a vehicle with a large rectangular footprint (obstacles are shown in black).

2. Multi-resolution Lattice State Space

A state lattice (Pivtoraiko and Kelly 2005) is a discretization of the configuration space into a set of states, representing configurations and connections between these states, where every connection represents a feasible path (see Figure 1 for an example of a lattice). As such, lattices provide a method for motion planning problems to be formulated as graph searches. However, in contrast to many graph-based representations (such as 4-connected or 8-connected grids), the



(a) (b)

Fig. 2. (a) High- and (b) low-resolution action spaces.

feasibility requirement of lattice connections guarantees that any solutions found using a lattice will also be feasible. This makes them very well suited to planning for non-holonomic and highly constrained robotic systems, such as passenger vehicles.

2.1. State Space

The two key considerations in constructing a lattice are the discretization (or sampling) strategy used for representing the states in the lattice and the action space (or control set) used for the inter-state connections. For our application we employ a four-dimensional (x, y, θ, v) state representation, where (x, y) represent the position of the center of the vehicle in the world, θ represents the orientation of the vehicle and v represents its translational velocity. The (x, y, θ) coordinates are important for computing the validity of the poses of the vehicle in the world and making sure that no path in the lattice requires an instantaneous change in the orientation of the vehicle. For the velocity v we use two possible values: maximum forward velocity and maximum reverse velocity. We take velocity into account because the time involved in switching between forward and backward directions is substantial so reasoning about this cost is important for generating fast, smooth paths[1].

2.2. Action Space

The action space for each state in the lattice is intended to be dense enough that every possible feasible path through the lattice can be constructed by combining sequences of these actions. However, because this action space represents the branching factor of the subsequent graph search, in practice it must be carefully constructed to provide flexibility in path selection while maintaining computational tractability.

The offline construction of our action space is based on work by Pivtoraiko and Kelly (2005) that attempts to create near-minimal spanning action spaces. Given a state s, we compute the action space by first calculating a subset of states within a distance d of s that is reachable via some feasible action. To generate the feasible actions we use a trajectory generation algorithm originally developed by Howard and Kelly (2007). This algorithm employs an accurate vehicle model to produce feasible, directly executable actions and an optimization technique to minimize the endpoint error of these actions with respect to a desired endpoint state. We use this approach to "snap" the actions to the lattice so that the endpoint of each action lands on a lattice state. Next, we look at this set of actions and calculate whether any single action can be approximately recomposed out of a combination of other, shorter actions. If so, these longer actions are discarded from our set. This provides us with a compact set of actions that approximate the full reachable space. However, in contrast to the approach in Pivtoraiko and Kelly (2005), we maintain multiple straight segments of varying lengths to improve the speed of the subsequent search, as we will discuss in the section on Anytime Dynamic A*. Figure 2(a) illustrates the action space for a single state (oriented to the right) in our lattice.

2.3. Multi-resolution Lattice

Even with a compact action space, planning long complex maneuvers over lattices can be expensive in terms of both computation and memory. An important observation, however, is that usually there exists a wide spectrum of smooth, dynamically feasible paths between the vehicle and goal configurations and it is a waste of time and memory to explore all of them. On the other hand, all of these paths start and end at the exact same configurations, and the challenge is in finding a path that satisfies the current vehicle configuration and the specific goal configuration precisely.

1. We do not reason about curvature (the orientation of wheels) because we found this to be less critical for the speeds we are interested in traveling at, as discussed in the results section.

This motivated us to take a novel, multi-resolution approach, where we use a high-resolution action space in the vicinity of the robot and the goal, and a low-resolution action space elsewhere. We call the resulting combination a multi-resolution lattice. With this approach, we can harness most of the benefit of the high-resolution representation without paying anything near the full computational cost. The trick is making sure that the high-resolution and low-resolution lattices connect together smoothly.

Our multi-resolution approach maintains the same dimensionality (x, y, θ, v) for both resolutions, but the action space for the low-resolution lattice is a strict subset of the action space for the high-resolution lattice. Figure 2(a) shows the action space used in the high-resolution lattice and Figure 2(b) shows the action space used in the low-resolution lattice[2]. Using this method ensures that the low-resolution lattice is utilized fully and that paths in the multi-resolution lattice are guaranteed to be feasible, which is a strong advantage over existing combined local and global approaches for navigation.

Theorem 1. *Every path in a lattice that uses only a low-resolution action space is also a valid path in our multi-resolution lattice. Further, every path in the multi-resolution lattice is a valid path in a lattice that uses only the high-resolution action space.*

Proof. The proof of the first claim follows trivially from the fact that any action in the low-resolution lattice is a valid action in both the low-resolution and high-resolution lattices, and therefore is a valid action in the multi-resolution lattice. A similar argument applies for the second claim. ■

Enforcing the low-resolution action space to be a subset of the high-resolution action space decreases the *branching factor* of the graph constructed by the search, which is certainly important, but it does not necessarily decrease the *size* of the graph. However, it is also possible to decrease the size of the graph as follows. Suppose A_h is an action space used in the high-resolution space and A_l is an action space used in the low-resolution space. Thus, $A_l \subset A_h$. Then, we can construct A_l by picking only the actions from A_h that end at states with a coarser discretization than the end states of actions in A_h. For example, we can choose for A_l only those actions whose end states have θ equal to one of 16 possible angles, while actions in A_h can connect states with 32 possible values of θ. (This is precisely what we used in our system.) Mathematically, the construction of the action space A_l can be expressed as follows in terms of a high-resolution discretization Q_h and a lower-resolution discretization Q_l of variables x, y, θ, v: an action a connecting states $s_1 = (x_1, y_1, \theta_1, v_1)$ and $s_2 = (x_2, y_2, \theta_2, v_2)$ belongs to A_l if and only if $a \in A_h$ and $(x_2, y_2, \theta_2, v_2) \in Q_l$.

Restricting Q_l to a coarser discretization for (x, y) or θ corresponds to using a discretization that adapts based on the vehicle and goal configurations. This technique can also be used to explicitly constrain the behavior of the vehicle in the different areas. For instance, restricting Q_l to contain only positive v-values prevents the vehicle from moving backward when far from the initial and goal configurations. This general approach allows for an arbitrarily reduced state and action space in the low-resolution portion of the lattice, and can also be trivially extended to more than two levels of resolution if desired.

