Navigation in Hybrid Metric–Topological Maps

Kurt Konolige, Eitan Marder-Eppstein, Bhaskara Marthi
Willow Garage, Inc.

***Abstract*— We present an approach for navigation in hybrid maps consisting of a topological graph overlaid with local occupancy grids. The topological graph is built on top of a graph SLAM system, which can be efficiently optimized even for very large environments. The novel feature of our system is that it navigates locally using local metric maps, while the overall plan is formed on the topological graph. Unlike many current SLAM methods, we never reconstruct a full occupancy grid of the environment for localization or path planning. We show that our method generates near-optimal plans, and deals gracefully with changes to the map.**

I. Introduction

Most current SLAM systems generate global, metric maps of the workspace. While convenient for small areas, global metric maps have inefficiencies of scale: for example, planning in a large metric map quickly grows unwieldy. There is also a problem of maintaining global consistency when closing large loops, which in the worst case grows cubically in the size of the map for both EKF and constraint-based maps. Finally, representing multilevel structures such as parking garages leads to even larger representations, as full 3D space must be taken into account, rather than the typical 2D roadmap.

Many researchers have noted that, for the purposes of navigation, a globally consistent metric map is not a necessity. Rather, over larger spaces, topological connections with rough metric information suffice for planning, while local metric information can be used for more precise localization and obstacle avoidance. This insight has led to many algorithms for SLAM mapping systems that use submaps to control the computational and consistency issues of scaling. Surprisingly, there has been very little work done on a major reason for mapping: navigation. In this paper, we address the problem of designing a practical navigation system for hybrid maps that exhibits good scaling properties, while maintaining an overall quality of path planning similar to global metric maps.

Our navigational system assumes a SLAM framework based on a set of 2D robot poses and constraints between them (a *pose graph* [1], [2]. This graph naturally forms a 2D *manifold* [3], which can represent 3D structures such as parking garages. Optimization of the global pose graph can be done very efficiently with recent methods [4]; however, reconstructing the global occupancy grid is a bottleneck. Instead, we only form occupancy grids in overlapping neighborhoods of limited size, using sensor readings stored at the pose nodes. The local grids are used for localization, map update, and local navigation (see Figure 1).

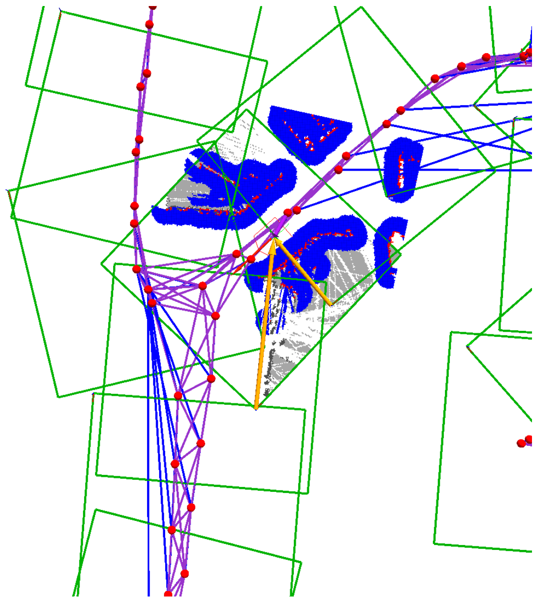
Fig. 1. The robot using the hybrid metric-topological navigation system. Red circles are nodes of SLAM and nav graphs; blue lines are SLAM constraints, purple ones are nav links. Green squares show the location of local occupancy grids. The robot is navigating in the grid using a local metric planner.

The main contribution of this paper is in the design and use of the local grid structures for localization and navigation, and the creation of a topological map and planner from the pose graph structure. The topological map is related to the pose graph, but imposes additional requirements.

- The nav graph is *consistent*, that is, two nodes are connected if and only if there exists a valid (metric) path between them.
- Nav graph planning is *efficient*, that is, the executed paths are as short as those from the global metric map.

The efficiency of the navigational planner comes from the use of local metric maps in the executing navigational plans on the topological graph.

The topological navigational system gives rise to some pleasing properties. Planning can be done much faster on the nav graph than on a full metric map. Since only neighborhood metric information is needed, occupancy grids can be constructed in an incremental manner for small areas, and redone when sensors indicate changes in the environment. This paves the way for real-time updating and replanning on the graph, and we show some preliminary results.

