Graph-based Planning Using Local Information for Unknown Outdoor Environments

Jinhan Lee, Roozbeh Mottaghi, Charles Pippin and Tucker Balch
{jinhlee, roozbehm, cepippin, tucker}@cc.gatech.edu
Center for Robotics and Intelligence Machines
Georgia Institute of Technology, Atlanta, GA 30332, USA

***Abstract*— One of the common applications for outdoor robots is to follow a path in large scale unknown environments. This task is challenging due to the intensive memory requirements to represent the map, uncertainties in the location estimate of the robot and unknown terrain type and obstacles on the way to the goal. We develop a novel graph-based path planner that is based on only local perceptual information to plan a path in such environments.**

In order to extend the capabilities of the graph representation, we introduce Exploration Bias, which is a node attribute that can implicitly encode obstacle features at immediate surrounding of a node in the graph, the uncertainty of the planner about a node location and also the frequency of visiting a location. Through simulation experiments, we demonstrate that the resulting path cost and distance that the robot traverses to reach the goal location is not significantly different from those of the previous approaches.

I. INTRODUCTION

In many real world, outdoor settings, the environment contains a variety of features, which make the task of robot navigation more difficult in some parts of the environment. If we want to store the global information about the traversability and the observed features of the environment, we will run out of memory as the map that contains this information grows rapidly due to the large number of features and variations in the environment. Furthermore, in field robotics applications, we usually require that the robot traverse a large area, which requires us to store a large map of the region and the map grows as the working area of the robot becomes larger and also choosing the right size for the map resolution depends on many factors in the environment. Therefore, a sparse representation of the environment will reduce the amount of the required memory.

A second challenge to tackle in the navigation of outdoor robots is that in many cases, the robot should traverse an unknown area and we cannot obtain the global map beforehand. Sometimes, even acquiring and building a detailed global map on the fly is not feasible because of the various sources of noise that we have in the environment. Therefore, a method that can solve the navigation problem with local information is desirable.

We usually face the problem of finding a path between two distinct geographic locations for outdoor robots. There is always a trade-off between the cost of the traversed path and the time spent to reach a goal from a source location, so finding an optimal strategy to reach the goal is not a trivial task. Another factor that should be considered is the planning time, which is the computation time that we require to find the next waypoint location. In addition, having partial knowledge of the environment and using only local perception, will make the problems mentioned above even more difficult. In this paper, we present a novel method for path planning for outdoor robots in unknown environments. Motivated by methods like [1] and [2], we use a graph-based path planner to satisfy the sparsity requirement as opposed to a grid-based planner. We develop a heuristic to plan a path through unknown areas using only local information and we show the strength of our method compared to some of the existing planners.

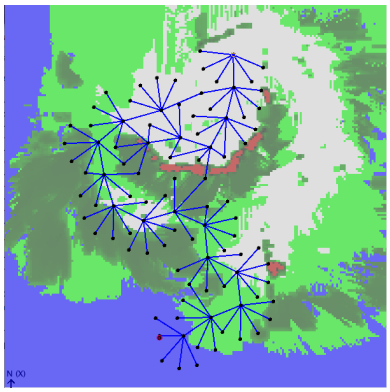

Fig. 1. The nodes and edges of an example graph have been shown on a cost map. The cost map reflects obstacle and traversability information in each cell. White and lighter green represent path and low cost areas. Darker green represents traversable but high cost areas. Red represents obstacles or walls. Unknown (unobserved) areas are shown in blue.

Our method has advantages over the previously developed methods for path planning of outdoor robots. We will mention a few related works that motivate our work since describing all categories of the planners is impossible. Latombe [3] represents a basic Wavefront planner. This planner requires the full map of the environment but it is very common that the outdoor robots should operate in an unknown environment. [4], [5] and [6] represent three types of grid-based planners. To use the grid-based (or graph of connected grid cells) methods we should deal with the trade-off between the accuracy of representation and the amount of required memory space. Choosing the resolution of the grid

becomes an important issue, especially when a robot needs to navigate in large scale environments.