3. Anytime, Incremental Search

Given a search space (in our case, in the form of a multi-resolution lattice) and a cost function associated with each action, we need an efficient method for searching through this space for a solution path. A* search is perhaps one of the most popular methods for doing this (Nilsson 1980). It utilizes a heuristic to focus the search towards the most promising areas of the search space. While highly efficient, A* aims to find an optimal path which may not be feasible given time constraints and the size of environments autonomous vehicles need to operate in. To cope with very limited deliberation time, anytime variants of A* search have been developed (Zhou and Hansen 2002; Likhachev et al. 2003). These algorithms generate an initial, possibly highly suboptimal solution very quickly and then concentrate on improving this solution while deliberation time allows. Furthermore, these anytime algorithms are able to provide bounds on the suboptimality of the solution at any point of time during the search.

A* and its anytime variants work best when the search space, and thus environment, is mostly known *a priori*. In robotic path planning this is rarely the case, and the robot typically receives updated environmental information through onboard and/or offboard sensors during execution. To cope with imperfect initial information and dynamic environments, efficient incremental variants of A* search have been developed that update previous solutions based on new information (e.g. from sensors) (Barbehenn and Hutchinson 1995; Stentz 1995; Koenig and Likhachev 2002). These algorithms repair existing solutions for a fraction of the computation required to generate such solutions from scratch.

When faced with limited deliberation time *and* imperfectly-known or dynamic environments, it is extremely useful to have a search algorithm that is both anytime and incremental. The Anytime Dynamic A* algorithm developed by Likhachev et al. is a version of A* search that combines these two properties into a single approach and has been shown to be very effective for a range of robotic planning tasks (Likhachev et al. 2005). We employ this algorithm for planning and replanning paths in our multi-resolution lattice.

2. In practice, choosing the appropriate set can be achieved with a basic check: if the (x, y) location of a state is not within some distance d of the vehicle or goal, its action set is the low-resolution set.

*3.1. Anytime Dynamic A**

Anytime Dynamic A* (AD*) exploits a property of A* that can result in much faster generation of solutions, namely that if consistent heuristics are used and multiplied by an inflation factor $\epsilon > 1$, then A* can often generate a solution much faster than if no inflation factor is used (Gaschnig 1979), and the cost of the solution generated by A* will be at most ϵ times the cost of an optimal solution (Davis et al. 1988). AD* operates by performing a series of these inflated A* searches with decreasing inflation factors, where each search reuses information from previous searches. By doing so, it is able to provide suboptimality bounds on all solutions generated and allow for control of these bounds, since the user can decide how much the inflation factor is decreased between searches. To cope with updated information, AD* also borrows ideas from the D* and D* Lite algorithms (Stentz 1995; Koenig and Likhachev 2002) and only propagates updated information through the affected and relevant (given the current search) portions of the search space.

To enable efficient anytime planning and replanning as the vehicle moves, we use AD* to search backwards from the goal configuration towards the current configuration of the vehicle. The heuristic used thus needs to estimate the cost of a shortest path from the vehicle configuration (rather than goal) to each state in question.

The effectiveness of Anytime Dynamic A* is highly dependent on its use of an informed heuristic to focus its search. An accurate heuristic can reduce the time and memory required to generate a solution by orders of magnitude, while a poor heuristic can diminish the benefits of the algorithm. It is thus important to devote careful consideration to the heuristic used for a given search space. Further, because we are inflating heuristic values, it is useful to have long actions that can skip over several nodes and reduce the number of states in the search. It is for this reason we add several straight line actions of varying length in both the forwards and backwards directions to our action set (as was mentioned previously).

The control of the operation of AD* happens via the parameter ϵ. Once the planner receives a new goal, it first sets ϵ to a sufficiently high value (e.g. 2.0–3.0 in our experiments), so that AD* can generate an initial, possibly quite suboptimal, solution quickly. (It is important to note that AD* never generates the whole multi-resolution lattice. Instead, it constructs the lattice as it expands states during the search in order to minimize the memory requirements and the computations required to evaluate the transitions in the lattice.) If planning time remains, then AD* decreases ϵ and recomputes the solution to satisfy the new bound. If not, then the solution is passed on to the vehicle for execution. While the vehicle moves, AD* continues to decrease ϵ and recompute the trajectory for the bound. The trajectories are being recomputed always with respect to the current position of the vehicle. When ϵ reaches 1, the solution is guaranteed to be optimal. Whenever map updates are received, AD* processes the updates to the corresponding edge costs of the states that were generated (the states that have not been generated so far do not need to be updated). At any point of time, if map updates are significant and some of them reside on the currently planned trajectory, then ϵ is set back to its initial value. This ensures that the planner regenerates the new path quickly. Otherwise, ϵ is decreased, and AD* recomputes the trajectory that satisfies the new suboptimality bound and at the same time is correct with respect to the updated edge costs. More algorithmic details about AD* can be found in Likhachev et al. (2005).

3.2. Informative Heuristics

The purpose of a heuristic is to improve the efficiency of the search by guiding it in promising directions. A common approach for constructing a heuristic is to use the results from a simplified search problem (e.g. from a lower-dimensional search problem where some of the original constraints have been relaxed). In selecting appropriate heuristics, it is important to analyze the original search problem and determine the key factors contributing to its complexity. In robotic path planning these are typically the complexity inherited from the constraints of the mechanism and the complexity inherited from the nature of the environment.

To cope with the complexity inherited from the mechanism constraints, a very useful general heuristic is the cost of an optimal solution through the search space assuming a completely empty environment. This can be computed offline and stored as a heuristic lookup table, and several efficiencies can be used to reduce the required memory for this table (Knepper and Kelly 2006). This is a very well-informed heuristic function when operating in sparse environments and is guaranteed to be an optimistic (or admissible) approximation of the actual path cost. In obstacle-laden environments, however, this heuristic function can grossly underestimate the true costs and mislead the search into the exploration of wrong regions of the state space.