In the rest of the paper, we present our navigational algorithm. In Section II related work is presented and compared. In Section III, we describe the SLAM graph, and then the construction of the navigational graph on top of it. The subsequent section details planning and execution on the nav graph, followed by experiments and the conclusion.

II. Related work

Our work is in the tradition of a long line of research that combines metric and topological information in creating and navigating in large-scale maps. We can distinguish several broad themes. First, SLAM research in large-scale mapping often invokes hierarchies of submaps with constraints between them. Second, topological approaches are primarily concerned with behavioral navigation between "important perceptual places." Third, and closest to the present work, are hybrid approaches that use local metric information, while maintaining a graph structure at a more abstract level for efficiency.

In large-scale SLAM, a popular representation is to use submaps that are metrically consistent, and connect them with metric constraints [5], [6], [7], [8], [9]. Typically these methods use local reference frames for each submap; the robot is localized within the local coordinate system, and gets re-localized when switching submaps. Our system borrows from this idea: our submaps consist of locally-constructed occupancy grids for navigation and localization. However, we differ in that the pose graph is a global structure, which can be updated efficiently with recent sparse nonlinear methods [4].

Topological mapping constructs maps that navigate between places that can be recognized perceptually, as in Kuiper's classic work on the Spatial Semantic Hierarchy [10]. Typically the navigation is done by behavioral methods using simple perceptual clues such as wall-following [11], [12], [13] or, more recently, geometric image matching [14]. We also maintain a topological network among places, but these places are not perceptually significant – they are just places from which the robot takes a laser scan. Navigation and localization use scan-matching techniques from SLAM, rather than behavioral schemas.

The closest parallel to our work are hybrid systems that use local metric maps for localization and planning [15], [16]. In Thrun's work [15], the major emphasis is on making planning efficient by abstracting a topological map from an underlying metric map. Thus, he first builds a global occupancy map, and then forms a roadmap network for planning, using Voronoi diagrams. He shows that planning on the topological map leads to path lengths only a few percent greater than the grid-based paths. In our work, in contrast, the global grid map is never constructed: the nav graph is created from the SLAM pose graph nodes. Also, while the global plan is graph-based, local optimization is carried out in each submap, leading to the same path lengths as with the global grid.

Zivkovic et al. [16] present a similar system, but based on a graph-cut clustering model rather than Voronoi diagrams. Their results show higher computation times and path lengths compared to [15].

III. Map representation

A. The pose graph

Our navigation system assumes input from a graph SLAM algorithm that generates a hybrid metric-topological map. The underlying map representation is a pose graph, defined to as $\mathcal{G} = \{N, E, S, C, P\}$, where:

- N is a set of nodes. The nodes are referred to using unique ids that are stable over time, i.e., a given id always refers to one particular pose in the world.
- For each node n, S_n is the associated sensor data, stored in the coordinate frame of that node's pose. For the experiments, we use 2D scans from a laser range finder.
- E is a set of edges, representing soft constraints between node poses.
- For each edge e, C_e is the associated constraint, in the form (μ, Σ^{-1}), representing a Gaussian over the transformation between the two frames.
- P consists of the globally optimized pose p_n of each node.

We also assume the robot is localized in this graph. The localization is of the form (n, p), where $n \in N$, and $p \in SE(2)$ is a pose in the frame at n. Thus, unlike typical SLAM systems, we do not make use of a global pose.

Although it is not the focus of this paper, we now describe the specific topological SLAM algorithm that we use (see [17]. The algorithm maintains a pose graph and a localization with respect to it. The localization is updated using scan matching: given a previous localization, a relative pose based on wheel odometry, and a new laser scan, we scan match against the local neighborhood in the graph, find a new best pose, and then update the reference node to be the one closest to the new pose. Whenever the localization (n, p) is such that $\|p\|$ is greater than some threshold, we add a new node to the graph.

In our system, there are three kinds of constraints that may be added when a new node is generated (the optimization procedure is, of course, independent of the particular type of constraints used, and other sources such as GPS and visual odometry could be incorporated straightforwardly):

- A wheel odometry constraint linking this node to the previous one.
- Scan-matching constraints linking this node to the set of nearby nodes to the reference node, with respect to graph distance.
- Loop-closure constraints, based on a multiresolution scan match, linking this node to others that are not close in the graph but have optimized poses that are close (where the distance is between barycenters of the corresponding laser scans) in metric distance.

