2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 24–28, 2017, Vancouver, BC, Canada

The Maverick planner: An efficient hierarchical planner for autonomous vehicles in unstructured environments

Neal Seegmiller, Jason Gassaway, Elliot Johnson, and Jerry Towler

Abstract— **Planning kinodynamically feasible trajectories for autonomous vehicles is computationally expensive, especially when planning over long distances in unstructured environments. This paper presents a hierarchical planner, called the Maverick planner, which can find such trajectories efficiently. It comprises two parts: a waypoint planner that uses a simplified vehicle model and an RRT* planner that respects full kinodynamic constraints. The waypoint planner quickly finds a directed graph of waypoints from start to goal, which is then used to bias sampling and speed up computation in RRT*. The Maverick planner is capable of anytime planning and continuous replanning. It has been tested extensively in simulation and on real vehicles. When planning on a sensor-generated map of the SwRI test track it can find a feasible path over 0.5 km in under 16 ms, and refine that path to within 1% of the local optimum in 0.5 seconds.**

I. INTRODUCTION

Motion planning for autonomous vehicles is challenging because their motion is subject to holonomic and non-holonomic constraints. Maneuvering a vehicle in unstructured and cluttered environments is challenging even for human drivers. In many applications the environment is not mapped a priori or can change dynamically, which limits the opportunity for precomputation. The motion plan must be continually updated as new perception data is received. This paper presents a hierarchical planner called the Maverick[1] planner, which can navigate efficiently in such environments.

A. Related Work

Many path planning approaches have been developed for autonomous vehicles. Planning trajectories that respect all kinematic and dynamic constraints is computationally expensive, so it is frequently done only over a receding horizon [1] with remaining cost to go merely estimated using a heuristic. While this is adequate for on-road driving and open environments, it can cause the vehicle to get stuck. For example, the heuristic may direct the vehicle to a narrow gap that turns out not to be traversable.

Lattice planners address this problem by planning a feasible path the entire way from start to goal. They build a graph of regularly spaced vehicle states connected by edges of kinodynamically feasible trajectories [2]. Trajectories between any two states in the graph can be found using graph-search algorithms like A*. Building a graph that covers the entirety of drivable space in the environment is expensive, especially when the perceived environment keeps changing.

Probabilistic planners such as rapidly-exploring random trees (RRTs) can potentially find entire paths more quickly. They work by randomly sampling vehicle states and connecting them to nearest neighbors in a tree of reachable states rooted at the start. RRTs are capable of respecting kinodynamic constraints [3][4], but doing so increases the dimensionality of the state space and thus the number of samples required. Naïve RRT planners also require many samples to find paths through narrow passageways. Many techniques have been proposed to reduce the number of samples required for probabilistic planners. For example, biasing samples to be near obstacles by Gaussian blurring them [5], or biasing them towards unexplored areas by sorting samples based on a dispersion-reduction metric [6], or biasing them away from some nodes in the tree based on local limitations of system dynamics or a history of unsuccessful expansion [7].

Karaman and Frazzoli presented a variant of RRT called RRT* that optimizes the path to the goal by rewiring nodes in the tree through newly inserted ones [8]. To speed up the optimization, Islam et al. propose biasing samples to be near obstacle vertices, as approximately determined by performing visibility tests on nodes along the path [9].

B. Key Features of the Maverick planner

The contribution of this work is to combine existing motion planning techniques into a novel hierarchical planner that mitigates the weaknesses and leverages the strengths of its components. The Maverick planner uses Voronoi diagram and cell decomposition techniques to quickly generate a directed acyclic graph (DAG) of waypoints (displayed in green in Fig. 1). The waypoint graph represents several routes (sequences of waypoints from start to goal) belonging to different homotopy classes. The waypoint graph aids an RRT* variant to find a kinodynamically feasible path more quickly than traditional RRT*, by reducing the sample space and speeding up the search for neighboring nodes.

Key features of the Maverick planner include:

- In practice, it retains the probabilistic completeness of traditional RRT* when the waypoint planner meets the condition explained in Section II-A.