To cope with the complexity inherited from the nature of the environment, it is not practical to precompute heuristic values for all possible environment configurations, as there are an effectively infinite number of possibilities for any reasonably sized environment. However, in this case it is beneficial to solve online a simplified search problem given the actual environment and use the result of this search as a heuristic to guide the original, complex search. In particular, we solve a 2D $((x, y))$ version of the problem by running a single Dijkstra's search starting at the cell that corresponds to the center of the current vehicle position. The search computes the costs of shortest paths from the cell the robot is in to all other cells in the environment. This search is therefore rerun every time the vehicle pose is changed. Even though the search is very fast, we still restrict this search to only compute the states that

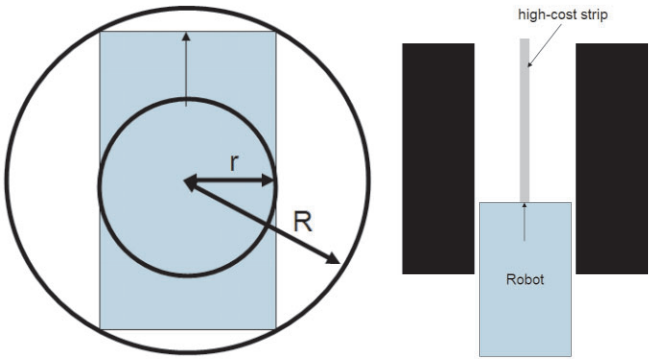


Fig. 3. (a) Inner (r) and outer (R) radii of the robot. (b) Example where the 2D heuristic function may overestimate the cost of a path derived purely from convolution.

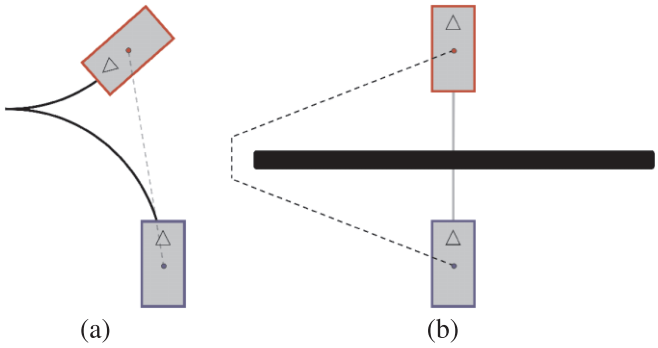


Fig. 4. Mechanism-constrained (solid) and environment-constrained (dashed) heuristic paths. In each case, the initial and desired vehicle poses are shown as blue and red rectangles, respectively (with the interior triangles specifying the headings). (a) The mechanism-constrained heuristic is perfectly informed when no obstacles are present in the environment. (b) The environment-constrained 2D heuristic can provide significant benefit when obstacles exist. Here, an obstacle (shown in black) resides over the direct path to the desired pose.

are no more than twice as far (in terms of path cost) from the vehicle cell as the goal cell. In other words, the search is terminated whenever the cost of the cell it is about to compute is larger than or equal to the cost of the goal cell. To make the heuristic function more informative, the cost of each cell in the 2D grid used for computing this 2D heuristic is set to infinity whenever the distance from it to the closest obstacle is less than the inner radius of the robot (shown in Figure 3(a)). Distances from cells to obstacles can be efficiently computed using a distance transform (see the section on optimizations).

AD* requires the heuristics to be admissible and consistent. This holds if $h(s_{start}) = 0$ and for every pair of states s, s' such that s' is an end state of a single action executed at state s, $h(s)+c(s,s') \geq h(s')$, where $h(s)$ is a heuristic of state s, s_{start} is a state that corresponds to the vehicle configuration and $c(s,s')$ is the cost of the action that connects s to s'. The cost $c(s,s')$ of the action is typically computed as the length of the action times the average of the costs of the cells covered by the vehicle when moving from state s to state s'. The heuristic based on the 2D search, however, may overestimate these costs since it estimates the cost of moving the center of the vehicle only. To demonstrate this, imagine a path that involves the vehicle moving through a narrow corridor with a high-cost strip going exactly along the center of this corridor (Figure 3(b)). The cost of the 2D path from the initial coordinates of the vehicle to the goal coordinates corresponds to the summation of the costs of the transitions going along the high-cost strip. Cells on either side of the strip are impassable (have infinite cost) since they lie closer to the obstacles than the inner radius of the vehicle, and therefore the center of the vehicle can not reside in any of them. The cost of the actual path, on the other hand, is lower than the cost of the path along the high-cost strip because the cost of each actual action is computed as an average of the cells covered by the vehicle.

To resolve this inadmissibility of the heuristic function, the cost of each transition $c(s,s')$ is computed as the length of the transition times the maximum of two quantities: (a) the average value of the costs of the cells covered by the vehicle when moving from state s to state s' (same as before), and (b) the maximum of the 2D grid cell costs, used to compute heuristics, traversed through by the center of the vehicle when moving from s to s'. Intuitively, this cost function penalizes slightly more those actions for which the vehicle traverses high-cost areas (e.g. obstacles) that reside right under its center. In addition, the heuristics are scaled down by a factor of 1.08 to compensate for the suboptimality of optimal paths in 8-connected grids. It can be then shown that our 2D heuristic function is admissible and consistent with respect to this cost function.

Each of these heuristic functions, mechanism-relative and environment-relative, have strong and complementary benefits (see Figure 4). Rather than selecting one, it is possible to combine the two. We do this by constructing a new heuristic that, for each state s, returns the value $h(s) = \max(h_{fsh}(s), h_{2D}(s))$, where $h_{fsh}(s)$ is the heuristic value of state s according to the mechanism-constrained heuristic (freespace heuristic) and $h_{2D}(s)$ is the value according to the environment-constrained heuristic (2D heuristic). As shown in the experimental results, this combined heuristic function can be an order of magnitude more effective than either of the component heuristic functions. Since both $h_{fsh}(s)$ and $h_{2D}(s)$ are admissible and consistent, the combined heuristic is also admissible and consistent (Pearl 1984). This property satisfies the conditions required by AD* to guarantee ϵ-bound

on the suboptimality of the paths it returns (see Theorem 1 in Likhachev et al. (2005)).

