

# View management for lifelong visual maps

Nandan Banerjee\*, Ryan C. Connolly, Dimitri Lisin, Jimmy Briggs, Manjunath Narayana, and Mario E. Munich

**Abstract**—The time complexity of making observations and loop closures in a graph-based visual SLAM system is a function of the number of views stored [1], [2]. Clever algorithms, such as approximate nearest neighbor search, can make this function sub-linear. Despite this, over time the number of views can still grow to a point at which the speed and/or accuracy of the system becomes unacceptable, especially in computation- and memory-constrained SLAM systems. However, not all views are created equal. Some views are rarely observed, because they have been created in an unusual lighting condition, or from low quality images, or in a location whose appearance has changed. These views can be removed to improve the overall performance of a SLAM system. In this paper, we propose a method for pruning views in a visual SLAM system to maintain its speed and accuracy for long term use.

## I. INTRODUCTION

Mobile robots capable of mapping their environment using SLAM are increasingly common today, not only in research labs and warehouses, but also in homes and businesses. Often such robots create a map once, and then use it to perform their tasks [3]. A static map is sufficient for highly controlled environments such as a warehouse, or spartan environments such as hotel hallways. In more common household environments, dynamic conditions arise due to variations in lighting, movement of furniture, and appearance of clutter. In time, a static map no longer represents the current state of the environment, leading to degradation or failure in localization.

Ideally, a robot should detect changes in its environment and update its map accordingly. Maintaining an up-to-date representation of the space allows the robot to estimate its location more accurately and to make better navigation decisions. We refer to the maps being continuously updated by the robot as *lifelong visual maps* [4].

In the SLAM literature, the term *view* refers to a uniquely identifiable scene in the world whose location can be estimated by the robot using its sensors. In the context of vision-based SLAM systems, the sensor is a camera, and a view is a representation of a visual scene (see Fig. 1) that can be observed by the robot to estimate its location and orientation.

The main contribution of this paper is an efficient view management algorithm for resource-constrained visual SLAM systems. The algorithm selects views that can be removed from the map without sacrificing localization performance. View removal is based on observation statistics collected over time. The algorithm has been developed for



Fig. 1: A sample view from an environment as seen by our robot with the red crosshairs indicating detected keypoints.

the monocular visual SLAM system by Eade et al. [1]; however, it can be easily adapted to fit any graph-based SLAM system containing a notion of views.

In our system, we represent the map of the robot’s environment as a pose graph [1] where each node represents either a robot pose or a view, and each edge represents a measurement either between a pair of robot poses or between a robot pose and a view (see Fig. 2). Pose-to-pose edges give a robot trajectory estimate that may drift over time due to measurement errors from dead reckoning sensors. Pose-to-view edges create loop closures in the graph [1]. These loop closures allow the SLAM system to minimize accumulated measurement errors with a graph optimization algorithm (e.g. [5], [6]) thus correcting for the drift.

Conceptually our SLAM system can be divided into the *back end*, responsible for constructing and optimizing the graph, and the *front end* responsible for creating and recognizing the views. A view is created by selecting a pair of video frames, finding point correspondences between them, triangulating the resulting pairs of points, and refining the 3D points using non-linear optimization to minimize the re-projection errors.

The back end’s most expensive operation is the nonlinear graph optimization, whose complexity is a function of the number of nodes and edges in the graph. These in turn are a function of time rather than space. As the robot moves, the graph grows, and the cost of determining the robot’s pose can become prohibitive. To counteract this problem, graph sparsification techniques that periodically remove nodes and edges have been developed [1], [7].

In the front end, most of the computational cost comes from observing existing views. Creating a new view incurs a constant cost, because it does not use information about

\*The authors are with the iRobot Corporation, Bedford, MA 01730, USA. {nbanerjee, dlin, jbriggs, mmarayana, mmunich}@irobot.com, ryan.connolly@duke.edu

existing views. In contrast, view observation requires the robot to determine if an image captured by the camera matches one of the known views. In our system, we select a fixed number of likely candidates from the pool of all views using a candidate selection algorithm, and then compare these candidates to the current image. Thus, the complexity of the candidate selection method during the process of view observation is a function of the total number of views in the system.