*This work was supported by the U.S. Army Tank Automotive Research Development and Engineering Center (TARDEC). UNCLASSIFIED: Distribution Statement A. Approved for public release; distribution is unlimited.

The authors are with the Unmanned Systems Section, Southwest Research Institute, San Antonio, TX 78238, USA. neal.seegmiller@swri.org, jason.gassaway@swri.org, elliot.johnson@swri.org, jerry.towler@swri.org

[1]The term "maverick" refers to an unbranded head of cattle or wanderer. We named our planner Maverick because its initial application was exploratory wandering in unstructured environments.

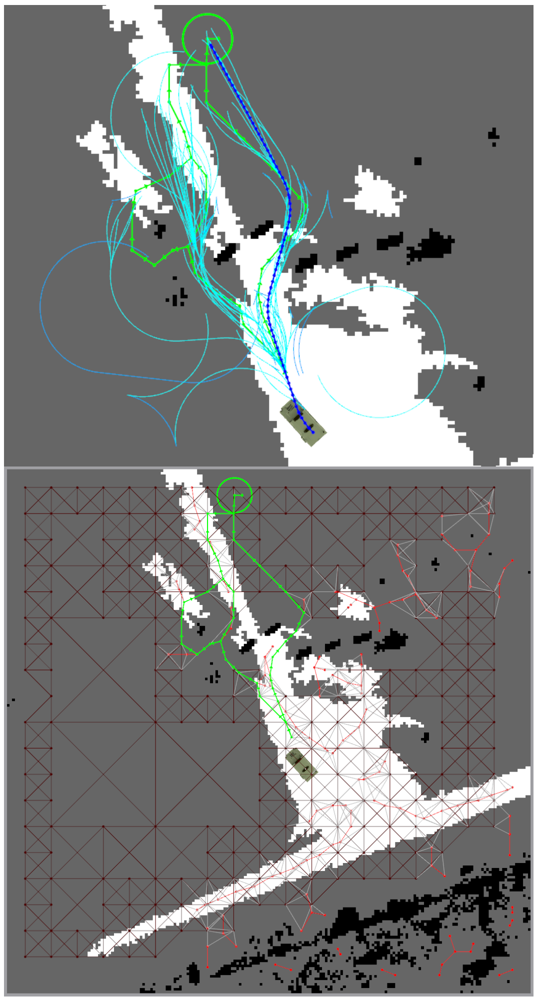

Fig. 1. A visualization of the Maverick planner in action at the SwRI test track (best viewed in color). The top image shows the RRT* tree of kinodynamically feasible paths (light blue) and the best path to the goal (dark blue). The bottom image shows the lattice generated by the waypoint planner composed of a Voronoi graph (bright red) and grid graph (dark red). The waypoint graph (green) is extracted from the lattice and is used to bias sampling and speed up computation in RRT*. The rasterized costmap is binned into three values: road (white), offroad (gray), and obstacle (black).

- In practice, it converges on the same optimal solution as traditional RRT* when continuously replanning, as explained in Section III-B.
- It can leverage structure in the environment when available, but doesn't require it. It prefers to follow roads/trails, but will navigate off them when necessary.
- Its RRT* steer function is capable of driving in reverse, executing K-turns, etc. to maneuver in tight spaces. It respects steering limits and (optionally) steering rate limits.
- It is capable of "anytime" planning. It finds a feasible path quickly, then spends any remaining computation time optimizing the path.
- It is capable of continuous replanning in real time. This enables planning into previously unmapped and dynamic environments.
- It is integrated with "persistent map" software for exploration and planning over very long distances.

The remainder of the paper describes our waypoint planner implementation (Section II), our RRT*-based kinodynamic planner (Section III), experimental results (Section IV), and possible extensions for the Maverick planner (Section V).