All scan matching is done using the open-source matcher that is included with the Karto SLAM package [18]. We run a graph optimization procedure (sparse pose adjustment [4]) periodically on the constraint graph; for even very large

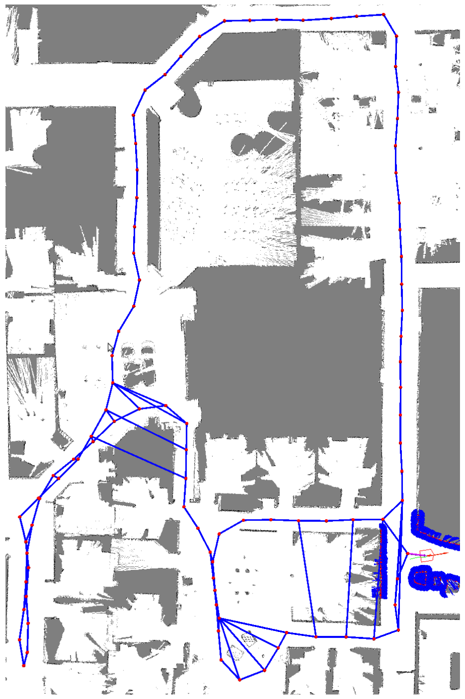
Fig. 2. Visualization of the pose graph generated by our graph SLAM algorithm on a simulated environment. The circles represent nodes in the graph, with edges representing constraints between them. The ground truth position of the robot is indicated by the pentagon, and the arrow to it represents the localization.

graphs, it typically takes only 10s of millseconds, and we run it once a second. Figure 2 shows the pose graph generated online by the SLAM system during a simulated run, and a corresponding localization of the robot.

B. The navigation graph

The pose graph represents constraints on relative node poses but does not directly represent navigability. Our first contribution is to overlay a *navigation graph* over the pose graph. A navigation graph is defined to be of the form $\mathcal{R} = \{\mathcal{G}, O, E, C, S\}$ where:

- $\mathcal{G}$ is a pose graph.
- O is a set of *local occupancy grids*. Each local grid is attached rigidly to some central node.
- For each grid o, $C_o \subset N$ is the set of *contained nodes* of o. The relative optimized poses of nodes in C_o in the grid's frame must actually lie within the bounds of the grid.
- For each grid o, $S_o \subset N$ is the set of *overlaying nodes* of o. o is computed by combining the sensor data of the corresponding nodes (using the relative optimized poses to register them).
- E is a set of edges. Each edge is of the form (n_1, n_2, o, c) where n_1 and n_2 are nodes, o is a grid containing both of them, and c is a numerical cost.

In our construction procedure, S_o consists of the contained nodes C_o, as well as other nodes whose scan barycenters lie within o (this is to minimize the unobserved portion of the

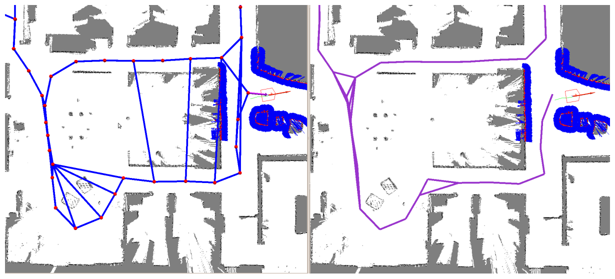
Fig. 3. Visualization of nav graph generated online while running in the same simulated environment as in Figure 2. The left image shows the pose graph for the relevant section of the environment. The right image shows the navigation graph where purple links represent edges. Notice that navigation graph edges represent connectivity information in the grid, while pose graph edges represent SLAM constraints, resulting in different graph structures.

grid in the case where all poses in the grid are facing the same way). The grid is computed by a standard ray tracing procedure [19]. For a given cell, let m be the number of rays passing through it, and n be the number of rays terminating at it. The cell is marked free iff $m > 2$ and $n/m < 0.1$. The grid size is a parameter to the algorithm, and we evaluate the performance at various grid sizes in Section VI. A square grid of side r is defined to *cover* a node if the node is contained within a square with side αr with the same center (we use $\alpha = 0.6$). The construction algorithm maintains the invariant that every node is covered by at least one grid by adding new grids when there exist uncovered nodes.