If the robot operates for long periods of time in the same space, it might continue to create new views because of changes in the appearance of its surroundings, such as different lighting conditions or moving furniture (Fig. 3). Eventually, the distribution of the views can become very dense (Fig. 4), which can make view observation unacceptably slow, and degrade the robot’s performance, rather than improve the localization accuracy. Unfortunately, graph pruning only removes pose nodes, it does not limit the growth of the number of views over time.

## II. RELATED WORK

Reduction of computation and memory devoted to view recognition has been an ongoing goal of the SLAM community for some time. In this section, we explore some of the more successful approaches and compare them to our algorithm.

Eade et al. attempt to control the complexity of view recognition by using an approximate nearest-neighbor algorithm [8]. Instead of comparing the camera image to all existing views, it pools the visual feature descriptors from the views into a tree structure for approximate nearest neighbor matching, and uses a voting scheme to select a short list of candidates. This approach is logarithmic in the number of the view features, and sub-linear in the number of views. Nevertheless, if the number of views is allowed to grow unabated, eventually the cost of view recognition becomes

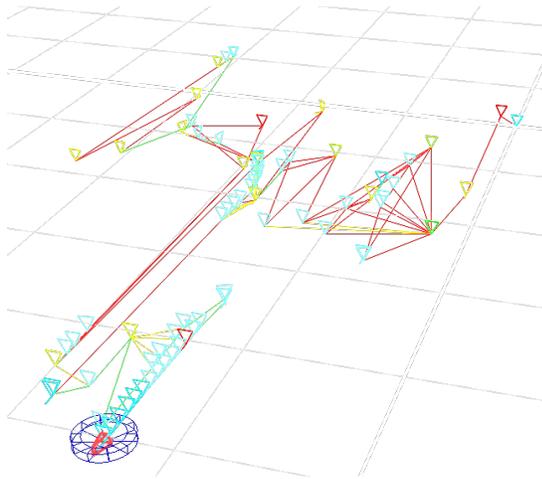


Fig. 2: The graph as created by the robot over a 10-minute run, where the cyan  $\nabla$  represents pose nodes, the red, yellow, and green  $\nabla$  represents view nodes that are new, slightly observed, and frequently observed respectively. The yellow edges are observation edges, the green ones are combined observation edges, and the red ones are combined observation and odometry measurement edges.

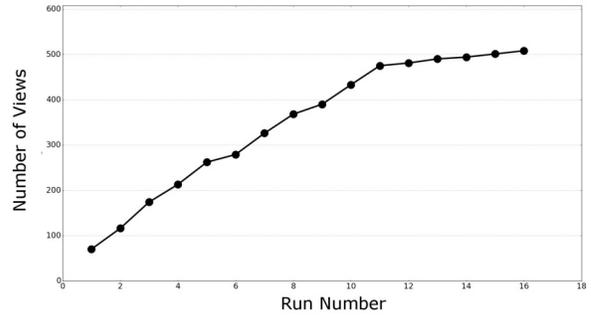


Fig. 3: The number of views in a visual SLAM system grows over time. The graph shows the number of views increasing as the robot runs in the same environment several times under different lighting conditions. Eventually, too many views will degrade the speed and accuracy of the system.

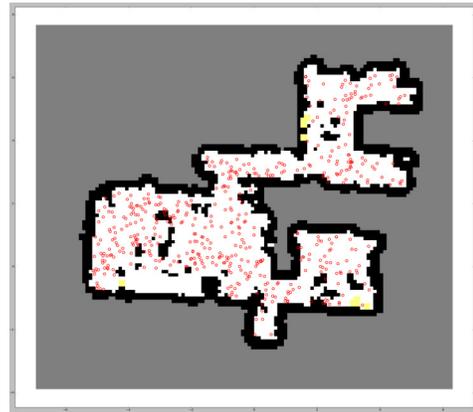


Fig. 4: Map of an environment (around  $500 \text{ ft}^2$ ) without view management after 20 runs – the distribution of views (represented in red) in the map is very dense.

prohibitive for a resource-constrained robot platform. It is therefore necessary for the SLAM system to keep the number of views below a maximum determined by the hardware capabilities.

