Efficiently Using Cost Maps For Planning Complex Maneuvers

Dave Ferguson
Intel Research Pittsburgh
4720 Forbes Ave
Pittsburgh, PA
dave.ferguson@intel.com

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

***Abstract*— We have recently developed 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. It has been implemented on an autonomous passenger vehicle that competed in, and won, the Urban Challenge. Much of the speed and robustness of our approach owes to the clever design and use of grid-based cost maps that were used throughout the planning process. In this paper, we explain the design and use of these various grid-based cost maps.**

I. Introduction

The focus of this work is planning for autonomous vehicles operating in complex urban environments. Example scenarios include navigating through congested roads and intersections and navigating and parking in large unstructured parking lots (on the order of 200×200 meters). Maneuvering at human driving speeds ($\backsim$ 15 mph) through such areas requires very efficient planning, especially if they contain previously unknown static obstacles or other moving vehicles.

Roboticists have concentrated on the problem of mobile robot navigation for several decades, providing a large body of related research. Early approaches concentrated on *local* planning, where very short term reasoning is performed to generate the next dynamically-feasible action for the vehicle[1, 2, 3]. The major limitation of these approaches is 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. More recent algorithms are based on incorporating *global* as well as local information [4, 5, 6, 7, 8, 9, 10, 11, 12]. Typically, these approaches generate a set of candidate 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, the mismatch between approximate global planning and more precise local planning, can still cause the vehicle to get stuck or take highly suboptimal paths.

Discouraged by this mismatch, a third class of planners were developed that concentrate on improving the quality of global planning to the point where a global path can be easily tracked by the vehicle [13, 14, 15, 16, 17]. However, the computational expense of generating complex global plans over large distances is challenging, and typically these approaches are restricted to either small distances, fairly simple environments, or highly suboptimal solutions.

Fig. 1. “Boss”: Tartan Racing’s autonomous vehicle entry into the Urban Challenge.

Our approach falls into this last category of high-fidelity global planners but attempts to overcome the challenges faced by these planners. In brief, there are two main ideas behind of our planner. 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.

Much of the robustness and efficiency of our approach owes to its abundant use of well-designed 2D grid-based cost maps. If properly designed, 2D cost maps can be computed efficiently and used to speedup a planner dramatically by avoiding

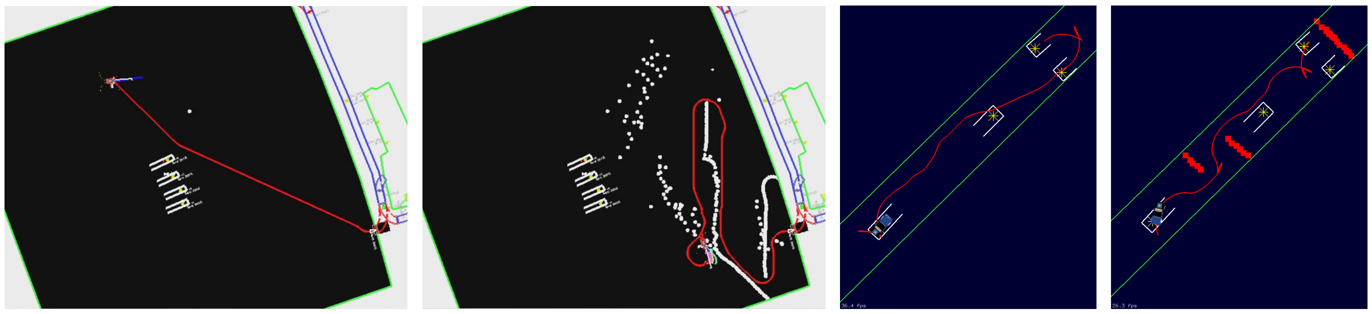
(a) initial planning (b) replanning (c) initial planning (d) replanning

Fig. 2. (a,b) show planning and replanning in a large 200m by 200 parking lot with a large number of initially unknown obstacles (shown as white dots). (c,d) show planning and replanning in a highly-constrained (very narrow) environment with initially unknown obstacles (shown in red). This environment requires trajectories that require very complex maneuvers including numerous backup maneuvers. All planning and replanning was done in real-time.

unnecessary computations. In our approach, such cost maps were used in a number of ways including: the biasing of global and local plans away from static and dynamic obstacles, the efficient generation of an informative heuristic function that guided the anytime incremental search, the reduced processing of convolutions, and the focussing of the replanning efforts of the search. All of these individual uses are orthogonal to each other and may be incorporated separately in the optimization of other planners. This paper describes each of these uses and how they were combined in our system.