II. WAYPOINT PLANNER

This section presents our waypoint planner implementation, which takes as input start and goal locations and a 2D costmap. The waypoint planner converts the map into a sparse representation of drivable space: a lattice (or graph) of x,y coordinate vertices connected by undirected straight-line edges. These edges are drivable for a simplified vehicle model (Section II-A). The lattice is constructed from Voronoi and grid components (Sections II-B and II-C). Several of the lowest-cost routes from start to goal are extracted from the lattice and represented as a waypoint graph for use by the kinodynamic planner (Section II-D).

Our implementation uses a rasterized costmap in which pixel values represent the maximum allowable speed when traversing that space (faster on roads, slower off-road, and zero for obstacles). Edge cost in the the lattice is the time required for traversal, which equals edge length divided by the costmap-allowed speed. Our implementation is sufficiently fast to replan in real-time as the costmap is dynamically updated. Our approach can be generalized to other map representations.

A. *Simplified Vehicle Model*

In practice, the Maverick planner retains the probabilistic completeness properties of traditional RRT* because for every kinodynamically feasible path from start to goal, the waypoint planner can find a homotopic route (i.e., the route can be deformed into the path without crossing obstacles). This means the waypoint planner does not preclude RRT* from searching in any homotopy class that may contain a feasible path.

Our waypoint planner satisfies this condition by using a simplified vehicle model, with fewer constraints and smaller dimensions than the full-fidelity model used by RRT*. Specifically, the waypoint planner assumes a disc-shaped differential drive vehicle with a diameter less than the width of the true vehicle (Fig. 2). The simplified vehicle can turn in place, so the waypoint planner doesn't need to consider yaw. The simplified vehicle can drive straight lines between any sequence of x,y locations, and collision checking any x,y location requires only checking a single pixel in a dilated costmap. Note that the simplified vehicle can drive any path that is kinodynamically feasible for the true vehicle, but not vice versa. Using this simplified model enables the waypoint planner to quickly construct a graph (or lattice) of drivable space.

B. *Voronoi Graph*

The waypoint planner uses the generalized Voronoi diagram (GVD) of the rasterized costmap to construct the lattice on the centerlines of narrow passageways between obstacles.

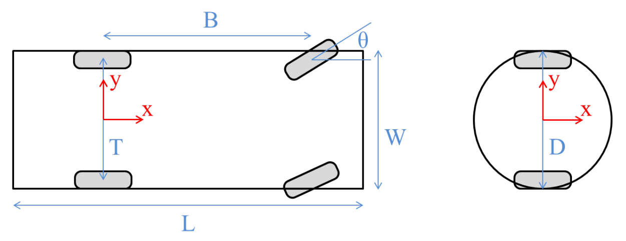

Fig. 2. The kinodynamic planner uses a full-fidelity model of an Ackerman-steered vehicle (left). The waypoint planner uses a simplified model (right): a differential-drive vehicle with no steering/curvature limits whose footprint is circumscribed by the actual vehicle footprint ($D \leq W$).

Many algorithms have been published for computing GVDs from raster maps [10][11][12], so we omit the exact algorithm here. We convert the GVD into a graph, then sparsify it by iteratively contracting edges until doing so would violate an edge length limit. To generate the Voronoi lattice on roads, offroad pixels are treated as obstacles.

C. Adaptive Resolution Grid Graph

The waypoint planner uses an adaptive resolution grid to complete the lattice in open areas. Construction of the grid is performed similarly to prior work on quadtree approximate cell decomposition [13]. If a cell in the costmap is clear of obstacles, it is represented in the graph by 4 fully-connected corner vertices; else, the cell is recursively subdivided into 4 smaller cells until the resolution limit is reached. The Voronoi and grid graphs are merged by adding edges between Voronoi vertices and their 4 neighboring grid vertices if the line between them is collision-free.

Individually, Voronoi and approximate cell decomposition planning approaches have weaknesses. The Voronoi approach maximizes distance from obstacles; while this provides a useful safety margin in cluttered areas, it causes suboptimality and ranging sensor deprivation in open areas. Approximate cell decomposition is not complete, as it can fail to find routes through narrow passageways. Our combination of these approaches mitigates these weaknesses. This is illustrated in Fig. 1 where lattice vertices/edges are colored bright red if constructed from the GVD and dark red if constructed from cell decomposition.