To overcome the limitations of the grid-based planners, some authors have developed path planning methods for outdoor robots based on Visibility Graphs [7][8]. These methods usually plan a path through a map in which, the obstacles have been transformed into polygons. Planning using these methods in high-dimension spaces is not computationally efficient and another drawback of these methods is that sometimes we close off good paths when we polygonize the environment.

Our heuristic for planning a path on the graph is mainly based on *Exploration Bias*, which is an adaptive attribute defined for each node in the planning graph. The idea for introducing this term is to avoid certain regions of the environment. For example, it is used to avoid the local minima that we encounter in a cul-de-sac and to avoid the obstacles in general. Furthermore, this factor can be used to probabilistically plan a path when we have large uncertainty in the location estimation of the robot. In the following sections, we explain how a planning graph is built and how we incorporate the Exploration Bias into the nodes of a graph. Then we compare our method with a few existing methods that have remarkable results to demonstrate that there is no significant difference in the resulting path cost, planning time and the traveling time of the robot.

II. Sensor Estimated Graph-based Planning

In this section, we describe how we build a graph based on local perception of the robot and how we plan a path using the planning graph. We begin this section by defining the planing graph structure and the newly introduced variables associated with the graph.

A. Definition of Graph Structure

As the basic graphs, our planning graph $G_p = (V, E)$ consists of a set of nodes and edges. Each node $\nu \in V$ represents a geometric location in the environment so a location parameter is associated with each node in the graph. Throughout this paper, we use two dimensional location estimates but it can be easily extended to higher dimensions. Another attribute associated with a node is the *Exploration Bias*. The Exploration Bias is considered as a cost for the node and represents how good a node is to be chosen as the next waypoint. We incorporate this cost in our planning process, as discussed later in Section II-C.

We define an edge between two nodes whenever a node is visible and traversable from the other node. The visibility is determined based on local perception information. The perception information also determines the cost of an edge that is how traversable between two nodes is. Higher traversability corresponds to a lower cost for an edge. An example graph is shown in Figure 1.

In the following sections, we explain how the graph is constructed with local perception information and then we show how one of the nodes of the planning graph is chosen as an intermediate (interim) goal at each planning cycle.

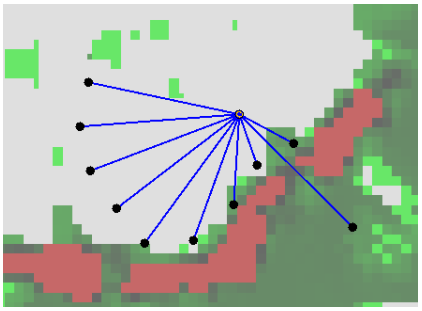

Fig. 2. An example of traversable rays from the robot overlaid on a cost map has been shown. Some rays are fully extended and others are pre-stopped due to collision with obstacles.

B. Graph Construction Procedure

For the graph construction, it is assumed that a robot does not have any prior knowledge of the working environment so the robot starts with a graph, which has a single node and no edges, and we assume the robot always have some knowledge about the location of the goal and the location of itself in a global reference frame.

The procedure for constructing the graph is as follows:

- The robot receives local perception information. In this paper, by local information we mean the observation of the robot from its immediate surrounding. In general, the perception information can be any useful information for planning for example, the cost map, elevation map or traversability map.
- In our case, we shoot out traversability rays in the visible range on the cost map. The rays are in different angles from the location of the robot. Figure 2 shows some example rays on a cost map. An edge whose one end is the robot location and the other end is the maximum distance that the robot can observe, will be added to the graph.
- To assign a cost to each edge, we sample each edge and calculate the average of sample costs on the edge. The cost of the edge will be determined by multiplying the average cost value to the length of the edge.
- We assign Exploration Bias, EB, to a newly added node ν as shown in Equation 1. We assign zero Exploration Bias to a node unless the edge that connects the node to the robot collides with an obstacle.

$$EB(\nu) = \begin{cases} k & \text{if } \nu \text{ is close to obstacle} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The value for k can be chosen depending on how much we want to get close to obstacles during a run. We can avoid planning a path through dangerous nodes by increasing k value. We also increase the Exploration Bias for a node, whenever a robot visits that node. By visiting a node we mean that the node is within a certain distance from the robot. This means that a plan is very unlikely to succeed when it contains many nodes with large Exploration Bias.