II. Overall Approach

To efficiently plan a smooth path to a distant goal pose, we use a lattice planner that searches over vehicle position (x, y), orientation (θ), and velocity (v) to generate a sequence of feasible actions (each action being up to $\backsim 5$ meters long) that are collision-free with respect to the static and dynamic obstacles observed in the environment.

For each (θ, v), we pre-compute offline the set of possible actions $(x = 0, y = 0, \theta, v)$ using a trajectory generation algorithm originally developed by Howard and Kelly [9]. 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. During planning, for any state (x, y, θ, v), the planner computes the set of possible actions by looking up the set of precomputed actions for $(x = 0, y = 0, \theta, v)$ and translating it by (x, y).

The cost of each action is proportional to the time it takes to execute it. In addition, the cost is increased if the action happens in the vicinity of an obstacle. This way, paths that minimize costs are biased away from undesirable areas within the environment such as curbs.

To efficiently generate complex trajectories over large, obstacle-laden environments, the planner relies on an anytime, replanning search algorithm known as Anytime D*, developed by Likhachev et al. [16]. Anytime D* quickly generates an initial, suboptimal plan for the vehicle and then improves the quality of this solution until deliberation time expires. When new information concerning the environment is received (for instance, a new static or dynamic obstacle is observed), Anytime D* is able to efficiently 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 repair is required.

To further improve efficiency, the planner uses a multi-resolution search and action space. In the vicinity of the goal and vehicle, where very complex maneuvering may be required, a dense set of actions and a fine-grained discretization of orientation are used during the search. In other areas, a coarser set of actions and discretization of orientation are employed. However, these coarse and dense resolution variants both share the same dimensionality and seamlessly interface with each other, so that resulting solution paths overlapping both coarse and dense areas of the space are smooth and feasible. For more details on this lattice planner and its multi-resolution state and action space, see [18].

III. Using Grid-based Cost Maps

2D grid-based cost maps were employed in a number of places throughout the planning process. In the following sections we explain how they were used in each of these cases.

A. Perception Cost Maps

The most common use of grid-based cost maps in robotics is for storing the information about obstacles in the environment. In our approach, we also maintain a 2D static obstacle cost map derived from the perceptual information about the environment. We will refer to this map as a **perception map**. Geometric information from various laser range finders is processed to generate a grid map with $0.25m$ resolution, in which every grid cell contains some cost ranging from FREE to LETHAL. LETHAL costs correspond to impassable areas. By using a range of costs rather than a binary (FREE/LETHAL) map, we are able to plan paths that take into account the relative difficulty of traveling over traversable but undesirable areas, such as curbs. Detected LETHAL obstacles in the perception map are also slightly expanded by the planner

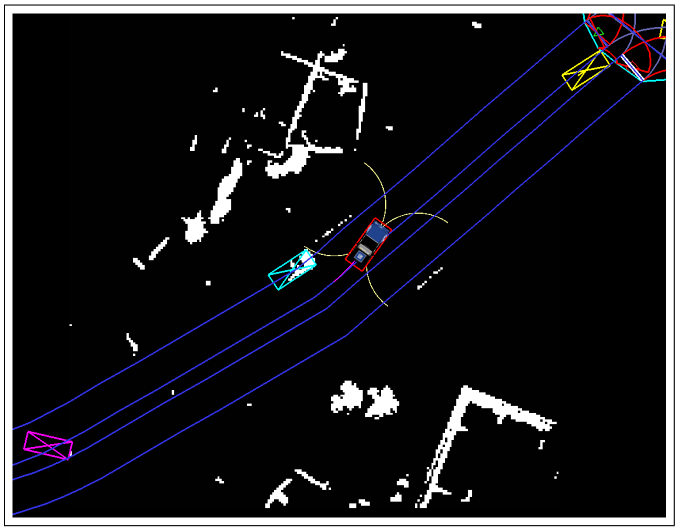

Fig. 3. Perception static obstacle map.