D. Searching the Graph

Vertices are added to the lattice for the start and goal locations. The cheapest route between them can be found using A*, but finding multiple routes belonging to different homotopy classes is recommended in case no kinodynamically feasible path can be found along the cheapest route. We do this by computing the total route cost via each vertex, then iterating over the routes from cheapest to most expensive, adding them to the waypoint graph only if they diverge sufficiently from the routes already tried. If no kinodynamically feasible path can be found along any route in the waypoint graph, a larger graph can always be requested. Unlike the undirected lattice edges, waypoint graph edges are directed towards the next vertex on the route to the goal. Waypoints also have a yaw angle based on the orientation of the edge to the next vertex.

Algorithm 1 RRT* using Waypoint Graph

```
 1: Given: $W$, $T$, $C$ ▷ waypoint graph, RRT* tree, costmap
 2: while time elapsed < time limit do
 3:     sample waypoint index $i$
 4:     sample pose $\rho$ near $W[i]$
 5:     if $T.insert(\rho)$ fails then
 6:         $\rho \leftarrow flipOrientation(\rho)$
 7:         if $T.insert(\rho)$ fails then
 8:             continue
 9:         end if
10:     end if
11:     $T.rewire(\rho)$
12: end while
```

III. KINODYNAMIC PLANNER

This section presents a kinodynamic planner that takes as input a rasterized costmap and a DAG of waypoints from start to goal. In simple environments, one might smooth routes (sequences of waypoints in the DAG) to find a kinodynamically feasible path; we instead use a variant of RRT* to search for feasible paths near routes. This sampling-based approach is much more effective in challenging environments, such as when K-turns are required to maneuver through clutter. Section III-A and Algorithm 1 explain the planning process. Section III-B explains the process for continuous replanning.

A. RRT* Planning Using the Waypoint Graph

1) Sampling: Sampling involves randomly choosing a waypoint from the graph (line 3), then sampling a pose in the continuum within some radius and yaw tolerance of that waypoint. For best "anytime" planning performance, we maintain two exclusive subsets of waypoints in the graph: traversed and frontier. When a node is successfully inserted into the tree, its associated waypoint and that waypoint's ancestors are put in the traversed set. n generations of the associated waypoint's descendents are put the frontier set (if not already traversed). When randomly choosing a waypoint, we choose from the frontier set with probability p and from the traversed set with probability $(1 - p)$. If p is high, the planner will prioritize expanding the frontier until the planned path reaches the goal over optimizing the path.

2) Finding neighbor nodes: Both the insert and rewire operations in RRT* require finding neighbor nodes. Traditionally, this is done by computing some distance metric for every node in the tree, which can be expensive for large trees. Traditional distance metrics (Euclidean, Dubins car) may identify a node as a neighbor even though steering to/from that node is unlikely to succeed (e.g., if separated by a wall). The waypoint graph provides a simple, fast way to find neighbors for which steering success is likely. Nodes are considered neighbors if their associated waypoints are within n degrees of separation in the waypoint graph. For an insert

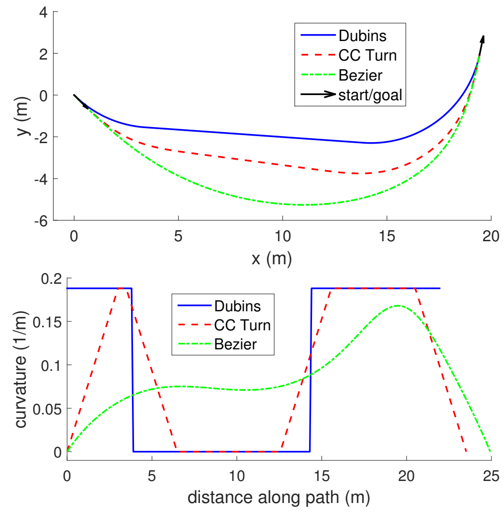

Fig. 3. Steer function solutions for a feasible path between two vehicle poses. The Dubins car path satisfies steering (curvature) limits. The continuous curvature turn path (CC Turn) path additionally satisfies steering rate limits. The Bezier path corresponds to continuous and (in this example) less-sharp steering inputs.

operation, we find ancestors of the new node's associated waypoint, but for a rewire operation we find descendents. If we maintain a mapping between associated waypoints and nodes, we can quickly look up neighbor nodes for a set of ancestors/descendents without iterating over the entire tree.