Theorem 2. *The cost of a path returned by Anytime Dynamic A* is no more than ϵ times the cost of a least-cost path from the vehicle configuration to the goal configuration using actions in the multi-resolution lattice, where ϵ is the current value by which Anytime Dynamic A* inflates heuristics.*

4. Optimizations

4.1. Efficient Convolution

Typically, one of the most computationally expensive parts of planning for vehicles is computing the cost of actions, as this involves convolving the geometric footprint of the vehicle for a given action with a map from perception. In our application, we used a 0.25 m resolution 2D perception map and the (x, y) dimensions of our vehicle were 5.5 m × 2.25 m. Thus, even a short 1 m action requires collision checking of roughly 300 cells. Further, the specific cells need to be calculated based on the action and the initial pose of the vehicle.

To reduce the processing required for this convolution, we performed two optimization steps. First, for each action a we precomputed the cells covered by the vehicle when executing this action. During online planning, these cells are quickly extracted and translated to the appropriate position when needed. Second, we generated two configuration space maps to be used by the planner to avoid performing convolutions. The first of these maps expanded all obstacles in the perception map by the inner radius (see Figure 3(a)) of the robot; this map corresponded to an optimistic approximation of the actual configuration space. Given a specific action a, if any of the cells through which the center of the robot executing action a passes are obstacles in this inner map, then a is guaranteed to collide with an obstacle. The second map expanded all obstacles in the perception map by the outer radius (see Figure 3(a)) of the robot and therefore corresponded to a pessimistic approximation of the configuration space. If all of the cells through which the center of the vehicle passes when executing action a are obstacle-free in this map, then a is guaranteed to be collision-free. Only those actions that do not produce a conclusive result from these simple tests need to be convolved with the perception map. Typically, this is a severely reduced percentage, thus saving considerable computation. To create these auxiliary maps efficiently, we performed a single distance transform on the perception map and then thresholded the distances using the corresponding radii of the robot for each map.

4.2. Efficient Map Updates for Incremental Planning

With incremental planning algorithms such as AD*, when changes are observed in the cost map, they must be propagated through the relevant portions of the search space. However, detecting which actions and states in the search space are directly affected by these changes in the cost map can be expensive. For example, if the cost of the cell (x_c, y_c) changes, then the costs of all actions that involve vehicle going over that cell may change. Typically, there could be thousands of such actions. AD* needs to iterate and update the values of all the states ((x, y, θ, v) poses) from which these actions can be executed. Given the large number of affected actions, this iteration can be very expensive. However, AD* needs to update the values of only those states that have actually been computed in the previous planning iterations. If a state has not been computed, then there is no need to update its value. We exploit this property to decrease the amount of computational effort involved in iterating over the states that may possibly be affected by changes in the cost map. In the following we explain this optimization.

First, we precompute offline all the states that have actions whose costs depend on the cost of the cell $(0, 0)$. These states are grouped into mutually disjoint sets, where each ith set $\Re^{x_i \ldots x_i+d, y_i \ldots y_i+d}$ contains all those states (x, y, θ, v), whose $x_i \leq x < x_i + d$ and $y_i \leq y < y_i + d$, where d is a (small) positive integer. We used $d = 5$. In other words, all the states whose values need to be updated by AD* whenever the cost of the cell $(0, 0)$ is modified are precomputed and stored in a low-resolution grid map. Let us denote this map by $\Re$. Each cell in this low-resolution grid map is d times wider and d times longer than a cell in the perception map.

Second, during online operations, we maintain a low-resolution replanning map of the same discretization as $\Re$. The value of each cell in this replanning map is true whenever at least one state whose (x, y) coordinates fall into this cell has been generated (computed) by AD*. Thus, while planning, whenever AD* generates (computes a value of) a state (x, y, θ, v), then it also sets the corresponding cell in the replanning map to be true.

Finally, whenever the cost of a cell (x_c, y_c) is modified, for each non-empty cell $\Re^{x_i \ldots x_i+d, y_i \ldots y_i+d}$ in $\Re$ we look up if any one of the following four cells in the replanning map are set to true:

$$(((x_i + x_c) \mod d), \quad ((y_i + y_c) \mod d)),$$

$$(((x_i + x_c) \mod d) + 1, \quad ((y_i + y_c) \mod d)),$$

$$(((x_i + x_c) \mod d), \quad ((y_i + y_c) \mod d) + 1),$$

$$(((x_i + x_c) \mod d) + 1, \quad ((y_i + y_c) \mod d) + 1).$$

If so, then we make AD* update the value of every state stored in $\Re^{x_i \ldots x_i+d, y_i \ldots y_i+d}$ translated by (x_c, y_c). No other

states need to be updated since it is guaranteed that they have not been generated by AD* previously. This optimization can save a tremendous amount of replanning effort.

5. Experimental Results

We have implemented our approach on an autonomous passenger vehicle (lower-left image in Figure 9) where it has been used to drive over 3,000 km in urban environments, including competing in the DARPA Urban Challenge. The multi-resolution lattice planner was used for planning through parking lots and into parking spots, as well as for geometric road following in off-road areas, and in error recovery scenarios. During these scenarios, the vehicle traveled at speeds of up to 15 mph while performing complex maneuvers and avoiding static and dynamic obstacles.

The overall architecture of the system is described in Urmson et al. (2008). In brief, whenever the multi-resolution lattice planner was invoked, it was constantly recomputing the path based on the latest map updates received from the perception module. The computed paths were sent down to the local planner, which was responsible for tracking the path. It operated by generating a set of candidate trajectories that followed the path while allowing for some flexibility in local maneuvering. The local planner ran at a fixed 10 Hz during operation. The lattice plan was typically updated once per second. However, in very difficult planning scenarios the lattice planner could take longer (up to a couple seconds) to generate its initial solution. All the modules ran in parallel and communicated via messages.

In all cases, the multi-resolution lattice planner searches backwards out from the goal pose (or set of goal poses) and generates a path consisting of a sequence of feasible high-fidelity maneuvers that are collision-free with respect to the static obstacles observed in the environment. This path is also biased away using cost functions from undesirable areas such as curbs and locations in the vicinity of dynamic obstacles.

