2011 IEEE/RSJ International Conference on
Intelligent Robots and Systems
September 25-30, 2011. San Francisco, CA, USA

# Occupancy Grid Rasterization in Large Environments for Teams of Robots

Johannes Strom Edwin Olson

*Abstract*— **We introduce a method for efficiently rasterizing large occupancy grids. Efficient Maximum Likelihood Estimation (MLE) of robot trajectories has been shown to be highly scalable using sparse SLAM algorithms such as SqrtSAM, but unfortunately such approaches don't directly provide a rasterized grid map. We harness these existing SLAM methods to compute maximum likelihood (ML) robot trajectories and introduce a new efficient algorithm to rasterize a dynamic occupancy grid. We propose a spatially-aware data structure that enables the cost of a map update to be proportional to the impact of any loop closures, resulting in better average case performance than naive methods. Furthermore, we show how redundant sensor data can be exploited to improve map quality and speed up rasterization. We evaluate our method using several data sets collected using a team of 14 autonomous robots and show success in mixed indoor-outdoor urban environments as large as 220m x 170m, with 0.1m resolution.**

Fig. 1. Rasterized map for a portion of Keswick Barracks, Adelaide, South Australia (180m x 100m). The map was produced with a mean processing time of 1.3 seconds on a single 2.53 GHz core. Permanent structure is shown in black; known free space in gray; unobserved in white.

## I. Introduction

Successful autonomous operation of a team of robots in unknown large-scale environments depends on having a robust and globally consistent mapping solution. In particular, rasterized occupancy grids have many important applications where efficiency is an important objective (See Fig. 1). For example, autonomous exploration presents particularly hard timing requirements – the planner requires map updates quickly so that it can compute updated plans without requiring robots to stop and wait [12]. Standard approaches to rasterization in the Simultaneous Localization and Mapping (SLAM) field scale linearly with the amount of robot sensor data. For applications where robots are deployed for extended periods of time, this eventually becomes problematic. We show two optimizations which improve on the standard methods by identifying similarity in subsequent maps and by mitigating the performance and quality reductions from redundant observations.

Relevant previous work in SLAM divides primarily into two categories: particle-based methods and parametric methods. Particle-based methods such as FastSLAM, RBPFs and DP-SLAM have the advantage that the occupancy grid can be included in the state representation, since each hypothesis commits to a fixed position for each sensor reading [9], [7], [6]. This makes computing a dynamic map for planning purposes trivial, since the ML map can always be retrieved from the particle with the highest likelihood. However, particle-depletion makes these methods inadequate for high dimensional state spaces, such as the mapping task we explore in this work. Various parametric methods such as GraphSLAM, SqrtSAM and ESEIF [15], [8], [16] have been shown to scale much better to high-dimensional state spaces, such as those encountered when computing large joint maps for a team of robots. An important benefit of some of these methods is that they do inference on the entire graphical model underlying the SLAM problem, providing accurate posteriors quickly for the full trajectories of a team of robots. Unfortunately, these parametric methods make the rasterization of the global map more difficult, as grid-based representations must be built explicitly. Computing a dynamic gridmap using parametric techniques is further complicated because the maximum likelihood location of a sensor reading can change over time as more information is collected about the environment.

Others have presented mapping results for teams of robots using graph-based SLAM approaches, however, these do not address scalability issues for creating very large grid-maps with large numbers of robots [4], [11]. Thrun et al. have generated large-scale grid maps of mines but do so using only a single robot under manual control [14]. Others have considered decomposing the global mapping problem by building metrical maps on a per robot basis using existing methods and then merging them [2], [3]. However, such approaches are unable to guarantee a map which reflects the MLE positions of all the sensor observations.

So far, no one has developed an algorithm to efficiently rasterize a global occupancy map with a parametric SLAM model as a backbone. This may be because in small or medium size environments, even a naive algorithm is fast enough to be useful. In large-scale environments, such as those in the MAGIC 2010 contest, efficient map rasterization is crucial in enabling autonomous management of the robots.

This work was supported by U.S. DoD grant W56HZV-04-2-0001.