3) Steer function: Both insert and rewire operations require a "steer function" which computes a kinodynamically feasible trajectory between two states, ignoring obstacles. Our steer function computes a set of Dubins and Reeds Shepp paths between poses [x,y,ψ (yaw)] and returns the cheapest. These paths implicitly satisfy steering limits. Dubins paths stay in a single gear, whereas Reeds Shepp paths can alternate between forward and reverse [14]. We compute only the "CCC" set of Reeds Shepp paths which includes K-turns.

Our steer function can optionally be configured to produce continuous curvature paths, as shown in Fig. 3. The "CCTurn" path is an optimized sequence of continuous curvature turns that respect steering rate limits (similar to [15]). The "Bezier" path also has continuous curvature and is computed using Pythgorean hodograph quintics [16] for fairness. When requiring continuous curvature (κ), the state space expands to [x,y,ψ,κ]. Velocity is omitted because we assume a controller that predictively computes speed and steering inputs such that the vehicle follows the planned path sufficiently closely to avoid collisions. Path sharpness affects the speed at which a steering rate limited vehicle can follow it; following a Dubins/Reeds Sheep exactly would require stopping at curvature discontinuities to steer the wheels.

Path cost is equal to the time required for traversal, which accounts not only for path length but also for slower speeds off road and in reverse gear and for the time to shift gears. Path cost is infinite if collision checking fails (i.e., the vehicle's footprint intersects an obstacle at any point).

4) Insert and rewire: To insert a new node or pose into the tree, we compute the cost to reach that node via each of its neighbors. We select the neighbor for which cost is cheapest to be the parent. If we fail to insert the node (i.e., if paths via all neighbors collide) we may flip the orientation of pose and attempt to insert again (line 6). This enables the vehicle to drive in reverse out of tight spaces. If the node is successfully inserted, we attempt to rewire other neighbors in the tree through it (line 11); rewiring only proceeds when it reduces the cost of reaching the neighbor.

B. *Continuous Replanning*

Replanning is required whenever the perceived environment changes, due either to new areas coming within range of the perception sensors or to moving objects (e.g., other vehicles or pedestrians). Our approach to replanning uses some techniques from prior work [17][18], and some new ones. These techniques make use of the planned path and RRT from the previous iteration.

First, we lock an initial portion of the planned path (called the "committed trajectory" in [18]) to prevent sudden changes in steering. The length of the locked portion is proportional to the speed of the vehicle. For safety, this requires that the controller executing the planned path be capable of stopping the vehicle when necessary to avoid a collision. Once the vehicle stops, the lock disappears and planning resumes from the vehicle's current pose.

Second, we augment the waypoint graph with additional waypoints sampled from along the planned path. This causes the RRT* planner to continue expanding the tree around the planned path, which improves stability and allows large optimizations over multiple iterations. In practice, the Maverick planner converges on the same optimal solution as traditional RRT*, but requires far fewer samples.

Finally, we reuse the tree between iterations, which requires several steps. If the costmap changed, we recompute the cost of paths between nodes. If the waypoint graph changed, we associate nodes with the nearest waypoint in the new graph. We rewire the entire tree through a new root node (located at the end of the locked path). We delete nodes that are no longer reachable (as does [17]). To prevent the tree from growing too large, [18] culls nodes if the cost to reach the node plus a heuristic cost to go exceeds the planned path cost. This isn't helpful in our case, because biasing samples to the waypoint graph already ensures that nearly all nodes satisfy this heuristic cost test. Instead, we cull unproductive nodes—those that have no children and for some number of insert operations have failed to be identified as a neighbor or failed to be selected as the parent.