When new information concerning the environment is received (for instance, a new static or dynamic obstacle is observed), the planner is able to incrementally repair its existing solution to account for the new information. This repair process is expedited by performing the search in a backwards direction, as in such a scenario updated information in the vicinity of the vehicle affects a smaller portion of the search space and so less repairs are required.

As mentioned earlier, the lattice used in this application does not explicitly represent curvature. Theoretically, this means that the paths produced over this lattice are guaranteed to be feasible only if we allow the vehicle to stop at each lattice state and reorient its steering wheel. However, in practice we reduce (by a small fraction) the maximum curvature used in generating connections between states and we reduce the maximum speed at which we execute higher-curvature sections of lattice paths (from 5 m/s down to 2 m/s) so that this curvature discontinuity is not a critical issue. We also use a lookahead during execution to slow down and stop when switching velocity directions[3]. As a result, we do not need to stop during execution unless the path contains velocity sign changes.

The lattice path is tracked using a local planner that employs the same trajectory generation algorithm used to provide the action space for the lattice. Although a simple, single-trajectory tracker would suffice given the feasibility of the lattice plan, multiple trajectories are produced to account for dynamic obstacles and new observations that could require immediate reaction (the local planner runs at 10 Hz). The trajectories generated by the trajectory generation algorithm are shown as short multi-color trajectories in the top row images in Figure 9. The local planner always chooses the trajectory that stays as close as possible to the path returned by the lattice-based planner while avoiding obstacles.

To ensure that a high-quality path is available for the vehicle as soon as it enters a parking lot, the lattice planner begins planning for the desired goal pose while the vehicle is still approaching the lot. By planning a path from the entry point of the parking lot in advance, the vehicle can seamlessly transition into the lot without needing to stop, even for very large and complex lots. Further, the anytime property of the search enables the solution to be improved during the preplanning stage and, depending on how much time is available for preplanning, the resulting path for the vehicle can converge to a (provably) optimal solution.

As well as providing smooth navigation amongst partially-known static objects, the efficiency of the multi-resolution lattice planner makes it possible to intelligently interact with several dynamic obstacles in the environment. In our application, we were able to not only avoid such obstacles but through updating regions of high cost as the obstacles moved, we could stay well clear of them unless necessary and also exhibit intelligent yielding behavior in unstructured areas (e.g. keeping to the right when approaching oncoming vehicles).

The multi-resolution lattice planner was also used for performing complex maneuvers in error recovery scenarios during on-road driving, such as when a lane or intersection is partially blocked with vehicles or obstacles, or a road is fully blocked and a U-turn is required. It was also used when there was some uncertainty as to where the road was; in these scenarios it uses the geometric perceptual information to bias the vehicle towards the center of the road (when there are perceivable curbs or berms).

We have included here a number of examples from the Urban Challenge and our testing to illustrate key characteristics of the approach.

3. A maximum lookahead of 2 m is required given our vehicle's maximum deceleration and the top speed used for following lattice paths, but we use a slightly higher lookahead for smooth deceleration.

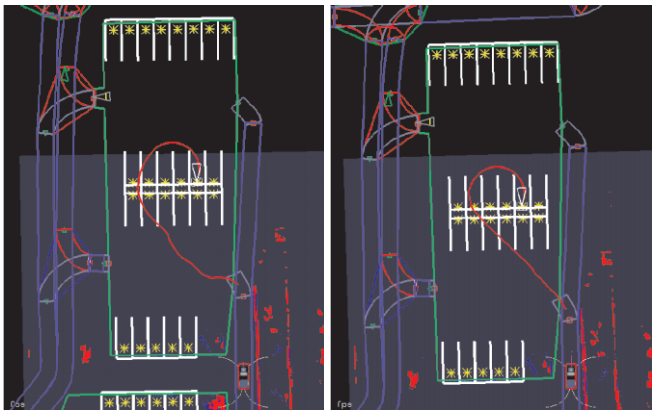


Fig. 5. Preplanning a path into a parking spot and improving this path in an anytime fashion.

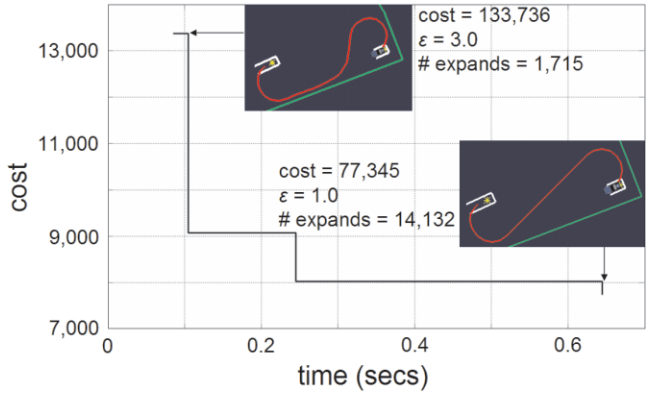
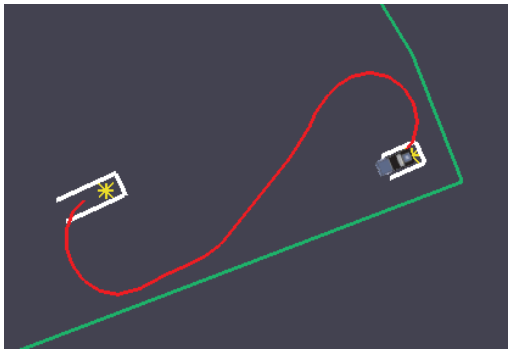


Fig. 6. An example highlighting anytime behavior of our approach.

Preplanning. Figure 5 illustrates the preplanning used by the lattice planner, as well as its anytime performance. The left image shows our vehicle approaching a parking lot (parking lot boundary shown in green, road lanes shown in blue), with its intended parking spot indicated by the white triangle. While the vehicle is still outside the parking lot it begins planning a path from one of the parking lot entries to the desired spot (path shown in red). Although the initial path shown in this left image is feasible, it is not ideal as it involves more turning than necessary. The right image shows how this path is improved over time as the vehicle approaches. This path is optimal with respect to our cost function and is generated well before the vehicle enters the parking lot.