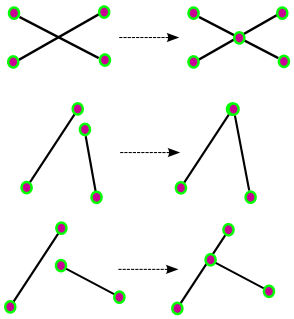

Fig. 3. Some example cases of merging a new edge with the existing edges.

- Finally, there is a chance that the new edges intersect with some of the existing edges. Some examples are shown in Figure 3. We merge the new edges with the existing edges and we prune the graph by removing the extra nodes and edges and we make the graph a connected graph by adding nodes. When we merge two nodes, the nodes will share the Exploration Bias and when we add a new node, the new node will have zero Exploration Bias.
  Benefits of merging graphs are of significant importance. One benefit is the reduction of the computations for planning. This is because merging usually reduces the path length to a node in the graph, which is a dominant factor determining the computational complexity of planning. Another words, the quality of the shortest path obtained from the constructed graph after finishing a run will be improved. Without merging, the shortest path in the worst case will be sum of distances the robot moves over a run.

C. Planning on Graphs

At the beginning of a run, the robot starts with a graph that consists a single node that represents the area of the current location of the robot. Once new traversable edges are added to the graph, one of the nodes in the graph has to be chosen as a sub-goal, which we denote as *interim goal*. The interim goal is more than just a waypoint because the robot has to reach it before performing re-planning to find a new interim goal or the goal. We put this constraint based on our preliminary observation from our tests with a real robot that too frequent re-planning often causes the robot to oscillate between different but equally low cost areas and prevents the robot from exploring areas thoroughly.

Selection of an interim goal is based on a heuristic function described below. We form a set of nodes, denoted by $\hat{V}$, in the graph that have the minimum Exploration Bias and were not generated by collision with obstacles. If all nodes in the graph would be considered as candidates for an interim goal, the method became more computationally expensive. Also, we do not observe any improvement in the final results when we consider all of the nodes. Then, the cost function $c(\nu)$ for each node $\nu \in \hat{V}$ is calculated as in Equation 2. The node with the minimum cost will be chosen from this set as the next interim goal.

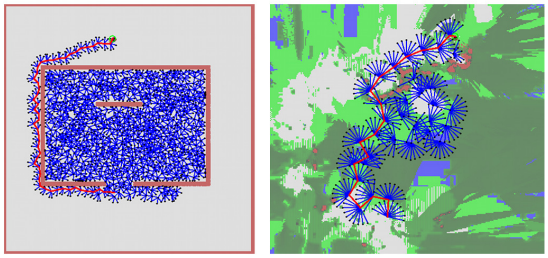

Fig. 4. Examples of planned paths for two different maps are shown. Left: a uniform synthetic cost map is shown as an example case of a cul-de-sac. Right: a non-uniform cost map represents cluttered areas with high costs and obstacles near the goal and the robot. The red line shows the final best path from the source to the goal found after finishing a run.

$$c(\nu) = eb(\nu) \times (\alpha\, p_A(\nu) + \beta\, p_B(\nu)) \tag{2}$$

where, $p_A(\nu)$ is the cost of the path between the current location of the robot to the node ν, $p_B(\nu)$ represents the cost from node ν to the goal. α and β are the normalization parameters and we show the effect of these two parameters in the experiments later.

We denote Exploration Bias of an area around node ν by $eb(\nu)$. We calculate this quantity for a node based on a normal weighted average of the neighbor nodes' Exploration Bias, where the distance of a node from its neighbors is defined on the graph. The idea is that we assign a high cost to a node which is located in an area with high Exploration Bias and we try to avoid that region. $eb(\nu)$ is defined as follows:

$$eb(\nu) = \frac{\sum_{\substack{i=1 \\ i \neq \nu}}^{N} EB_i \times exp(-\frac{1}{2} \left\| d(\nu, i) \right\|_{\Sigma}^2)}{\sum_{\substack{i=1 \\ i \neq \nu}}^{N} exp(-\frac{1}{2} \left\| d(\nu, i) \right\|_{\Sigma}^2)} \tag{3}$$

where, N is the number of neighbors of ν and $\left\| d(\nu, i) \right\|_{\Sigma}^2$ is the Mahalanobis graph distance between node ν and node i. EB_i is also the Exploration Bias for node i.