(by 0.5 meters, or 2 cells) to provide a conservative obstacle approximation and allow for small perceptual and execution errors. We will refer to this map as an **expanded perception map**.

Figure 3 provides an example of a perception cost map generated during the Urban Challenge. LETHAL cells are shown in white, with FREE cells in black.

B. Constrained Cost Maps

In addition to the perceptual information provided in the perception cost map, we incorporate context-specific constraints on the movement of the vehicle by creating an additional cost map, a **constrained map**. This 2D grid-based cost map encodes the relative desirability of different areas of the environment based on the road structure in the vicinity and, if available, prior terrain information. This constrained cost map is then combined with the expanded perception cost map to create the final **combined map** to be used by the planner. Specifically, for each cell (i, j) in the combined cost map $\mathcal{C}$, the value of $\mathcal{C}(i, j)$ is computed as the maximum of $\mathcal{EPC}(i, j)$ and $\mathcal{CO}(i, j)$, where $\mathcal{EPC}(i, j)$ is the expanded perception cost map value at (i, j) and $\mathcal{CO}(i, j)$ is the constrained cost map value at (i, j).

For instance, when invoking the complex planner to plan a maneuver around a parked car or jammed intersection, the constrained cost map is used to specify that staying within the desired road lane is preferable to traveling in an oncoming lane, and similarly that driving off-road to navigate through a cluttered intersection is dangerous. To do this, undesirable areas of the environment based on the road structure are assigned high costs in the constrained cost map. These can be both soft constraints (undesirable but allowed areas), which correspond to high costs, and hard constraints (forbidden areas), which correspond to LETHAL costs. Figure 4 shows the constrained cost map generated for an on-road maneuver,

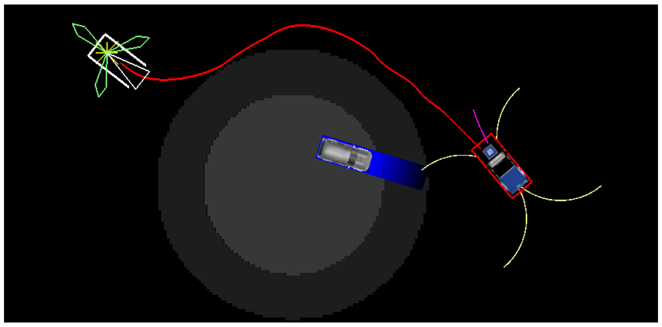

Fig. 6. Biasing the cost map for the lattice planner so that the vehicle keeps away from dynamic obstacles. Notice that the high-cost region around the dynamic obstacle is offset to the left so that Boss will prefer moving to the right of the vehicle.

along with the expanded perception cost map and the resulting combined cost map used by the planner.

For navigating in parking lots, we use the a priori specified extents of the parking lot to set all cells outside the lot in the constrained cost map to be LETHAL. This constrains the vehicle to operate only inside the lot. We also include a high, non-lethal cost buffer around the perimeter of the parking lot to bias the vehicle away from the boundaries of the lot.

When prior information exists such as overhead imagery, this information can be incorporated into the constrained cost map to help provide global guidance for the vehicle. For instance, this information can be used to detect features such as curbs or trees in parking lots that should be avoided, so that these features can be used by the planner before they are detected by onboard perception. Figure 5(a,b) shows overhead imagery of a parking lot area used to encode curb islands into a constrained cost map for the parking lot, and Figure 5(c) shows the corresponding constrained cost map. This constrained cost map is then stored offline and loaded by the planner online when it begins planning paths through the parking lot. By storing the constrained cost maps for parking lots offline we significantly reduce online processing as generating the constrained cost maps for large, complex parking lots can take up to a couple seconds.