Anytime Planning. Figure 6 illustrates the anytime behavior of the approach when planning between two parking spots. We have included a plot of the cost of the solution produced by AD* as a function of computation time. Here, the initial suboptimality bound ϵ was set to 3. The upper image shows the first path AD* finds. This path was found in less than 100 ms (and after 1, 715 state expansions). The cost of the path was 133, 736. Given additional deliberation time, AD* improves upon this solution, and after 650 ms, the search converges to an optimal solution. This solution is significantly shorter than the initial path (as seen in the bottom image) and has a cost of 77, 345.

Multi-resolution Planning. Figure 7 shows the benefits of using our multi-resolution lattice approach on the same simple example. The top row in the table represents a uniform high-resolution lattice, while the bottom row represents our multi-resolution lattice (in both cases, $\epsilon = 2$). Planning with the multi-resolution lattice is more than three times faster. Note that the improvement in states expanded is less than a factor of three. This is because using a multi-resolution lattice decreases not only the number of states expanded but also the time spent expanding each state, since the number of possible actions from each state is decreased.



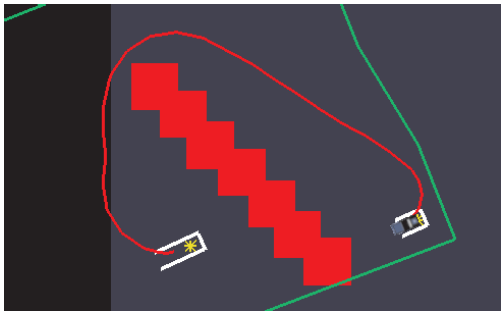
Lattice	States expanded	Time (s)
high-res	2,933	0.19
multi-res	1,228	0.06

Fig. 7. An example highlighting the benefits of the multi-resolution lattice.

Table 1 shows the comparison of planning with a multi-resolution lattice versus planning with a uniform high-resolution lattice in more detail. In particular, the table shows the time spent by the algorithm as it decreases its suboptimality bound ϵ. (The time is shown as cumulative.) The table also gives the solution cost generated by the planner in each of the cases (second column in each table). As expected, the data shows that planning with a multi-resolution lattice is about three times faster than planning with a high-resolution lattice both in producing the first solution as well as in the time it takes to converge to a provably optimal solution ($\epsilon = 1.0$). The solution quality, on the other hand, at the time of convergence is roughly the same (just slightly better for the planner that uses a high-resolution lattice).

Table 1. The performances of planning with (a) a multi-resolution lattice and planning with (b) a uniform high-resolution lattice on the example shown in Figure 7.

(a)				(b)			
ϵ	Sol. cost	# of expands	Cumul. time (s)	ϵ	Sol. cost	# of expands	Cumul. time (s)
2.0	83850	1228	0.062	2.0	102462	2933	0.191
1.9	83850	0	0.063	1.9	102462	157	0.205
1.8	83850	0	0.064	1.8	102462	269	0.225
1.7	83850	0	0.065	1.7	102462	757	0.264
1.6	83850	0	0.065	1.6	83840	1839	0.375
1.5	83850	0	0.066	1.5	83840	0	0.377
1.4	83850	0	0.066	1.4	83840	0	0.378
1.3	83850	0	0.067	1.3	83840	0	0.379
1.2	83850	0	0.068	1.2	83840	0	0.381
1.1	78168	2937	0.221	1.1	77416	5027	0.786
1.0	77345	10034	0.534	1.0	76740	16912	1.618



Heuristic	States expanded	time (s)
h	2,019	0.06
h_{2D}	26,108	1.30
h_{fsh}	124,794	3.49

Fig. 8. An example highlighting the benefits of the combined heuristic function.

Combining Mechanism-relative and Environment-relative Heuristics. Figure 8 demonstrates the benefits of using our combined heuristic function on a simple example. The first row in the table represents our combined heuristic function. It combines the 2D environment-constrained heuristic (second row) and freespace mechanism-constrained heuristic (third row). Using this combination is over 21 times faster than using the 2D heuristic alone and over 58 times faster than using the freespace heuristic alone.

Replanning. Figure 9 illustrates the replanning capability of the lattice planner. These images were taken from a parking task performed during the National Qualification Event. The top-left image shows the initial path planned for the vehicle to enter the parking spot indicated by the white triangle. Several of the other spots were occupied by other vehicles (shown as rectangles of varying colors), with detected obstacles shown as red areas. The trajectories generated to follow the path are shown emanating from our vehicle (the selected trajectory is shown in blue). As the vehicle gets closer to its intended spot, it observes more of the vehicle parked in the right-most parking spot (top, second image from left). At this point, it realizes its current path is infeasible and replans a new path that has the vehicle perform a loop (top, third image from left) and pull in smoothly (top, rightmost image). This path was favored in terms of time over stopping and backing up to reposition. The three rightmost photographs on the bottom row were taken by an onboard camera during the run. They show the vehicle approaching the row of parked cars, trying to pull in and after the vehicle is parked (in the order of left to right).

Long-range Planning. As with other teams participating in the Urban Challenge, our vehicle underwent extensive testing before and during the competition. During the competition, the planner was able to continuously plan and replan without having the vehicle ever stop to wait for a plan. The scenarios we used for testing before the competition were numerous and included expansive obstacle-laden parking lots as well as those that were narrow and highly constrained. An example of the former is shown in Figures 10(a) and (b). This parking lot is 200 m by 200 m. The robot is initially at the top of the parking lot and its goal is right at the exit of the lot (bottom right). Initially, the parking lot is unknown and as the robot traverses the lot, it discovers a series of obstacles (shown as white dots in the image on the right). The robot has to replan in real-time to account for these obstacles. The time for replanning in this scenario varied from a few milliseconds for small replanning adjustments to the path to a few seconds for finding drastically different trajectories, such as the one shown in Figure 10(b).