The next part of the problem is to plan a path from the current location of the robot to the candidate node. We can choose any method that finds the shortest path between two nodes in a graph. If the candidate nodes comprise a large portion of the graph, the individual calculations for each candidate node would be computationally expensive. We use Dijkstra's algorithm, which finds shortest path from one source node to every other nodes in the graph. Since the location of the robot is a common source node for each search, the single execution of the Dijkstra's algorithm keeps a shortest path for every node in the graph and acts as a lookup table.

The planned paths for two different maps are shown in Figure 4. We have shown the full map in the figure but a robot only visits and perceives approximately the contour of the constructed graph.

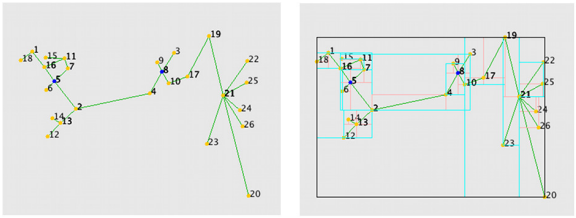

Fig. 5. An example of a planning graph $G_p(V, E)$ (left) and the corresponding R-Tree (right). Each edge in the graph is tagged with a Minimum Bounding Rectangle (MBR), which is the minimum size of a rectangle containing an edge.

D. Computational Complexity of Graph Operations

Several graph-related operations are used for both constructing graphs and planning over graphs and they are computationally expensive since they all require pairwise comparison of existing nodes and/or edges with a newly added node and/or edge. For instance, finding edges in a graph that intersect with a new edge takes $O(|E|)$.

As a robot traverses an environment and perceives more local information, computations of graph-based operations linearly increase. To make those computations almost constant regardless of the number of edges and nodes in a graph, we adopt a spatial tree-based data structure, the R-Tree, which was proposed by Guttman [9] and has been successfully used to search spatial objects stored in a spatial database that are inside a window query.

The R-Tree, as shown in Figure 5 (right) is a tree data structure, which splits the space with hierarchically structured minimum bounding rectangles (MBRs). A tree consists of two different parts: 1) leaf nodes and 2) internal nodes. Each leaf node contains a spatial object (an edge in this paper) and the object's MBR, two dimensional minimum size of rectangle box bounding the object. Each internal node defines a MBR bounding its children nodes (which could be either/both leaf nodes or/and other internal nodes). We cannot use data structures storing points such as KD-Tree or Quadtree because those data structures do not support searching for intersection between edges. In section III, we show through experiments that using the R-Tree for searching nodes and edges significantly reduces the overall planning time. Below, we briefly describe how the R-Tree is constructed and balanced while building a planning graph. More details about R-Trees can be found in [9].

- **Searching nodes in a graph within a certain distance from a query point.** To merge close nodes to simplify the graph, it is required to find nodes close to the query. A minimum bounding rectangle at which the query location is centered is determined and the size of the rectangle depends on the search radius. The search starts from the root node of the tree. Each child node belonging to the root is checked if the child node's bounding rectangle overlaps with the query's. If so, then that child node is also searched; otherwise, the node (and children nodes of the node) are excluded from the search. The search stops if all of the nodes in a tree are either traversed or excluded. The search retrieves all of edges whose bounding rectangles overlap with the query's. Final distance checking should be done with all nodes of retrieved edges to acquire close nodes to the query point.
- **Searching edges intersecting with a query edge.** Once a minimum bounding rectangle containing the query edge is determined, the same steps are taken for retrieving edges whose bounding rectangle overlaps with the query's. The intersection checking should be done with retrieved edges to acquire edges that intersect with the query edge.
- **Adding a new edge to the graph.** When a new edge is added to the graph, that edge as well as its minimum bounding rectangle should be added to the tree. Inexpensive search operations, taking $O(log|V|)$, described above are possible at the expense of additional computations for insertion and deletion operation. First, a minimum bounding rectangle containing the edge should be determined. Then, the edge tagged with its MBR is added to the tree by traversing from the root node of the tree and recursively selecting a child node of the traversing node, whose bounding rectangle needs least enlargement to include the new edge until reaching to a leaf node. Then, the edge is added to the selected leaf node.
- **Deleting an existing edge from a graph.** When a new edge intersects with existing edges in the graph, those existing edges in the graph are deleted from and splitted edges are added to the tree. To delete them from the tree, minimum bounding rectangles containing those edges are determined and we search for leaf nodes whose bounding rectangles are the same as the deleted edges' and remove those leaf nodes from the tree.