C. Incorporating Dynamic Obstacles into the Cost Map

The combined cost map of the planner is also used to represent dynamic obstacles in the environment so that these can be avoided by the planner. In our perception architecture, we represent static and dynamic obstacles independently, which allows the planner to treat each type of obstacle differently. Our planner adapts the dynamic obstacle avoidance behavior of the vehicle based on its current proximity to each dynamic obstacle. If the vehicle is close to a particular dynamic obstacle, that obstacle and a short-term prediction of its future trajectory is encoded into the combined cost map as a LETHAL obstacle so that it is strictly avoided. For every dynamic obstacle, both near and far, the planner encodes a varying high-cost region around the obstacle to provide a safe clearance. Although these high-cost regions are

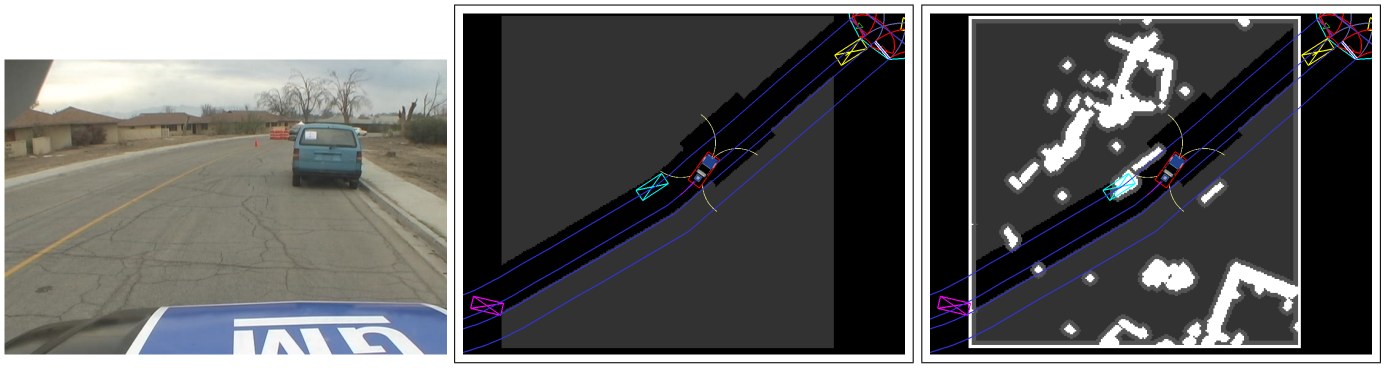
(a) (b) (c)

Fig. 4. Combining constrained cost map with expanded perception cost map - (a) show onboard image from gauntlet in course B of NQE and (b) show constrained map of road boundary and (c) show combined cost map.

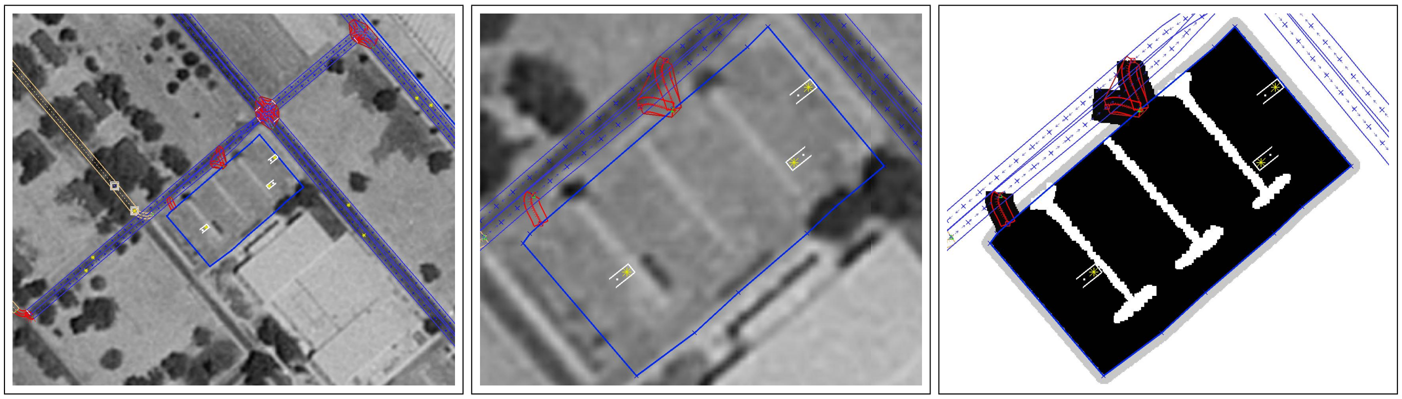
(a) (b) (c)