IV. EXPERIMENTAL RESULTS

The Maverick planner has been tested extensively on multiple vehicles, including drive-by-wire Polaris MRZR, Jeep Rubicon, and HMMWV platforms. On all these vehicles the Maverick planner runs in real time on onboard computers, even as the map is dynamically updated. Due to space limitations, this paper presents experimental results

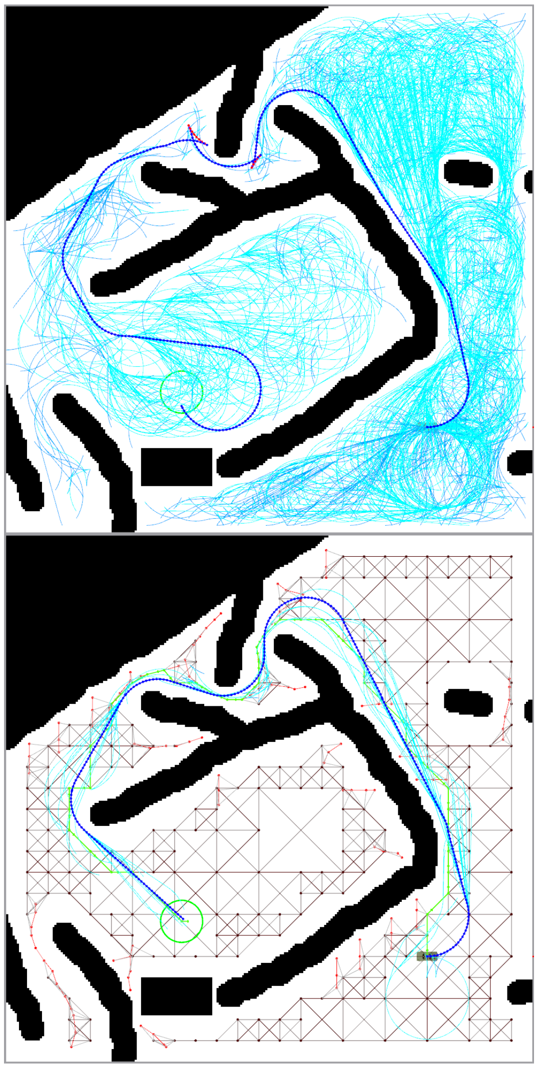

Fig. 4. The RRT* tree and planned path when using unbiased sampling after 20 s (top) and when using the waypoint graph after just 0.1 s (bottom). When using unbiased sampling, a much larger tree is required to find a feasible path, and the path is more expensive as some segments require driving in reverse (drawn in red vs. blue).

in simulation and using sensor-generated map data offline. This allows multiple repetitions of controlled experiments to obtain statistically significant results.

A. Comparison to Unbiased Sampling

Here we compare the effectiveness of the RRT* kinodynamic planner with and without the aid of the waypoint graph. When without a waypoint graph, the RRT* planner performs unbiased sampling of poses throughout the costmap and determines nodes to be neighbors if they are within a Euclidean distance tolerance (20 m). Fig. 4 shows a representative RRT* tree and lowest-cost path to the goal for both variants of the planner in a simulated environment; here RRT* must find a path over 130 m long through a narrow passageway. Table I presents the average path cost, number of samples, number of nodes in the tree, and elapsed computation time for both planner variants operating at

TABLE I

COMPARISON TO RRT* WITH UNBIASED SAMPLING[1]

planner	path cost (s)	num samples	num nodes	elapsed time (ms)
unbiased (1st[2])	65.4	3807.0	1170.2	10447.3
unbiased (0.05 Hz)	53.3	5128.2	1854.0	19996.8
Maverick (1st)	47.2	60.6	27.1	11.1
Maverick (10 Hz)	36.5	165.6	107.8	89.7
Maverick (1 Hz)	30.5	520.5	393.0	991.5

1. Values are averaged over 10 or more iterations in a single thread on an Intel Xeon E5-1620 processor.
2. Average values when the first feasible solution is found.

various frequencies. When using the waypoint graph, the computation time required to find a feasible solution is reduced by three orders of magnitude. Furthermore, path cost is 43% cheaper when planning for 1 s with the waypoint graph compared to planning for 20 s without.