III. Experiments

We performed experiments in simulation using three sample cost maps: one map created from real outdoor runs of a mobile robot platform and the other two maps created manually to test some difficult situations. The robot-generated cost map, as shown in Figure 6 (right), represents an area containing various obstacles including trees, walls, and barrels and also open terrain. The cost map is obtained from the information from two pairs of stereo cameras and the bumpers of the robot. We call this cost map the *cluttered* cost map. The first manually generated cost map, as shown in Figure 4 (left) is a binary map where the the goal is on the outside of rectangular obstacles and the only way to reach to the goal is to get out of the room via an open area located at the bottom of the map. It is denoted by *room* cost map. The second manually generated cost map shown in Figure 6 (left) is denoted by *cul-de-sac* cost map. Each cost map represents a grid of $400m \times 400m$ and the terrain observed in *cluttered* cost map is approximately $100m \times 100m$.

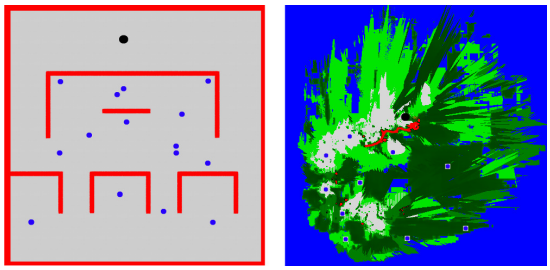

Fig. 6. (Left) *cul-de-sac*, a synthetic cost map which is a uniform cost map having three small cul-de-sacs at the bottom and a large one at the top. (Right) *cluttered*, a robot-built cost map which includes various types of low cost and high cost obstacles. The large black dot represents the location of the goal and the small blue dots correspond to start locations of different trials.

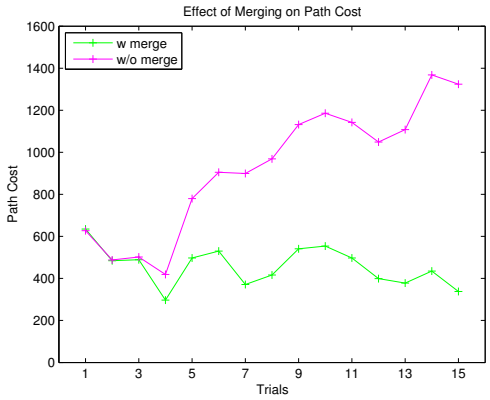

Fig. 7. The cost of the resulted shortest path per each trial on synthetic *cul-de-sac* cost map. The shortest path is found after the planning is finished. The green line shows a set of runs with using graph merging, and the purple line shows a set of runs without graph merging.

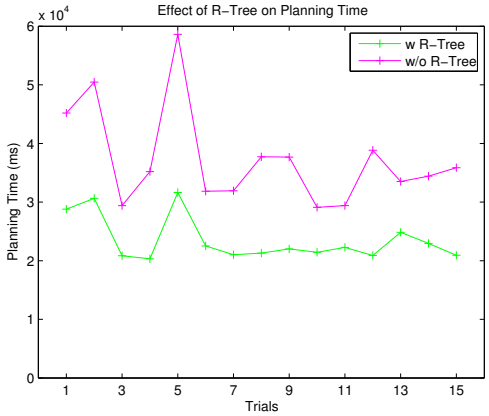

Fig. 8. Performance graph for planning time per each trial in *cul-de-sac* cost map. The green line indicates a set of runs with using the R-Tree, and the purple crosses correspond to the same starting locations but without using R-Tree.