Each edge of the navigation graph is associated with one of the occupancy grids. An edge represents that it is possible to navigate from the source to the destination pose on the corresponding grid. The edge is given a weight equal to the cost of this path (we just use path length). Note that navigability, and therefore the structure of the navigation graph, depends on the shape of the particular robot.

Our construction procedure uses A* search within the grid (with obstacles inflated by the radius of the robot) to compute edges between each pair of nodes within a certain distance threshold of each other. Figure 3 shows the navigation graph generated during a simulated run.

The occupancy grids will be used for navigation, as described in Section IV. To make this work, we publish the local grid whose center is closest to the robot's pose. We also publish the robot's pose in that grid's coordinate frame (this is inferred by transforming the node-relative localization estimate received from SLAM).

IV. Navigation planning and execution

Navigation consists of forming a plan to get from point A to point B in the map, and then following it while avoiding unmapped obstacles. The basic form of a plan in our hybrid maps is:

- Select a start node near the robot, and plan a metric path to it.
- Select a goal node near the goal, and plan a metric path to it.
- Plan a topological path between the start and goal nodes.

A. Selecting Start and Goal Nodes

To project the robot's position onto the navigation graph, we require a local grid that contains the robot. We choose a grid whose center point has the minimum distance to the current position of the robot. Note that this search is carried out using the manifold distances along the pose graph.

Once a grid is selected, we find the closest navigation graph node to the robot by computing the configuration space for the grid assuming a circular robot, and planning metrically from the robot to each navigation graph node in the grid. The metric plan is efficiently formulated by computing a navigation function for the grid using Dijkstra's algorithm: each point in the grid has a potential associated with it that represents its distance to the robot as well as to associated obstacles. We then look up the potential for each navigation graph node contained in the grid and pick the one with the lowest potential as the starting node for planning in the navigation graph structure.

To select the goal node on the navigation graph for planning, we use a similar process with the desired goal pose. We find grids that contain the goal pose, select the grid whose midpoint is closest to the goal pose, compute configuration space and potential for the grid, and select the navigation graph node in the grid with the minimum potential.

B. Topological Planning

We create a plan between the start and goal nodes in the navigation graph using Dijkstra's algorithm. The plan created consists of a set of waypoints for navigation to follow. Each waypoint in the plan could be fed directly to the metric navigation system, but this would result in metric navigation closely following the navigation graph structure. This is not desirable since the navigation graph only covers some fraction of the space, and following it may lead to paths that are far from optimal. Instead, we'd like to plan metrically in the grid that currently contains the robot for as long as possible.

To this end, not every waypoint in the plan produced from the navigation graph is fed to the metric navigation system. Instead, we choose the last waypoint on the plan contained within the current grid for which a valid plan exists and pass it to the metric navigation system. Each time the robot switches grids, meaning it becomes closer to the center point of a new grid than the grid it is on, we again find the last waypoint in the plan contained on the grid and pass it to the metric navigation system. In this way, we allow the metric navigation system to be as efficient as possible for the grid size selected. Larger grid sizes lead to closer to optimal plans, smaller grid sizes require less memory. An example of the metric map being used to the robot's advantage is shown in Figure 4.

C. Metric Navigation

For performing navigation within a local grid, we use the ROS navigation stack [20]. The navigation stack takes in information from the robot's sensors, which it merges with the static obstacle information available in the local grid. This

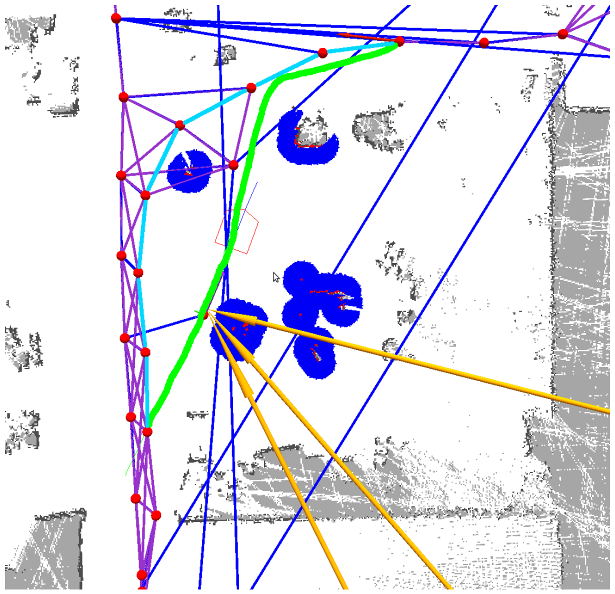

Fig. 4. An example of the robot creating a plan in its local metric grid that shortcuts the global navigation graph. The topological plan is shown in cyan while the metric plan is shown in green.