The Maverick planner is far more efficient than traditional RRT* even when accounting for the computation time required to generate the waypoint graph. In this experiment, computing the Voronoi/grid graph lattice takes 3.5 ms, and extracting the waypoint graph from the lattice takes 0.6 ms.

B. Results in a Real Environment

Here we evaluate Maverick planner performance operating on a "persistent map" of the SwRI test track. This map was generated from lidar/stereo camera data using proprietary SLAM and material classification software. Fig. 5 shows the RRT* tree and planned path, as well as the lattice and waypoint graph. Note that the planner gives preference to roads and searches along the top and bottom of the loop simultaneously. Table II presents path cost and computation cost metrics for the kinodynamic planner operating at various frequencies. On average, a feasible path over 580 m long is found in just 15.7 ms with only 79 nodes in the RRT* tree. Continuously replanning at 10 Hz for 0.5 s (per Section III-B) optimizes the path as effectively as allowing a single RRT* iteration tree to grow the tree for 5 s. Both reduce path cost to 118 s, but replanning is more efficient because it culls unproductive nodes, making insert/rewire operations much faster. In 0.5 s of replanning, the planned path is optimized to within 1% of the optimum that the planner ultimately converges on.

Persistent map updates are published on a per-tile basis; when an updated tile is received, the waypoint planner updates only the region of the lattice that intersects the tile. In this experiment, extracting the multi-route waypoint graph from the lattice takes 19 ms, but this is non-blocking. When continuously replanning, the waypoint and kinodynamic planners can run asynchronously; the kinodynamic planner only needs to wait for a new waypoint graph when the map or start/goal locations change dramatically. Replanning can occur while the vehicle drives along the current best path.

In these experiments, the steer function respects curvature limits but was not configured for continuous curvature (per Section III-A.3), because we currently use a predictive path following controller that does not require it.

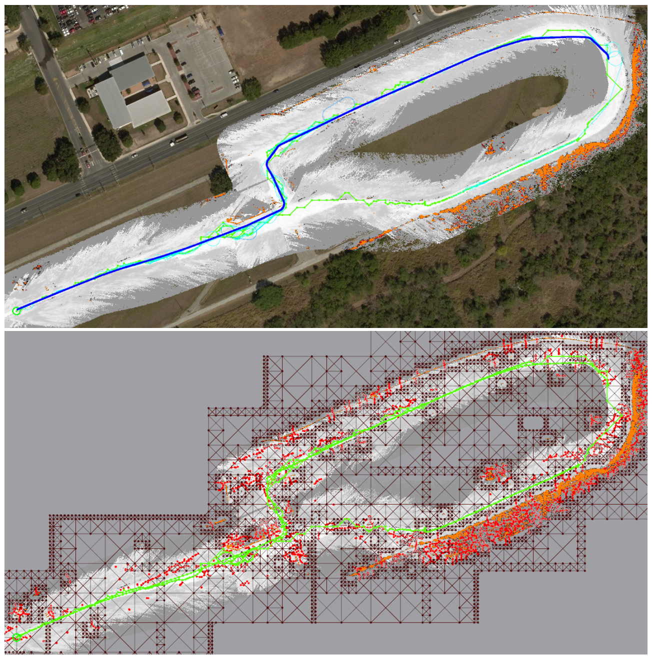

Fig. 5. (top) RRT* tree and planned path (584 m) when planning on a persistent map of the SwRI test track. (bottom) The lattice (red/gray) and waypoint graph (green) used to bias sampling in RRT*.