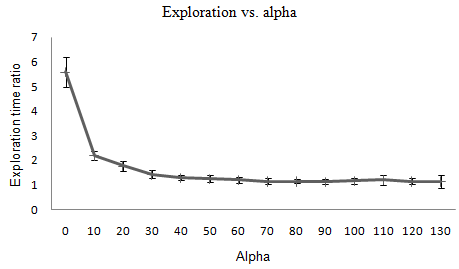

Fig. 9. Performance graph for exploration time ratio for different values of parameters $alpha$ and $beta$ in the synthetic *room* map. Different settings of $alpha$ with $beta$ fixed at 150 are tested. Vertical bars indicate the variance of the exploration time ratio.

A. Experimental Setup

Experiments were conducted in two different ways: 1) parameter experiments: the experiments with different settings for the parameters of our approach, and 2) method experiments: comparing our approach against two randomized sampling based methods, a variant of RRT [10] that basically samples from a set of configurations from previous solutions for fast and consistent convergence to a path to the goal, and another variant of RRT [11] that uses a cost-based distance function to incorporate cost information of terrains into planning process and dynamically adjusts the distance function to find the near optimal solution.

B. Parameter Experimental Results

In the first experiment, we tested the effectiveness of graph merging on the cost of the best path (the red paths shown in Figure 4) found after the planning is done in *cul-de-sac* cost map. As shown in Figure 7, graph merging reduces the path cost. Each trial in the figure corresponds to different starting location of the robot. It is obvious that without merging edges and nodes that cover spatially same areas, the resulting path is the same as the path that the robot traverses over a run. The path cost does not have a specific unit. We assign a value to each pixel of the map based on what we observe from the robot. For instance, tall grass has higher cost compared to short grass.

The second experiment involves the estimation of the computational complexity of graph-related operations (i.e. insertion, deletion, and search). It is shown in Figure 8 that using R-Tree for operations on planning graphs significantly reduces the computation and thus the overall planning time.

In the last experiment, shown in Figure 9, we consider exploration time ratio as a function of $alpha$ and $beta$ (refer to Eq. 2) used for calculating an interim goal on the $room$ cost map. Exploration time ratio is an average of running time ratio (ratio of the spent time to the minimum spent time). As the $alpha$ value becomes close to 0, the exploration time ratio almost goes up to 6. This is because smaller values of $alpha$ put more weight on p_B in Equation 2 and the location of new interim goal less depends on the location of the previous interim goal. Usually, as α becomes smaller, the robot spends less time for exploration of the immediate surrounding and it tries to reach the goal faster. However, in this experiment, the robot has been surrounded by a large obstacle, which forces the robot to explore farther areas as well. It should be mentioned that the robot starting location is different in each trial.

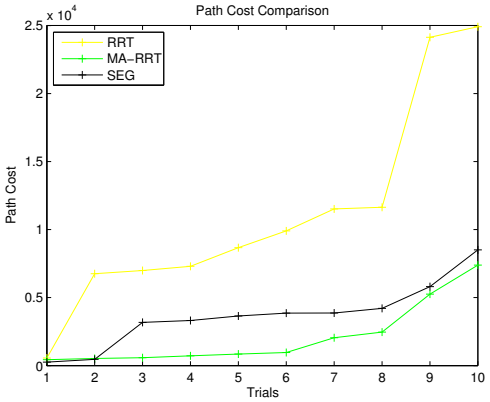

Fig. 10. Path cost for runs with different starting locations for *cluttered* cost map. The trials are sorted for each method separately.

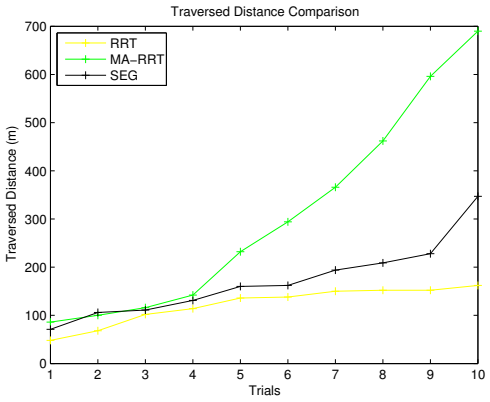

Fig. 11. Travel distance for runs with different starting locations for *cluttered* cost map. The trials are sorted for each method separately.