obstacle information is tracked in a three-dimensional voxel grid structure, and is projected down into two dimensions for path planning purposes. The navigation stack, as configured for this work, uses an A* planning algorithm that plans in configuration space. It then follows that plan using the Dynamic Window Approach [21] to forward-simulate potential trajectories and select commands that move along the global plan while avoiding obstacles.

D. Blocked Edges

The navigation system also needs to be robust in the presence of dynamic obstacles. It may be that a waypoint given to the metric navigation component of the system is infeasible because an unmodeled obstacle blocks the robot. In this case, the metric navigation component reports the goal as infeasible to the topological navigation component. From here, the topological navigation component needs to decide what edge in the navigation graph it should consider blocked. To do this, it requests metric navigation to make a plan from each waypoint in the current topological plan to its neighboring waypoint until it finds a link that the metric planner reports as not traversable. This link is added to a list of blocked links, and the planner is invoked to find alternate routes.

V. Updates to the Map

Although the purpose of this paper is to present the navigational system rather than the mapper, we briefly explain some simple procedures that show the promise of our technique as a basis for online map updating.

A. Incorporating pose graph updates

To ensure that our system can be run online, it is important to be able to update the navigation graph efficiently given changes to the pose graph. In our implementation, the pose graph mapper broadcasts incremental changes to the pose graph consisting of:

- New nodes
- New edges
- New scans attached to existing nodes
- Modifications to node poses

Given such a change, an exact procedure to update the navigation graph is to reoverlay every occupancy grid for which a node or scan has been added, or for which any node used to overlay it has shifted. We then recompute the shortest paths in each modified grid, and use these to add and delete edges and update edge lengths in the navigation graph.

In practice, such a procedure scales badly. The reason is that local changes can ripple through the entire pose graph during optimization, requiring overlaying and recomputing shortest paths on essentially every grid. Since we are incorporating changes every few seconds, this becomes infeasible for graphs with more than a few hundred nodes. We therefore adopt an opportunistic strategy: only those grids in the immediate graph neighborhood of the change are recomputed. Other grids are re-computed on demand, as the robot traverses into them when executing plans.

The effect of our strategy is that upon, say, a loop closure, we may temporarily be wrong about the connectivity of some distant portion of the environment. As we approach that area, though, we will update that part of the graph, correcting any out-of-date edges. An alternative would be to search from the point of loop closure outwards, until the effects of the closure on the *relative* poses of graph nodes is small. Given the results from relative bundle adjustment [22], we suspect that the number of affected local maps would be small.

B. Updates based on blocked paths

The map may not be static, so that the robot discovers an obstacle that blocks the connectivity of the nav graph. The general problem of deciding whether such an obstacle is transient or not is a difficult one. One simple method is to assume that all such blockages are temporary, and will be removed at some future time. We have already discussed how we identify blocked edges. Once a blockage is found, the relevant edge is added to a list of blocked links with a user-specified timeout. Each time the topological navigation component receives a new navigation goal to execute, it removes all edges between nodes pairs in the blocked link list. So, when a new plan is made, blocked links are not considered and the robot can receive a high level plan that avoids the obstacle. Blocked links are also removed from this list, and the appropriate edges restored to the roadmap, whenever they've persisted in the for longer than their specified timeout. This strategy is obviously sub-optimal, and is especially bad if a blockage is permanent; but in practice it has worked well.