The authors are with the Department of Computer Science and Engineering, University of Michigan, Ann Arbor, MI 48109 USA {jhstrom,ebolson}@umich.edu
http://april.eecs.umich.edu

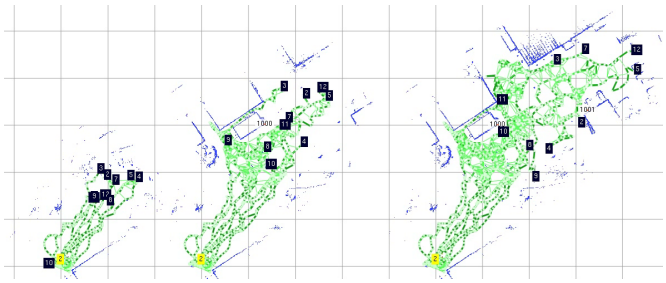


Fig. 2. Several GraphSLAM posteriors from the dataset that generated the map in Fig 1. Even though the SLAM graphs do not directly encode a grid map, we've shown that we can use this representation to efficiently compute an occupancy grid of the observed space.

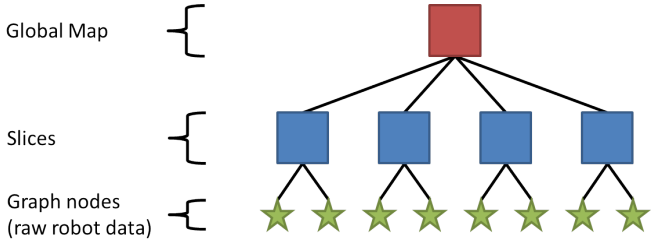



Fig. 3. Cache data structure using slices for $N = 2$. The leaves represent local maps from each robot. A single layer of globally aligned sub-maps are stored in the slices.

To address these shortcomings, we present a method which uses a sparse parametric SLAM algorithm to enable occupancy grid updates to be computed on-line for larger environments than any naive methods can handle. Our method exploits several properties of the mapping problem: We observe that after incorporating loop closures, the posterior of a historical robot position does not necessarily change, so parts of the gridmap may remain the same. We also observe that useful maps, e.g. for cooperative planning, only contain the static structure of the environment, omitting dynamic obstacles. Finally, we note that with some low-noise sensor types, e.g. laser range finders, a small number of views of a place are sufficient to form a complete map. These observations lead us to develop a map rasterization method which has the following properties:

- efficient average case update proportional to the impact of any loop closures
- globally consistent at every iteration
- applicable in dynamic environments by only rasterizing non-transient objects.
- fast enough for on-line use

## II. Problem Formulation

We consider the problem of generating a rasterized map using the laser-range-finder data associated with the trajectories of a team of robots. In particular, we build on the GraphSLAM formulation, where a SLAM graph $G$ contains a set of nodes $V$ and edges $E$. The nodes represent the location of a specific robot at a specific instant in time. Each node is associated with a local occupancy grid taken at that moment. The edges represent constraints between two poses that determine their relative position, including a covariance. An edge can be created using odometery and IMU readings from a single robot to form that robot's trajectory. Furthermore, additional edges can be added by finding correspondences in the sensor data associated with two nodes.

Several efficient methods exist for computing posteriors for the node locations, such as SqrtSAM, iSAM, or SGD [5], [8], [10]. Given the sensor data associated with each node and its posterior position, we can formulate a basic naive algorithm for building a global occupancy grid map. Starting with an empty map, we iterate over each node in the graph and rasterize the associated sensor data onto a global map. This rasterization algorithm scales linearly in the number of nodes in the graph: for applications where only the final map is desired, this method works quite well.

In the case of a team of actively exploring robots, we actually operate on a *sequence* of $n$ graphs $G_1, \cdots, G_n$, from which we want to produce maps $M_1 \cdots M_n$, e.g for use with an autonomous planner. We desire that the maps be computed on-line with a minimum of delay so that consumption by the planner can proceed directly. We further require that the maps reflect the exact MLE position of all the sensor data contained within it, and that moving obstacles are removed.

## III. Proposed Method

The proposed method uses a combination of techniques to speed up the computation of $M_{i+1}$ given previous maps $M_1 \cdots M_i$. We first introduce a spatially aware data structure that uses caching to exploit the similarity between subsequent maps. Second, we introduce the notion of a "node covering" which allows $M_i$ to be computed by potentially discarding redundant sensor data. These optimizations enable the time to compute a rasterization to scale with the size of the observed area, not with the duration of the system's operation (which could be quite long, or even unknown). Finally we show how this method can incorporate transient object removal.