Fig. 5. Generatingn constrained cost maps offline. (a) Overhead imagery showing testing area with RNDF overlaid. (b) Parking lot area (boundary in blue) in RNDF with overhead imagery showing curb islands. (c) Resulting constrained cost map incorporating boundaries, entry and exit lanes, and curb islands.

not hard constraints, they result in the vehicle avoiding the vicinity of the dynamic obstacles if at all possible. Further, the generality of this approach allows us to influence the behavior of our vehicle based on the specific behavior of the dynamic obstacles. For instance, we offset the high-cost region based on the relative position of the dynamic obstacle and our vehicle so that we will favor moving to the right, resulting in yielding behavior in unstructured environments quite similar to how humans react in these scenarios. Figure 6 provides an example scenario involving a dynamic obstacle along with the corresponding cost map generated.

D. Convolution with the Cost Map

The combined cost map is used by our planner to compute the feasibility and cost of each action. Typically, one of the most computationally expensive parts of planning for vehicles is computing these action costs, as this involves convolving the geometric footprint of the vehicle for a given action with a cost map. As mentioned, our cost map has a $0.25m$ resolution and the (x, y) dimensions of our vehicle were $5.5m \times 2.25m$. Thus, even a short $1m$ action requires collision checking over 230 cells. Further, the coordinates of each of the 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 perform two optimization steps. First, for every possible action a, we pre-compute 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. No rotation is necessary since every pre-computed action a is already computed for a specific orientation θ of the vehicle.

Second, we generate two configuration space maps to be used by the planner to avoid performing convolutions. The first of these maps, called an **optimistic map**, expands all LETHAL cells in the combined map by the inner radius (Figure 8(a)) of the robot; this map corresponds to an optimistic approximation of the actual configuration space. Given a specific action a and assuming a point robot, if any of the cells through which a passes are obstacles in this optimistic map, then action a is also guaranteed to collide with an obstacle in the combined cost map. The second map, called a **pessimistic map**, expands all non-FREE cells in the combined cost map by the outer radius (Figure 8(a)) of the robot and considers those cells as obstacles. It therefore corresponds to a pessimistic approximation of the configuration space. Assuming a point robot again, if all of the cells through which an action a passes in this map are obstacle-free, then a is also guaranteed to be collision-free in the combined cost map. Only those actions that do not produce a conclusive result from these simple tests need to be convolved with the combined cost map. Typically, this is a severely reduced percentage, thus saving considerable computation. To create these auxiliary maps efficiently, we perform a single distance transform on the combined cost map and then threshold the distances using the corresponding radii of the robot for each map. Figure 7 provides an example of the optimistic and pessimistic c-space maps generated for a particular combined cost map.

E. Generating Heuristics Using the Cost Maps

The effectiveness of the Anytime D* algorithm we used for planning 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.

Since in our setup Anytime D* searches backwards, the heuristics are supposed to estimate the distance from the robot pose to state in question. Anytime D* requires them to be admissible (not to overestimate the actual distance) and consistent [19]. For any state (x, y, θ, v), the heuristics we use is the maximum of two values. The first value is the cost of an optimal path from the robot pose to (x, y, θ, v) through the search space assuming a completely empty environment. These values are precomputed offline and stored in a heuristic lookup table [17]. 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. The second value is the cost of a 2D path from the robot x_R, y_R coordinates to (x, y) given the actual environment. These values are computed online by a 2D Dijkstra's search. This heuristic function is very useful when operating in obstacle-laden environments. By taking the maximum of these two heuristic values we are able to incorporate both the constraints of the vehicle and the constraints imposed by the obstacles in the environment. The result is a very well-informed heuristic function that can speed up the search by an order of magnitude relative to either of the component heuristics alone (see [18] for details).