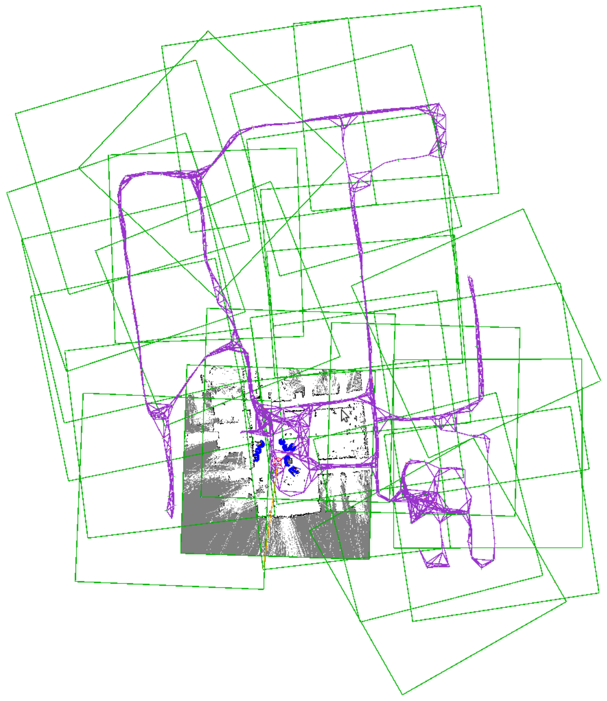
Fig. 5. The navigation graph generated for the simulation environment in which experiments were performed. The green boxes represent the local grids stored in the graph. The grid with obstacle information displayed is the active grid for the metric navigation system.

VI. Empirical results

To test our system we made use of both simulated and real robots with their associated envionments. In simulation, we compared different configurations of the system to attempt to learn what configuration would be promising for running on the real robot. Specifically, we performed three experiments. The first examines how the selection of a local grid size affects the optimality of plans in static environments. The second looks at how grid size selection influences plans in environments with dynamic and unexpected obstacles. The third comapres the performance differences between planning globally with a metric planner and planning globally on a graph structure. After performing these experiments in simulation, we re-ran them on our PR2 robot (`http://www.willowgarage.com/pages/pr2/overview`) to validate the results and prove the system in the real world.

A. Optimality

We first address the question of how good the plans from our navigation system are, by comparing to a metric planner that uses the ground truth occupancy grid of the full environment. For the purposes of this experiment, we define an optimal plan as that executed by the ROS navigation stack which uses a fully metric map [20]. From a simulated office environment, we built a navigation graph as shown in Figure 5 configured to allow for edges between nodes up to 3 meters apart. We then selected 20 locations on the navigation graph as goal points for the robot and planned to them using local grid sizes of 10, 15, and 20 meters. We

Simulation Results for Optimality	
Grid Size	Distance Traveled
Metric Map	723.15 meters
20 Meter Grid	721.16 meters
15 Meter Grid	713.35 meters
10 Meter Grid	715.36 meters
Exact Following of Graph	727.51 meters

TABLE I

COMPARISON OF DISTANCE TRAVELED FOR DIFFERENT LOCAL GRID SIZES WITH DISTANCE TRAVELED WITH A METRIC SYSTEM FOR A SIMULATED ROBOT.

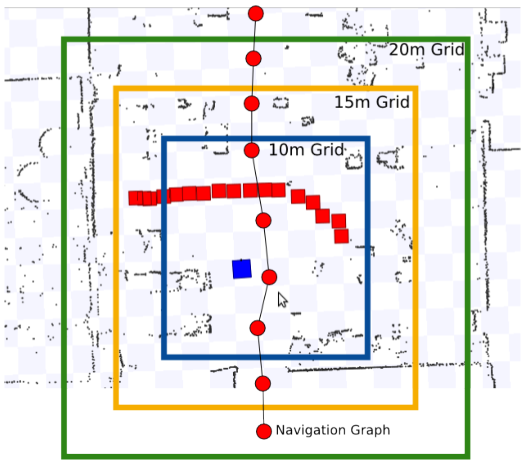

Fig. 6. An example of a sparse navigation graph with dynamic obstacles. The 10 meter grid doesn't have enough metric information to plan around the obstacles, the 15 meter grid has barely enough room in its map to create a plan, and the 20 meter grid has plenty of space to plan around the obstacle.

hypothesized that as grid size increased, plans executed with our system would become increasingly optimal at the cost of performance. However, as shown in Table I, each grid size compared favorably with the optimal plan over the course of the run as they were all within 1 percent of the optimal path length.