### A. Slice Caching

Each subsequent node set is a superset of the previous set ($V_i \subseteq V_{i+1}$), so we could theoretically improve on the naive algorithm if we can exploit the overlap. For example, if the edges added to $G_i$ to form $G_{i+1}$ didn't include significant information gain (in the probabilistic sense), then the maps $M_i$ and $M_{i+1}$ can be quite similar. What we desire is a principled way to determine which portions of $M_i$ we can reuse to form $M_{i+1}$, so that if the graph isn't gaining information, the map update is very fast. In cases where the graph changes significantly, we must be willing to wait for an accurate map $M_{i+1}$ to be formed if we want to ensure the map is consistent.

We can exploit similarities in subsequent maps by partitioning the nodes $V_i$ into $k$ disjoint subsets $S_1 \cdots S_k$ we call *slices* (See Fig. 3). We then compute small maps $m_j$ from each slice $S_j$, such that the final map $M_j$ is the rasterization of all $m_1 \cdots m_k$. For each map update, the number of subsets that need to be converted to sub-maps will vary. In the

worst case all subsets will contain nodes whose posterior has changed and the data from every node will be examined twice – once to form $m_j$, and once again to form $M_i$. To achieve faster map updates, we can avoiding recomputing $m_j$ if the posterior position of no node in $S_j$ changes enough to cause a change in $m_i$. As we show in our evaluation, it is usually the case that enough nodes are *not* part of a loop closure that it is worthwhile to cache $m_i$.

The method for partitioning $V_i$ into subsets greatly impacts the runtime of the algorithm. Unfortunately it is not possible to know an optimal partitioning in advance. Ideally, we design the subsets to contain highly correlated graph nodes. This way, if $l$ nodes need to be recomputed, only $\lceil \frac{|S_k|}{l} \rceil$ slices need to be recomputed. If the subsets were not correlated, we might need to recompute as many as $l$ slices. We found that in practice, a good heuristic partitioning is to group $N$ sequential nodes from a single robot into a slice. These slices contain spatially and causally correlated nodes, so their posteriors are likely to change together. This means a single node is less likely to cause the recomputation of an entire slice unless the other nodes in that slice also have moved.

### B. Spatially Non-Redundant Node Coverings

To further increase the speed of computing the rasterization for a map, we note that much of the sensor data collected by the robots is redundant. For environments which are structurally static, such as indoor-outdoor urban environments, a single sensor observation can be sufficient to build an accurate map of small area. In particular, this is the case for low-noise sensors such as laser range finders.

This observation allows us to reformulate the occupancy map generation using only a subset of nodes, $C_i \subseteq V_i$, which *cover* all the nodes in $N_i$. We define a set of nodes to be covering if there exists no node not in $C_i$ which adds significant information to the map $M_i$. We can also show that this assumption actually allows us to improve map quality subjectively, since multiple observations of a single place can only result in blurring of features, potentially erasing important topological features such as doorways.

In practice, determining a minimal covering which meets this definition is at least as expensive as computing $M_i'$ from $V_i$, since it is unclear without examining each local map $m_j$ in detail if it will contribute significantly to the global map. However, we can define a conservative heuristic which will guarantee a complete covering in most environments, though it may not be minimal. We do this by considering the field of view which each local map covers, and the location at which it was taken. If both of these are too similar to an existing measurement, we can remove it from the covering.

The covering paradigm enables a smart decimation of the sensor data, which enables the computation of $M_i$ to be bound by the area it covers, rather than the up time of the system. In the worst case, the area could still be linearly dependent on the runtime. However, at system design time, we will likely have a prior over the size of the environments we are mapping, even if we don't have a prior over its runtime.

**Algorithm 1** CoverComposite($G_i = \langle V_i, E_i \rangle$)

```
for v ∈ V_i | v ∉ V_{i-1} do
    r = robot(v)
    if coverTest(v, C_r) then
        append(C_r, v)
        s_k = getNextSlice(S_r)
        addToSlice(s_k, v)
    end if
end for
for s_j ∈ S do
    if projectionError(s_j) >thresh then
        regenerate(s_j)
    end if
end for
M_i = initialize(boundingBox(S))
for s_j ∈ S do
    compositeOnto(M_i, s_j)
end for
threshold(M_i)
return M_i
```