Complex Maneuvering. An example of a testing scenario involving a highly constrained parking lot is shown in Figures 10(c) and (d). The trajectory planned involves the robot making an initial narrow U-turn and then making another one immediately before pulling into the final parking spot. While executing the trajectory, the robot discovers a series of obstacles and has to replan as shown in Figure 10(d). The new trajectory now requires the robot to backup a number of times. Moreover, it requires the robot to enter the desired spot in reverse since the discovered obstacles prohibit the robot from pulling in.

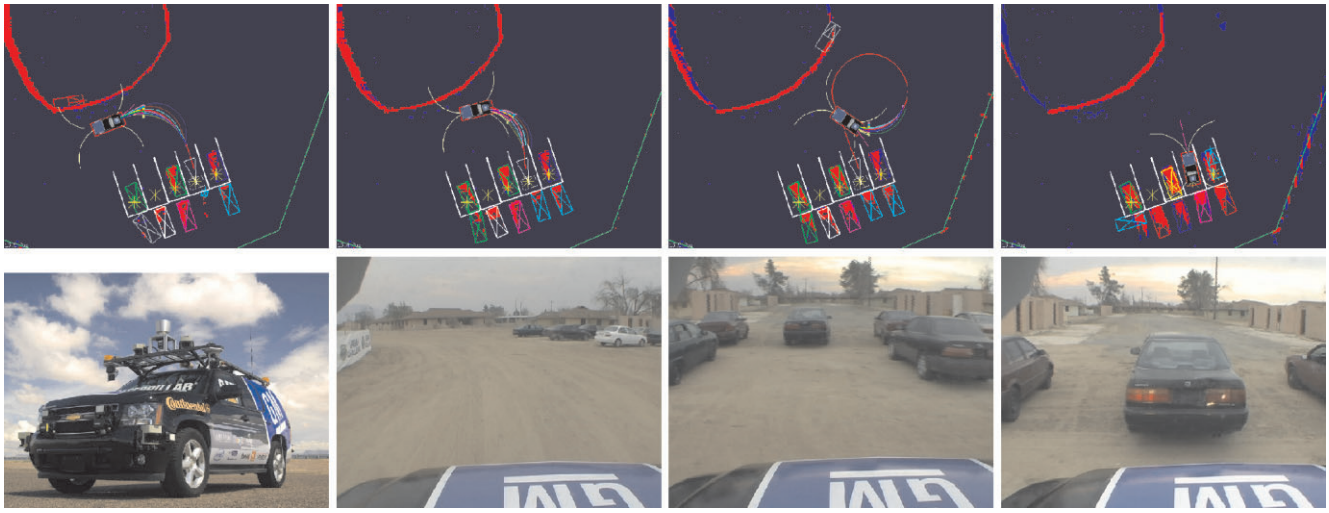
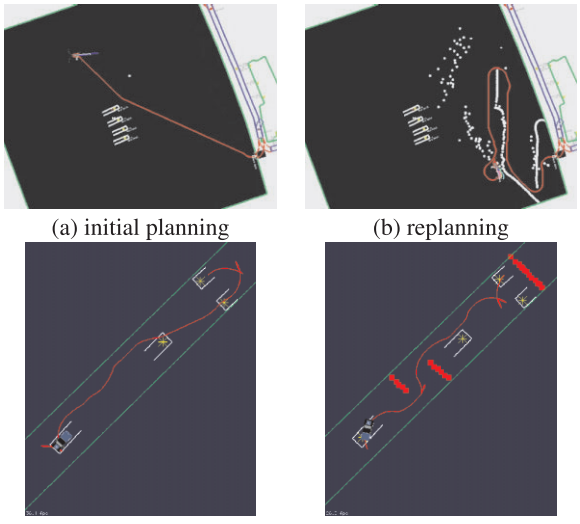


Fig. 9. Replanning when new information is received.



(a) initial planning (b) replanning

(c) initial planning (d) replanning

Fig. 10. Planning and replanning in large (a,b) and highly constrained (c,d) environments.

Coping with Dynamic Obstacles. Figure 11 shows the lattice planner being used to plan amongst several other moving vehicles in the simulation. In these images, the current goal is shown as the white triangle and the inferred short-term trajectories of the other vehicles are included as fading polygons.

Coping with Static Obstacles. Figure 12 provides an example testing scenario for our physical vehicle. The left image shows the layout of the parking lot, the static obstacles (initially unknown to the vehicle) and the parking spots to be visited in order (1 through 5). The vehicle entered the lot through the left entrance between spots 3 and 4. The other images show snapshots from an onboard camera during the vehicle's traverse through this difficult environment. They show the vehicle entering the parking lot (second image on the left), the vehicle pulling out of the parking spot 3 while turning in the direction of spot 4 (third image on the left), and traversing the parking lot towards the parking spot 4 (rightmost image).

6. Conclusions

We have presented a general approach for complex planning involving large, high-dimensional search spaces. Our approach employs a novel multi-resolution action and state space that significantly reduces complexity while providing a seamless interface between the resolutions, as well as guarantees of solution feasibility. The approach also relies on an anytime, incremental search algorithm for generating solutions in partially-known or dynamic environments when deliberation time is limited. This search exploits a low-dimensional environment-dependent heuristic coupled with a full-dimensional freespace heuristic for efficient focusing, a powerful technique applicable to many high-dimensional planning problems. The resulting approach provides global, feasible solutions to challenging navigation tasks, and all the core techniques presented are applicable to a wide range of complex planning problems.