We compute the second heuristic function by running a single Dijkstra's search on the 16-connected combined cost map grid, starting at the cell that corresponds to the center of the current vehicle position. This search is re-run every time the vehicle pose is changed. The cost of each transition in this search is computed by taking the maximum of the costs of all the cells through which the transition passes. In addition, if any of these cells are labeled as obstacles in the optimistic map, then the cost of the transition is set to infinity. Under this cost function, a single Dijkstra's search computes the costs of shortest paths from the vehicle coordinates to all other cells in the environment[1]. Figure 9 provides the 2D cost-to-goal value function generated for the perception cost map used in Figure 7. In this figure, the darker a cell the higher its path cost.

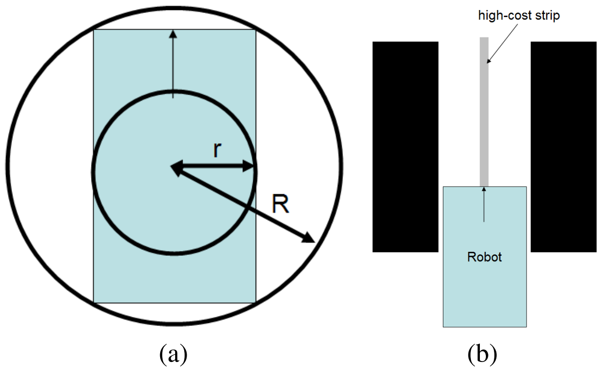

Fig. 8. (a) Inner (r) and outer (R) radii of the robot. (b) Example where the 2D heuristic function may overestimate the cost of a path derived purely from convolution.

This 2D heuristic function may overestimate the cost of the actual path. 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 8(b)). The cost of the 2D path from the initial (x_R, y_R) coordinates of the vehicle to the goal (x, y) 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 since the optimistic map will justifiably consider these cells as obstacles - 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 remedy this, we have slightly modified the cost of each action to be a maximum of two values. The first value is the convolution cost, as before. The second value is the maximum of the combined map costs of the cells that correspond to the center of the vehicle when moving along the action. This modification penalizes plans more if they involve the center of the vehicle going through high-cost areas. Most importantly, our heuristic function becomes provably admissible and consistent with respect to this cost function.

F. Efficient Incremental Planning With Cost Map Updates

With incremental planning algorithms such as Anytime D*, 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 status of the cell (x_c, y_c) in the combined cost map changes from free to LETHAL, then the costs of all actions that involve the vehicle traveling over that cell may change. Typically, there could be thousands of such actions. Anytime D* needs

[1]However, even though it is very fast, we still restrict this search to only compute shortest paths to states that are no more than twice as far (in terms of path cost) from the vehicle cell as the goal cell.

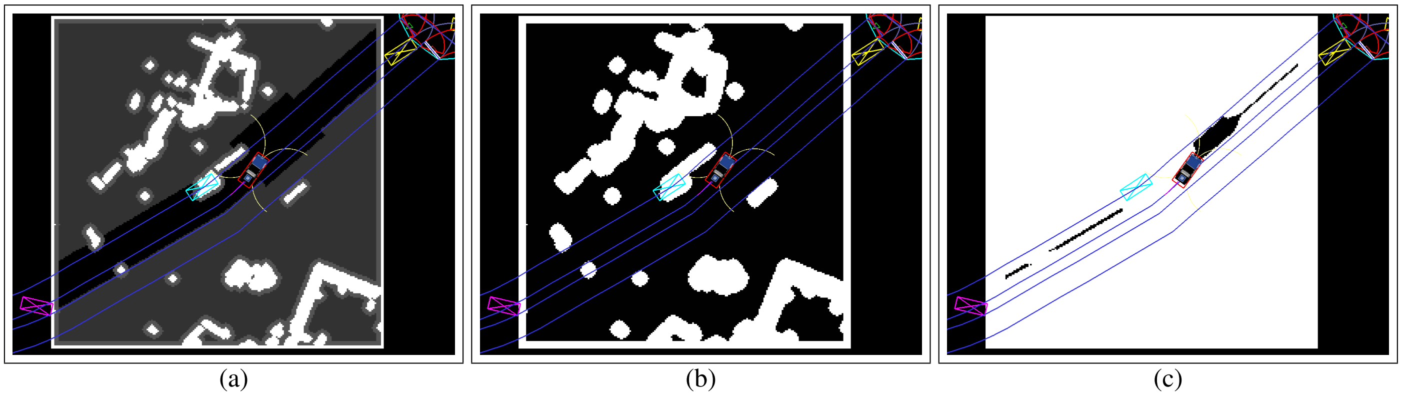

Fig. 7. (a) A combined cost map (same as from earlier figures). (b) The corresponding optimistic c-space map. (c) The corresponding pessimistic c-space map.