TABLE II

MAVERICK PLANNER RESULTS ON MAP OF SWRI TEST TRACK

rate	path cost (s)	num samples	num nodes	elapsed time (ms)
1st	203.3	80.5	79.1	15.7
1 Hz	122.2	924.2	871.5	988.3
0.2 Hz	118.0	1968.3	1904.9	4989.6
replan (10 Hz)	117.1[1]	92.1[2]	280.2	85.9

1. This is the path cost converged to. The path costs after the first 7 replan iterations are 187.8, 137.7, 120.7, 119.0, 118.4, 118.0, and 117.9.
2. This and remaining columns are average per iteration values

V. CONCLUSIONS

The Maverick planner presented in this paper combines several well-known motion planning techniques (Voronoi diagrams, cell decomposition, RRT*) in a novel way that mitigates their individual weaknesses and leverages their strengths. Most significantly, it improves the efficiency of RRT* by using a waypoint graph to bias sampling and speed up the search for neighboring nodes. In experiments, the Maverick planner found lower-cost paths up to three orders of magnitude faster than traditional RRT*.

Unfortunately, source code for the current Maverick planner implementation is confidential. In future work, we hope to make an open-source implementation that is integrated with Open Motion Planning Library [19]. This would permit benchmarking against other state of the art techniques.

In future work, we will investigate different topological map representations for waypoint planning, such as polygon decomposition, that may represent the environment topology with less redundancy. We also plan to extend the Maverick planner to accommodate other map representations besides raster maps, account for 3D terrain, and use a 3D vehicle dynamics model.

ACKNOWLEDGMENT

SwRI acknowledges the support of David Conger and other TARDEC personnel for conducting evaluation of the Maverick planner.

REFERENCES

[1] C. Urmson *et al.*, "Autonomous driving in urban environments: Boss and the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[2] M. Pivtoraiko and A. Kelly, "Kinodynamic motion planning with state lattice motion primitives," in *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2011.

[3] D. J. Webb and J. v. d. Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *International Conference on Robotics and Automation*. IEEE, 2013.

[4] J. Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma, "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," in *American Control Conference*, 2013.

[5] V. Boor, M. H. Overmars, and A. F. v. d. Stappen, "The Gaussian sampling strategy for probabilistic roadmap planners," in *International Conference on Robotics and Automation*. IEEE, 1999.

[6] S. R. Lindemann and S. M. Lavalle, "Incrementally reducing dispersion by increasing Voronoi bias in RRTs," in *International Conference on Robotics and Automation*. IEEE, 2004.

[7] L. Jaillet, J. Hoffman, J. v. d. Berg, P. Abbeel, J. M. Porta, and K. Goldberg, "EG-RRT: Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles," in *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2011.

[8] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Robotics: Science and Systems*, 2010.

[9] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution," in *International Conference on Mechatronics and Automation*. IEEE, 2012.

[10] C. Li, J. Chen, and Z. Li, "A raster-based method for computing Voronoi diagrams of spatial objects using dynamic distance transformation," *International Journal of Geographical Information Science*, vol. 13, no. 3, pp. 209–225, 1999.

[11] P. Bhattacharya and M. L. Gavrilova, "Roadmap-based path planning: Using the Voronoi diagram for a clearance-based shortest path," *Robotics and Automation Magazine*, vol. 15, no. 2, pp. 58–66, 2008.

[12] N. Kalra, D. Ferguson, and A. Stentz, "Incremental reconstruction of generalized Voronoi diagrams on grids," *Robotics and Autonomous systems*, vol. 57, no. 2, pp. 123–128, 2009.

[13] N. Katevas, S. Tzafestas, and C. Pnevmatikatos, "The approximate cell decomposition with local node refinement global path planning method," *Journal of Intelligent Robotic Systems*, vol. 22, pp. 289–314, 1998.

[14] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[15] T. Fraichard and A. Scheuer, "From Reeds and Shepps to continuous-curvature paths," *Transactions on Robotics*, vol. 20, no. 6, 2004.

[16] R. T. Farouki and C. A. Neff, "Hermite interpolation by Pythagorean hodograph quintics," *Mathematics of Computation*, vol. 64, no. 212, pp. 1589–1609, 1995.

[17] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *International Conference on Robotics and Automation*. IEEE, 2006.

[18] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT," in *International Conference on Robotics and Automation*. IEEE, 2011.

[19] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.