### C. Transient Object Removal

The final step of the rasterization process is determining how to ensure transient objects are not included in the final map. As previously mentioned, transient objects cause problems for autonomous planners because their presence can obscure the existence of more explorable terrain through a doorway. As a result, we must carefully detect which sensor readings are due to transient objects. In principle, such objects are easy to detect by looking for disagreement among sensor data as to the free or occupied status of a cell. In practice however, some disagreement can be caused by alignment error and does not necessarily indicate a transient object. In cases where alignment is bad (e.g. when the SLAM Graph is poorly constrained), we want to mitigate the possibility that entire walls will be removed. The "bias" to label an object as *transient* reflects the degree to which the system designer wishes to be resistant to alignment error vs the desire to ensure all doorways are correctly marked as passable terrain. An efficient solution to this problem is to view multiple sensor readings of a particular cell as observations of a binary variable with equal covariance. The probability of the cell being an obstacle is given by:

$$p(obs|z_0 \cdots z_n) \sim p(obs) \prod_i p(z_i|obs)$$

This can be computed directly by summation of log probabilities [13]. A simple threshold on $p(obs|z_0 \cdots z_n)$ can then be used to determine whether the cell should be marked as occupied or free, reducing the process to a vote at each cell.

### D. Proposed Implementation

By combining the improvements from the node covering with the slice method we propose the map compositing method '$CoverComposite$' in Algorithm 1. We assume that another module, e.g. GraphSLAM, is producing a sequence

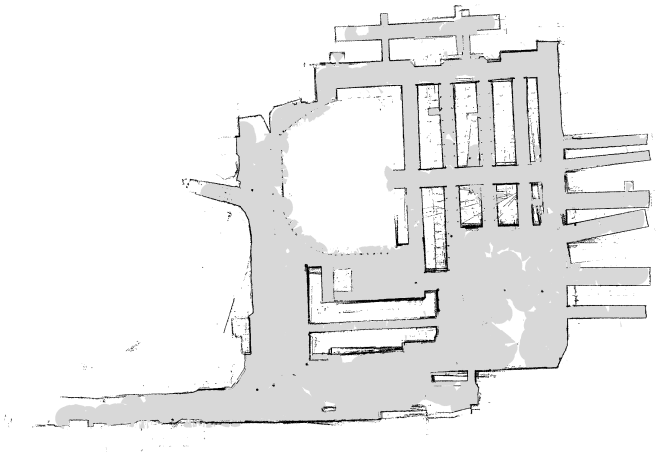


Fig. 4. Final occupancy-grid map generated from the phase 2 portion of the MAGIC 2010 contest. The map includes data from 15 robots collected over the course of 72 minutes. Size: 220m x 160m @ 0.1m resolution.

of posterior SLAM graphs $G_i, G_{i+1}, \cdots$. The algorithm we present is run on successive $G_i$. For convenience, the cover and slice sets are stored in memory as $C_r$ and a set of slices $S_r$ for each robot $r \in \{0, \cdots, R\}$. First, all the new nodes that first appear in graph $i$ are examined to determine if they belong in the covering for the robots which spawned them. If so, they will be added to an existing slice if there is room, or otherwise force the creation of a new slice. Next, each slice is examined to determine if the projection error of any update of the state of any of its member nodes would cause an error of more than some threshold (e.g. 1 pixel). If an update is needed, the cumulative gridmap for that slice is regenerated according to the most recent posteriors for each node. Finally, all the slices for all robots are composited together to form the final occupancy grid. The final grid keeps track of how many slices observed each cell as either observed or free. Using the probabilistic scheme described in the previous subsection, we convert that representation to a binary map via a fixed threshold.

The asymptotic complexity bounds of our method is the same as the naive algorithm we described earlier. In the worst case the proposed implementation is at most twice as slow, since each sensor datum must be examined at most twice. However, we have shown empirically that the average run time is much faster than the naive rasterization algorithm.

| Algorithm | Nodes composited |
|---|---|
| NaiveComposite | All |
| NaiveCoverComposite | Cover (every timestep) |
| CoverCompositeN | Cover (only when necessary) |

TABLE I

METHODS USED IN THE EVALUATION

## IV. EVALUATION

To evaluate the effectiveness of the proposed method, we implemented two naive rasterization algorithms, in addition