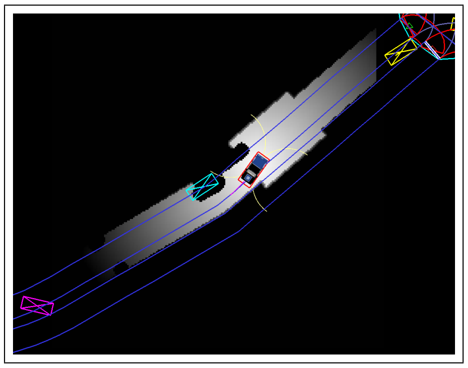

Fig. 9. 2D heuristic cost-to-robot map from the example in previous figures.

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, Anytime D* really only needs to update the values of those states that have actually been computed in the previous planning iterations. We exploit this property to decrease the computational effort involved in iterating over the states that may possibly be affected by changes in the cost map, as follows.

First, we pre-compute 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 i^{th} set $\Re^{x_i \dots x_i+d, y_i \dots y_i+d}$ 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 Anytime D* whenever the cost of the cell $(0,0)$ is modified are pre-computed 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 combined cost map.

Second, during online operations, we maintain another 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 Anytime D*. Thus, while planning, whenever Anytime D* 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) in the combined cost map is modified, for each non-empty cell $\Re^{x_i \dots x_i+d, y_i \dots y_i+d}$ in $\Re$ we look up if any one of the following four cells in the replanning map are set to true:

$$\begin{array}{llll}(((x_i + x_c) & \bmod d), & ((y_i + y_c) & \bmod d)) \\ (((x_i + x_c) & \bmod d) + 1, & ((y_i + y_c) & \bmod d)) \\ (((x_i + x_c) & \bmod d), & ((y_i + y_c) & \bmod d) + 1) \\ (((x_i + x_c) & \bmod d) + 1, & ((y_i + y_c) & \bmod d) + 1)\end{array}$$

If so, then we update the value of every state stored in $\Re^{x_i \dots x_i+d, y_i \dots y_i+d}$ translated by (x_c, y_c). No other states need to be updated since it is guaranteed that they have not been previously computed by Anytime D*. This optimization can save a tremendous amount of replanning computation.

G. Trajectory Evaluation Using Cost Maps

The path returned by our multi-resolution lattice planner 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 candidate trajectories are produced to account for dynamic obstacles and sudden new observations that could require immediate reaction (the local planner runs at 10 Hz). From this set of candidate trajectories, a single trajectory is selected for execution by the vehicle. Each of the trajectories terminates on the lattice path[2]. By having all trajectories return to the path we significantly reduce the risk of having the vehicle move itself into a state from which it is difficult to leave.

The trajectory selected for execution is typically the one that deviates least from the lattice path while also being collision-free with respect to the static and dynamic obstacles in the environment. To determine whether a trajectory is collision-free, a convolution is performed with the perception cost map.

[2]Each trajectory is in fact a concatenation of two short trajectories, with the first of the two short trajectories ending at an offset position from the path and the second ending back on the path.