In the future, it is important to investigate several directions in which our approach to planning may possibly be extended. Perhaps the most pressing direction would be to research an automatic way of building a multi-resolution lattice. In our current implementation, we use high-resolution lattices

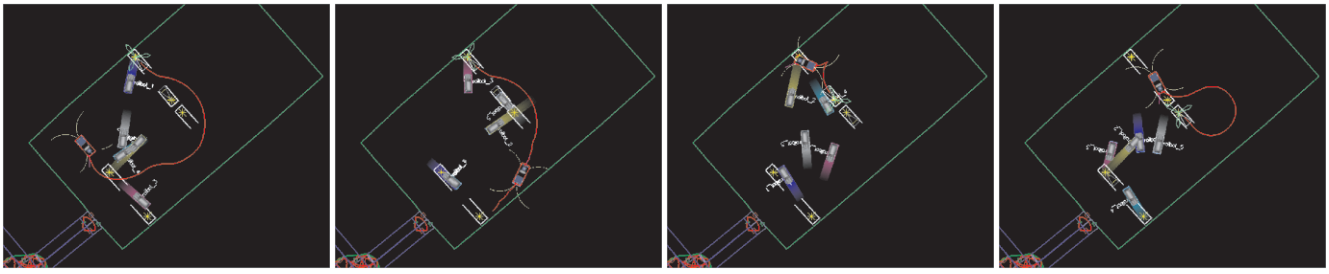


Fig. 11. Planning amongst moving obstacles.

Fig. 12. Planning in complex obstacle environments.

around the robot pose and around its goal. In some domains, however, it may be required to use high-resolution lattices in additional areas. One example of this is planning for relatively large robots operating indoors. The size of these robots often does not leave much freedom for how to traverse through doorways. As a result, the areas around the doorways would have to be modeled with high-resolution lattices. In these domains, it is therefore important to explore methods for building multi-resolution lattices that change resolutions adaptively based on some properties of the local map (for instance, obstacle density).

Another challenge that our approach would not be able to deal well with is operating in environments full of dynamic obstacles (e.g. many cars and pedestrians). In such environments, it is necessary to plan collision-free time-parameterized trajectories, as opposed to planning purely spatial trajectories that are produced by our current approach. In addition, estimating the trajectories of dynamic obstacles accurately is very difficult, and it is important to plan collision-free trajectories by taking into account the uncertainty in these estimates. Researching how to extend multi-resolution lattice-based planning to highly dynamic environments is therefore another topic of future research. Finally, we are currently working on applying our approach to other robots, including several different indoor ground robots and large outdoor unmanned vehicles.

Acknowledgements

This work would not have been possible without the dedicated efforts of the Tartan Racing team and the generous support of our sponsors including General Motors, Caterpillar and Continental. This work was further supported by DARPA under contract HR0011-06-C-0142.

References

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

Barraquand, J. and Latombe, J. (1993). Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, **10**(2–3–4): 121–155.

Braid, D., Broggi, A. and Schmiedel, G. (2006). The TerraMax autonomous vehicle. *Journal of Field Robotics*, **23**(9): 693–708.

Brock, O. and Khatib, O. (1999). High-speed navigation using the global dynamic window approach. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 341–346.

Davis, H. W., Bramanti-Gregor, A. and Wang, J. (1988). The advantages of using depth and breadth components in heuristic search. *Methodologies for Intelligent Systems, vol. 3*, Ras, Z. W. and Saitta, L. (eds.). New York, North-Holland, pp. 19–28.

Fox, D., Burgard, W. and Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics and Automation*, **4**(1): 23–33.

Gaschnig, J. G. (1979). Performance measurement and analysis of certain search algorithms. *Ph.D. Dissertation*, Carnegie Mellon University.

Howard, T. and Kelly, A. (2007). Optimal rough terrain trajectory generation for wheeled mobile robots. *International Journal of Robotics Research*, **26**(2): 141–166.

Kelly, A. (1995). An Intelligent Predictive Control Approach to the High Speed Cross Country Autonomous Navigation Problem. *Ph.D. Dissertation*, Carnegie Mellon University.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, **5**(1): 90–98.

Knepper, R. and Kelly, A. (2006). High performance state lattice planning using heuristic look-up tables. *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 3375–3380.

Koenig, S. and Likhachev, M. (2002). Improved fast replanning for robot navigation in unknown terrain. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 968–975.

Lamiraux, F., Sekhavat, S. and Laumond, J.-P. (1999). Motion planning and control for hilare pulling a trailer. *IEEE Transactions on Robotics and Automation* **15**(4): 640–652.

Laumond, J.-P., Jacobs, Taix, M. and Murray, R. M. (1994). A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, **10**(5): 577–593.

LaValle, S. and Kuffner, J. (2001). Rapidly-exploring Random Trees: Progress and prospects. *Algorithmic and Computational Robotics: New Directions*, pp. 293–308.

Likhachev, M. and Ferguson, D. (2008). Planning long dynamically-feasible maneuvers for autonomous vehicles. *Proceedings of Robotics: Science and Systems (RSS)*.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A. and Thrun, S. (2005). Anytime Dynamic A*: an anytime, replanning algorithm. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 262–271.

Likhachev, M., Gordon, G. and Thrun, S. (2003). ARA*: Anytime A* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems*. MIT Press.

Nilsson, N. (1980). *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga Publishing Company.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Boston, MA: Addison-Wesley.

Philippsen, R. and Siegwart, R. (2003). Smooth and efficient obstacle avoidance for a tour guide robot. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 446–451.

Pivtoraiko, M. and Kelly, A. (2005). Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 3231–3237.

Simmons, R. (1996). The curvature velocity method for local obstacle avoidance. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3375–3382.

Stachniss, C. and Burgard, W. (2002). An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 508–513.

Stentz, A. (1995). The Focussed D* Algorithm for real-time replanning. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1652–1659.

Svestka, P. and Overmars, M. H. (1997). Motion planning for car-like robots using a probabilistic learning approach. *International Journal of Robotics Research*, **16**(2): 119–143.

Thrun, S., Montemerlo, M., Dahlkamp, H. et al. (1998). Map learning and high-speed navigation in RHINO. *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, Kortenkamp, D., Bonasso, R. and Murphy, R. (eds.). Cambridge, MA: MIT Press.

Thrun, S., Ragusa, C., Ray, D. et al. (2006). Stanley: The robot that won the DARPA Grand Challenge. *Journal of Field Robotics*, **23**(9): 661–692.

Urmson, C., Anhalt, J., Bagnell, J. et al. (2006). A robust approach to high-speed navigation for unrehearsed desert terrain. *Journal of Field Robotics*, **23**(8): 467–508.

Urmson, C. et al. (2008). Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, **25**(8): 425–466.

Zhou, R. and Hansen, E. (2002). Multiple sequence alignment using A*. *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. Student abstract.