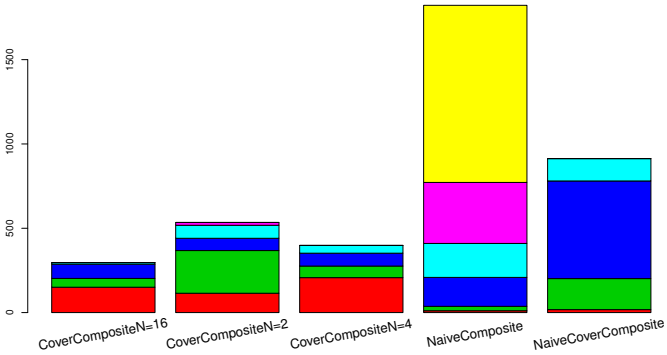



Fig. 5. Cumulative runtimes for each algorithm on phase 2. Color bands depict histograms indicating which portion of rasterizations finished in under 1.0s, 2.0s, etc, for colors of red, green, blue, cyan, magenta and yellow, respectively.

| Dataset | Size | # robots | Num. Graphs |
|---|---|---|---|
| Keswick Barracks | 150m x 150m | 10 | 68 |
| Showgrounds (phase2) | 220m x 160m | 14 | 429 |

TABLE II

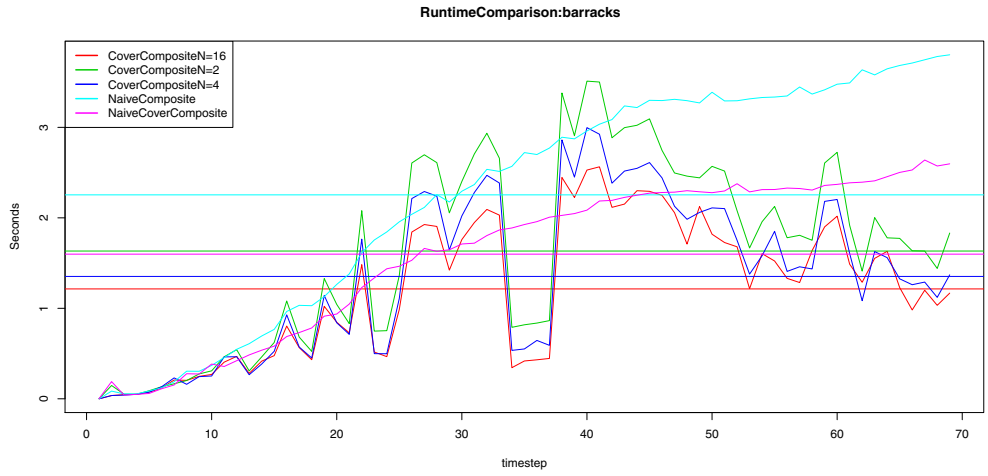
DATASETS USED IN THE EVALUATION

to three versions of the proposed method by varying $N$, the parameter which determines how many nodes are in each slice (see Table I). The first naive method, NaiveComposite, computes a batch composite of all the nodes every timestep. The second naive method, NaiveCoverComposite, rasterizes only the nodes in the cover at each timestep.

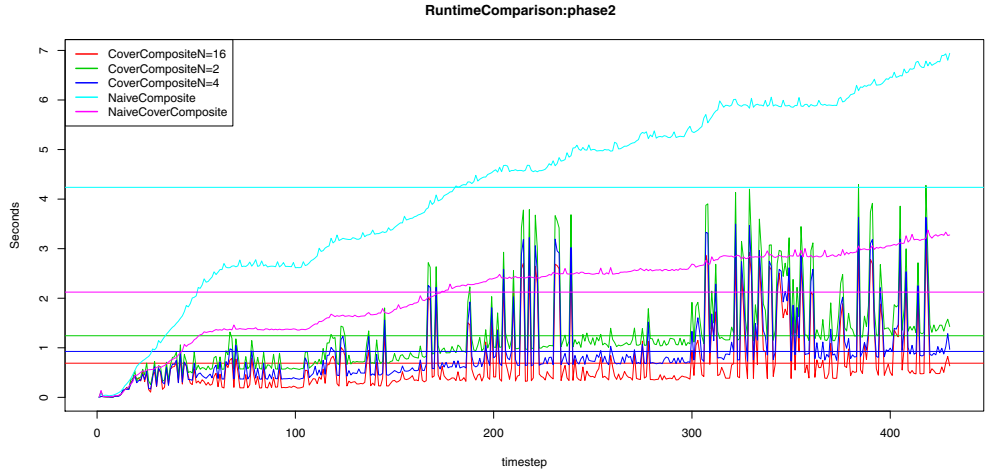
We evaluated all five algorithm parametrizations on two separate datasets: one involving 10 robots autonomously exploring a section of the Keswick Barracks, and another of 14 robots autonomously exploring a portion of the Adelaide Showgrounds as part of the MAGIC 2010 contest (see Table II for specific dimensions). While the datasets are both of structured, man made environments, they have significantly different topology, which enables us to evaluate performance under varying conditions. The maps corresponding to the final graphs in each dataset are shown in Fig. 1 and 4 Both data sets were taken in Adelaide, South Australia. To our knowledge, no other public datasets of this scale exist.