To explore this further, we attempted to run with a grid size of 5 meters. This size proved to be too small to successfully reach all of the waypoints visited, but yielded an interesting observation. The optimality of a plan in the navigation graph seemed more dependent on the density of the navigation graph than the local grid size. As long as the grid size was large enough to follow waypoints in the plan, coverage of the space was sufficient to produce a near optimal plan. To verify this, we also ran the experiment on a 20 meter grid where the robot was required to follow the plan given by the navigation graph exactly, instead of being given an exit point from each region. Even when required to reach every waypoint on the plan, the total distance traveled by the robot compared favorably with the optimal path length as shown at the bottom of Table I. This implies that the navigation graph is dense enough in the environment tested to produce near optimal plans regardless of grid size.

Graph vs Metric Planning			
Path Distance (meters)		Planning Time (seconds)	
Metric	Graph	Metric	Graph
33.95	33.69	0.12521	0.01776
24.79	25.83	0.06862	0.01649
24.78	25.52	0.07562	0.02044
25.23	24.86	0.09117	0.02060
29.97	31.80	0.08798	0.01520
17.44	17.55	0.06472	0.01693
46.64	38.12	0.14870	0.01194
35.12	37.55	0.13294	0.01323
55.53	58.61	0.14245	0.00010
65.17	67.77	0.16414	0.02216
44.42	45.11	0.13104	0.01674
18.43	19.22	0.07238	0.00012
36.38	36.52	0.09612	0.00019
50.97	49.39	0.14645	0.00016
34.12	35.25	0.06304	0.00020
40.25	39.99	0.14480	0.01475
2.89	2.94	0.03589	0.02489

TABLE II

COMPARISON OF RUNTIMES AND PATH DISTANCES FOR THE GRAPH AND METRIC PLANNERS.

B. *Graph Density vs Grid Size*

Although the size of local grids had little effect on plan optimality in our first experiment, we postulated that with a sparse graph, or a situation where the robot was required to move far off the navigation graph structure to match the metric plan, grid size would still be a major factor. To test this hypothesis, we designed an example where the robot encountered an obstacle in a large room of our simulated environment. As shown in Figure 6, the navigation graph only runs down the center of the space between two doorways. In effect, the navigation graph is extremely sparse in this section of the environment, and the dynamic obstacle blocking the main path through the room forces the robot off of the graph structure. Figure 6 clearly shows that for the 10 meter grid, the obstacle will be impassible, while the 15 and 20 meter grids are large enough to allow the robot to avoid the blockage.

This example implies that there is a direct relationship between navigation graph node density and the size of local grids. A dense graph reduces the size of local grids necessary to successfully navigate in an environment, but holds a higher maintenance cost as connectivity must be computed between many nodes. A large local grid size makes computing the edges in the navigation graph less computationally expensive, but with this comes increased metric planning cost.

C. *Metric vs Graph Planner Comparison*

To compare the efficiency of running planning on a metric structure versus planning on a graph, we ran an experiment that compared runtimes of the metric planner with that of the graph planner in an office environment. For the experiment, we compared 17 paths of varying distance. The metric planner used a 58 meter by 45 meter grid at a resolution of 0.025 meters/cell for planning, while the graph planner operated on a navigation graph with 374 nodes and a local grid size of 10

Real World Results for Optimality	
Grid Size	Distance Traveled
Metric Map	62.77 meters
10 Meter Grid	74.90 meters

TABLE III

COMPARISON OF DISTANCE TRAVELED FOR DIFFERENT LOCAL GRID SIZES WITH DISTANCE TRAVELED WITH A METRIC SYSTEM FOR A REAL ROBOT.

meters at a resolution of 0.025 meters/cell. As shown in Table II, the graph planner is significantly more efficient, and the gap between the two scales based on distance planned. This implies that in larger environments, the disparity between the graph and metric planners would become even greater. Table II also shows the distances covered by the robot when using the metric and graph planners are nearly identical across a number of different path lengths. There is one case, however, where the graph planner takes a significantly shorter route than the metric planner. On this particular plan, the metric planner chose to go the long way around an obstacle to minimize its cost function, while the graph planner chose to traverse through a narrow passage.

D. Real World Trials

To verify that the navigation system works in the real world, we ran a smaller version of our first experiment on a PR2 robot. In this experiment, the robot was tasked to navigate to 5 waypoints in our office environment using both the hybrid system presented in this paper and a global metric grid approach. As shown in Table III, the robot created plans comparable to those of the metric planner despite using a small, 10 meter grid. During these experiments, the robot also encountered dynamic obstacles that caused edges in the navigation graph to be blocked, but it was able to create plans in the roadmap around obstacles using alternate links.