C. Comparison with Other Planning Methods

We compared our approach (Sensor Estimated Graph planner) with the basic RRT [10] and MA-RRT [11] to determine the relationship between the final path cost and the distance traveled by the robot using each method. We define the final path cost to be the cost of the path calculated on each model (grid maps for RRT and MA-RRT, and graphs for our approach) after finishing a run.

As shown in Figure 10, the cost of the paths found by SEG method is better than those from RRT but they are slightly worse than the solutions of MA-RRT. The MA-RRT method, as shown in Figure 11, spends more time on exploring the robot surrounding to find a better path compared to the other two methods.

Throughout this experiment, we demonstrate that the SEG algorithm is capable of finding a high quality path solution by traversing reasonable amount of areas.

IV. Conclusion and Future Work

We presented a graph-based path planning method which uses only local perception of the currently visible area around the robot to plan a path from a source location to a goal. We used this path planning method to explore unknown areas and find a low cost path to the goal. Apparently, the sparsity of the graph in contrast to the grid cells, improved the memory requirements of the existing planners while we show that resulting path cost and the traveled distance of the robot is not significantly different from those of the previous approaches.

We introduced Exploration Bias as a node parameter that determines how much the surrounding area of a node has been visited and how likely it is that the node be a candidate node for the next waypoint. One of the advantages of Exploration Bias is that a node implicitly encodes the neighbor obstacles and we do not need to save the obstacles in memory.

The low memory requirement is one of the main advantages of our method. In outdoor experiments, where the robot operates in large environments a sparse representation is required as even with the current memory technologies, the robot encounters memory problems in such large environments. Addition of other dimensions to a map representation such as height or path type may even increase the need for a sparse representation.

In some applications, we require that the robot repeats the same task in an environment. If we save the path and use the nodes and edges that formed the best path, we do not need to plan a path again. There are usually some uncertainties in the outdoor robot location estimates obtained using GPS. The graph representation is more robust against these uncertainties if we can go from a node to a neighbor node with a single control law since the location of these nodes will not be important any more and noise in location estimate will not have a drastic effect on path planning.

Although this method works in both indoor and outdoor environments it is more suitable for outdoor settings since the path cost is uniform in the indoors and also outdoor terrains have more open areas.

References

[1] D. Wooden and M. Egerstedt, "On Finding Globally Optimal Paths through Weighted Colored Graphs", *IEEE Conf. on Decision & Control*, 2006.

[2] M. Ryan, "Graph Decomposition for Efficient Multi-robot Path Planning", *Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, 2007.

[3] J.-C. Latombe, *Robot motion planning*, Kluwer Academic Publishers, Dordrecht, Netherlands, 1991.

[4] S. Koenig and M. Likhachev, "Fast Replanning for Navigation in Unknown Terrain", *IEEE Trans. on Robotics and Automation*, 21(3), pp. 354–363, 2005.

[5] X. Sun, S. Koenig, and W. Yeoh, "Generalized Adaptive A*", *Proc. of the Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.

[6] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* Search with Provable Bounds on Sub-Optimality", *Proc. of Conf. on Neural Information Processing Systems (NIPS)*, 2003.

[7] H.-P. Huang and S.-Y. Chung, "Dynamic Visibility Graph for Path Planning", *Proc. 2004 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2004.

[8] D. Wooden and M. Egerstedt, "Oriented Visibility Graphs: Low-Complexity Planning in Real-Time Environments", *Intl. Conf. on Robotics and Automation (ICRA)*, 2006.

[9] A. Guttman, "R-trees: A dynamic index structure for spatial searching", *Proc. of SIGMOD Intl. Conf. on Management of Data*, 1984.

[10] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation", *IEEE/RSJ Intl. Conf. on Robots and Systems*, 2002.

[11] J. Lee, C. Pippin, and T. Balch, "Cost Based Planning with RRT in Outdoor Environments", *IEEE/RSJ Intl. Conf. on Robots and Systems*, 2008.