To evaluate the effectiveness of each method, we ran each algorithm on a sequence of graphs for each dataset, collected at 10 seconds intervals. We evaluated each method primarily on the total runtime of the rasterization. All algorithms use the same node posteriors, so we expect the maps from each algorithm to be very similar. While it would also be desirable to quantitatively evaluate the map quality from each method, as in [1], we lack accurate ground truth for the environments in question (see Fig. 4). In practice, we found that our cover heuristic consistently produced structurally accurate maps. Subjectively, we even saw a slight improvement to map quality using the covering method, as can be seen in Fig. 7.

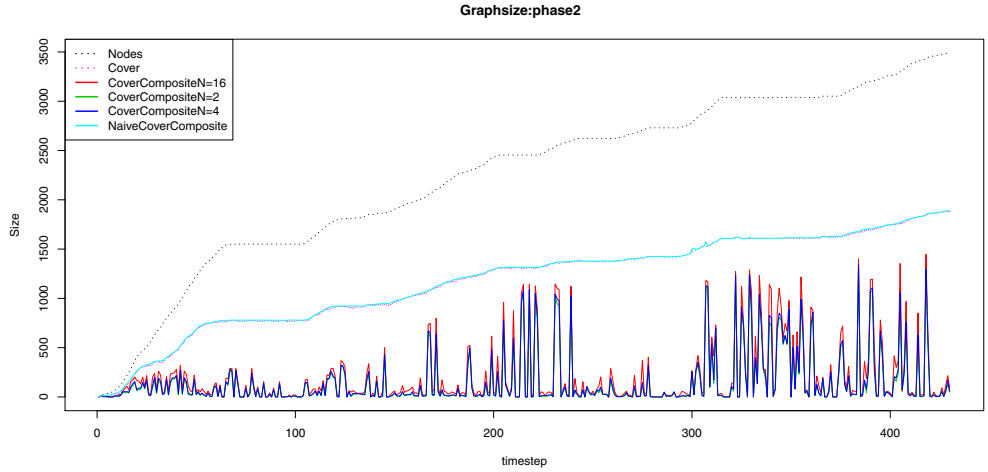
For evaluation, we timed the duration of each map update for each method on a 2.53 Ghz Intel i5 laptop. Our algorithms are single threaded and implemented in Java. To determine which algorithm is the most CPU efficient,




(a) Detailed timing for the graphs $G_0 ... G_{67}$ on the Keswick dataset. Horizontal lines indicate average runtime for that algorithm. Large runtimes indicate loop closures occurred between the $(i-1)^{\text{th}}$ and the $i^{\text{th}}$ timestep which changed the ML posterior for the position of many nodes.

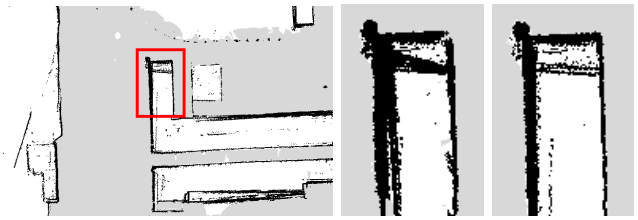



(b) Detailed timing for the graphs $G_0 ... G_{429}$ on the phase 2 dataset. Horizontal lines indicate average runtime for that algorithm. Vertical spikes are indicative of significant loop closures. In this case, the system is still functioning well after over an hour in a large environment with many loop closures.




(c) Time varying totals for the number of nodes in the graphs, the number of nodes in the cover, and the number of nodes which must be regenerated for each algorithm at each timestep.

Fig. 6.



(a) Context (b) Original (c) w/ Cover

Fig. 7. Map quality improvement using a node covering for a problematic section of the phase2 map. This technique reduces feature blurring while still providing structurally equivalent maps.

consider the integral time over the phase 2 dataset shown in Fig. 5 (lower is better). The plot shows that although $N = 16$ has the lowest total time, the $N = 4$ parametrization actually has more instances finishing in under 1.0 seconds (red) than any other method. Despite being slightly slower, we choose $N = 4$ as the best, since it reduces the planning delay the most often. Note that when large scale loop closures occur, we must be willing to incur a heavier planning delay if we want a map which accurately reflects the MLE position of each sensor scan. This is because any cached rasterization is largely useless when the underlying SLAM graph changes significantly. This plot also shows that the CoverComposite algorithm out-performs both naive implementations.