VII. CONCLUSION

We have described a hybrid metric-topological navigational system that achieves a performance similar to that from a global metric map. The hybrid system achieves its performance by using local metric grids for enhanced local planning, while avoiding the computation of a complete global grid. As a consequence, the hybrid system can be created and updated incrementally.

While we have shown some simple algorithms for map update, there is much room for improvement. For example, when updating the map, it would be useful to show that relative displacements among graph nodes die out as the graph distance from the update increases, so that only local updates are ever necessary.

Another interesting extension would be to maintain the local metric maps as full 3D structures. Since the area of the each local grid is limited, it should be possible to do the construction online, and to cache them when they are no longer immediately needed.

The mapping and navigation system we have presented is available as open-source software as part of the ROS system (`http://ros.org`).

REFERENCES

[1] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, pp. 333–349, 1997.

[2] S. Thrun and M. Montemerlo, "The GraphSLAM algorithm with applications to large-scale mapping of urban structures," *International Journal on Robotics Research*, vol. 25, no. 5/6, pp. 403–430, 2005.

[3] A. Howard, G. Sukhatme, and M. Mataric, "Multirobot simultaneous localization and mapping using manifold representations," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1360–1369, July 2006.

[4] K. Konolige, G. Grisetti, R. Kümmerle, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," in *In IEEE International Conference on Intelligent Robots and Systems*, 2010.

[5] M. Bosse, P. Newman, J. Leonard, and S. Teller, "Simultaneous localization and map building in large-scale cyclic environments using the atlas framework," *International Journal of Robotics Research*, vol. 23, no. 12, 2004.

[6] C. Estrada, J. Neira, and J. Tardos, "Hierarchical slam: Real-time accurate mapping of large environments," *IEEE Transactions on Robotics*, vol. 21, no. 4, 2005.

[7] J. Modayil, P. Beeson, and B. Kuipers, "Using the topological skeleton for scalable global metrical map-building," in *International Conference on Intelligent Robots and Systems*, 2004.

[8] K. Ni, D. Steedly, and F. Dellaert, "Tectonic sam: exact, out-of-core, submap-based slam," in *In Proc. IEEE International Conference on Robotics and Automation*, 2007, pp. 1678–1685.

[9] L. Paz, J. Tardós, and J. Neira, "Divide and conquer: EKF SLAM in O(n)," *IEEE Transactions on Robotics*, vol. 24, no. 5, October 2008.

[10] B. Kuipers and Y.-T. Byun, "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations," *JOURNAL OF ROBOTICS AND AUTONOMOUS SYSTEMS*, vol. 8, pp. 47–63, 1991.

[11] E. Fabrizi and A. Saffiotti, "Augmenting topology-based maps with geometric information," *Robotics and Autonomous Systems*, vol. 40, no. 2, 2002.

[12] D. Rawlinson and R. Jarvis, "Topologically-directed navigation," *Robotica*, vol. 26, no. 2, pp. 189–203, 2008.

[13] P. Beeson, N. K. Jong, and B. Kuipers, "Towards autonomous topological place detection using the extended voronoi graph," in *In IEEE International Conference on Robotics and Automation*, 2005, pp. 4373–4379.

[14] F. Fraundorfer, C. Engels, and D. Nistér, "Topological mapping, localization and navigation using image collections," in *IROS*, 2007, pp. 3872–3877.

[15] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998.

[16] Z. Zivkovic, B. Bakker, and B. J. A. Kröse, "Hierarchical map building and planning based on graph partitioning," in *ICRA*, 2006, pp. 803–809.

[17] J. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," in *Proc. IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, Monterey, California, November 1999, pp. 318–325.

[18] Karto Robotics, http://www.kartorobotics.com.

[19] J. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems J.*, vol. 4, no. 1, pp. 25–30, 1965.

[20] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *ICRA*, 2010, pp. 300–307.

[21] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[22] G. Sibley, C. Mei, I. Reid, and P. Newman, "Vast-scale outdoor navigation using adaptive relative bundle adjustment," *International Journal of Robotics Research*, vol. 29, no. 8, 2010.