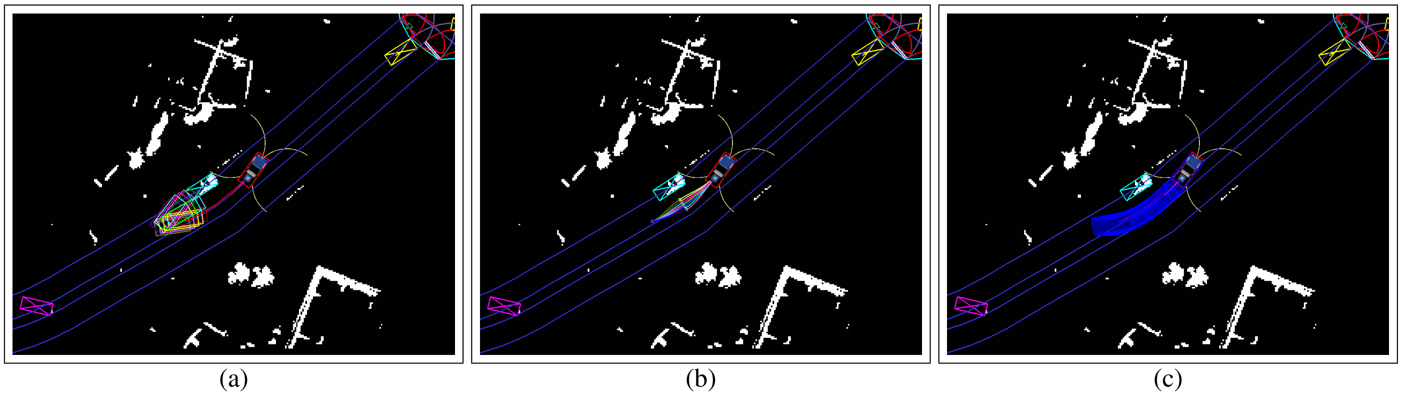

Fig. 10. Complex planning final solution. (a) The set of goals being planned to, along with the resulting path in red. (b) The trajectories generated by the local planner to track this path. (c) The convolution of one of these trajectories (in blue) with the static obstacle map from perception.

A second convolution is also performed with an extended vehicle shape to determine whether any obstacles are within a small distance of the vehicle's intended trajectory. The results of these convolutions (and other factors, such as the deviation from the path) are incorporated into the overall cost of the candidate trajectory, with the least costly trajectory chosen for execution. Figure 10(c) shows the convolution of a candidate trajectory with the perception cost map.

IV. Conclusions

In this paper, we have described how our planner uses grid-based cost maps to construct an effective cost function, to compute efficient heuristics to guide its planning efforts, to avoid unnecessary convolution and replanning calculations, and finally to evaluate various short-range trajectories generated by a local planner. The effectiveness of these techniques was demonstrated by the robustness and the speed of the planner as used in the Urban Challenge.

All of the described cost map techniques are orthogonal to each other and therefore can be used as standalone components. They are also applicable to other, non-lattice planners (e.g. grid-based planners). Given that grid-based cost maps are simple to implement and cheap and easy to maintain, we hope that the techniques presented in this paper will be helpful in the development of planners by other researchers and robotic software developers.

References

[1] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[2] R. Simmons, "The curvature velocity method for local obstacle avoidance," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1996.

[3] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance." *IEEE Robotics and Automation*, vol. 4, no. 1, 1997.

[4] S. Thrun *et al.*, "Map learning and high-speed navigation in RHINO," in *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, D. Kortenkamp, R. Bonasso, and R. Murphy, Eds. MIT Press, 1998.

[5] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[6] A. Kelly, "An intelligent predictive control approach to the high speed cross country autonomous navigation problem," Ph.D. dissertation, Carnegie Mellon University, 1995.

[7] R. Philippsen and R. Siegwart, "Smooth and efficient obstacle avoidance for a tour guide robot," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[8] S. Thrun *et al.*, "Stanley: The robot that won the DARPA Grand Challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, August 2006.

[9] T. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, 2007.

[10] C. Stachniss and W. Burgard, "An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[11] C. Urmson *et al.*, "A robust approach to high-speed navigation for unrehearsed desert terrain," *Journal of Field Robotics*, vol. 23, no. 8, pp. 467–508, August 2006.

[12] D. Braid, A. Broggi, and G. Schmiedel, "The TerraMax autonomous vehicle," *Journal of Field Robotics*, vol. 23, no. 9, pp. 693–708, August 2006.

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

[14] G. Song and N. Amato, "Randomized motion planning for car-like robots with C-PRM," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2001.

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

[16] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime Dynamic A*: An Anytime, Replanning Algorithm," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.

[17] R. Knepper and A. Kelly, "High performance state lattice planning using heuristic look-up tables," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[18] M. Likhachev and D. Ferguson, "Planning Dynamically Feasible Long Range Maneuvers for Autonomous Vehicles," 2008, submitted to Robotics: Science and Systems (RSS).

[19] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.