More detailed timing information for both datasets is available in Figs. 6(a) and 6(b). These plots show the time required by each method to rasterize the graph at each timestep in both datasets. The average time for each method is shown as a horizontal bar. The naive algorithms show steadily increasing compute time as the number of nodes in the graph (or cover) grows, which is expected. The cached CoverComposite algorithm has variable runtime depending on the number of slices which need to be recomputed. Fig. 6(c) shows the number of nodes in the graph, the size of the cover, and the number of nodes which are recomputed at each step. Comparing Figs. 6(c) and 6(b) confirm our claims that the CoverComposite family of algorithm only incurs significant CPU time when large numbers of nodes have moved significantly due to a loop closure. The data show that the proposed method is significantly more efficient than the highly optimized naive algorithms we compared it to.

## V. Conclusion

We explored the problem of generating large occupancy grid maps using a parametric SLAM method as a backbone. We introduced a method that improves on the scaling properties of standard rasterization methods to produce occupancy grids. We analyzed two significant optimizations, first by formulating a subset of nodes whose sensor data effectively covers all the nodes in the SLAM graph, and secondly by formulating a spatially-aware data structure to exploit similarities in the posterior positions of the nodes in successive SLAM graphs $G_i$ and $G_{i+1}$. The improvements our proposed method makes allow us to generate occupancy grid maps online and use them as for high level exploration planner or for human observation of a team of robots.

Future work in this domain might explore node coverings which are smaller than the conservative one we presented. Additionally, methods for optimal partitioning of the graph could be explored in greater detail, potentially by formulating a better prior over the correlation between graph nodes. If the single-threaded performance demonstrated here is insufficient for certain applications, several portions of the map building process are trivial to parallelize, enabling significant speedup if additional CPUs or GPUs are available, as in [17].

## References


[1] B. Balaguer, S. Balakirsky, S. Carpin, and A. Visser. Evaluating maps produced by urban search and rescue robots: lessons learned from robocup. *Autonomous Robotics*, 27:449–464, 2009.

[2] A. Birk and S. Carpin. Merging occupancy grid maps from multiple robots. In *Proceedings of the IEEE*, volume 94, pages 1384–1397, 2005.

[3] S. Carpin. Fast and accurate map merging for multi-robot systems. *Autonnomous Robotics*, 25:305–316, 2008.

[4] H. J. Chang, C. S. G. Lee, Y. C. Hu, and Y.-H. Lu. Multi-robot slam with topological/metric maps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[5] F. Dellaert and M. Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Research*, 25(12):1181–1203, December 2006.

[6] A. Eliazar and R. Parr. Dp-slam 2.0. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[7] G. Grisetti, S. Grzonka., C. Stachniss, P. Pfaff, and W. Burgard. Efficient estimation of accurate maximum likelihood maps in 3d. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, USA, 2007. Accepted for publication.

[8] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Fast incremental smoothing and mapping with efficient data association. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Rome; Italy, April 2007. To appear.

[9] M. Montemerlo. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2003.

[10] E. Olson, J. Leonard, and S. Teller. Spatially-adaptive learning rates for online incremental SLAM. In *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

[11] M. Pfingsthorn, B. Slamet, and A. Visser. A scalable hybrid multi-robot slam method for highly detailed maps. In *RoboCup 2007*, pages 457–464, 2008.

[12] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *Proceedings of AAAI-00*, 2000.

[13] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[14] S. Thrun, D. Hahnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker. A system for volumetric robotic mapping of abandoned mines. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[15] S. Thrun and M. Montemerlo. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *International Journal of Robotics Research*, 25(5-6):403–430, May-June 2006.

[16] M. R. Walter, R. M. Eustice, and J. J. Leonard. Exactly sparse extended information filters for feature-based SLAM. *Int. J. Rob. Res.*, 26(4):335–359, 2007.

[17] M. Yguel, O. Aycard, and C. Laugier. Efficient gpu-based construction of occupancy grids using several laser range-finders. *International Journal of Vehicle Autonomous Systems*, 6, 2008.