View clustering techniques use various heuristics to prune views which are likely to duplicate information. Konolige and Bowman [4] introduced this class of methods with a “least recently used heuristic” to delete views. Hochdorfer et al. explored clustering algorithms to reduce the co-observation of features in multiple views [9]. These early works decreased the number of views stored without significantly affecting localization performance, but were sensitive to changes in lighting and movement of clutter. Our algorithm maintains performance even with these variations. Other heuristics developed include approaches which remove views unlikely to survive the ANN search [10], filter by estimates of distinctiveness, detectability, and repeatability [11], or learn which features are most likely to perform well, and remove the others [12].

Mühlfellner et al. [13] proposed a two-phase algorithm known as *Summary Maps*. In the first phase, high quality views are selected in an offline processing step. In the second phase, the resultant subset of views (the summary map) is sent back to the navigation system and used for subsequent SLAM runs. Bürki et al. extended this approach to

match loaded summary maps to lighting conditions [14], and Dymczyk et al. explored linear and quadratic programming approaches for the offline computation step [15]. While these techniques do speed up computation in the vision front-end, they all require expensive offline computation. Often this computation is outsourced from an agent to a separate server. On resource-constrained, independent platforms, however, this offline computation is prohibitively expensive. Our method can manage views continuously on the agent itself during SLAM and thus can work without requiring a connection to a server.

Other work on lifelong localization and mapping addresses outdoor autonomous systems with high compute power, such as those used by self-driving cars [16], [17]. Because of this high computational capacity, such systems are not overly concerned with reducing the size of the map.

### III. PROBLEM STATEMENT

A mobile robot equipped with a visual SLAM system starts building a new map at the beginning of a run, and maintains localization by observing the map’s views. Up to a point, having more views improves the robot’s localization accuracy, by increasing the likelihood of observation. However, the number of views cannot grow without bound. There is a critical number of views, beyond which the speed or accuracy of retrieving a matching view becomes unacceptable, or which simply exceeds the available memory capacity. This maximum number of views must be determined experimentally by testing that the system runs at an acceptable frame rate, leaves enough CPU cycles for other tasks needed by the robot, and, of course, does not crash because it ran out of memory.

In a confined space where the lighting does not change much, the SLAM system may stop creating new views before their number reaches the maximum imposed by the hardware, but in a large space where lighting conditions can vary widely, the system is likely to reach the maximum number of views eventually, especially if it saves the map at the end of a run, and re-uses it for the next run. Therefore, we need a method for pruning the views to keep their number under the limit.

### IV. METHOD

In this section, we describe our proposed algorithm for identifying views that are to be pruned. First, we define the terminology used in our visual SLAM system. Then we discuss the properties of our system that we would like to preserve after the pruning of views. Finally, we explain the metrics used for determining the quality of a view and explain our proposed algorithm for managing views. We use the terms *management* and *pruning* interchangeably in this paper.

The SLAM graph can consist of multiple disjoint sub-graphs, called *components*, which ideally represent different environments. A map is defined as a component of the graph with all its associated views and occupancy information. Note that each component has its own coordinate system.

When the robot begins a run without any prior knowledge of its location, it creates a new component. It may also have one or more components in its SLAM graph, which were saved during the previous runs. Let us denote the newly created component as *A*. If there is a component *B* in the graph, corresponding to the robot’s current environment, then the robot is able to merge *A* and *B* after observing some minimum number of views from *B*. We call this event a *relocalization* into a map.

There are several properties of the SLAM system that we must preserve throughout the view pruning process. First, despite removing some of the views from a component, the robot must still reliably observe enough of the remaining views to relocalize into that component. The appearance of any given environment can change drastically between runs, because of lighting changes and moved objects. These changes can make it impossible for the robot to observe some of the views during certain runs. For example, a view that was created during daylight hours might not be observable in the evening, when the electric lights are on. Other views may be observable in both lighting conditions. It is critical to keep more of the views, which were observed in multiple runs, which are more likely to allow the robot to relocalize despite the appearance changes. On the other hand, removing the views that have not been observed in a long time provides a way for the SLAM system to forget visual scenes that are no longer observable, such as when furniture has been moved.

Along with preserving the views observable across different runs, view pruning should also attempt to keep the distribution of views across a component as uniform as possible. Otherwise, there may be areas in the environment, where a robot can travel for too long without observing a view, and thus accumulate localization error. We enforce the uniformity of the view distribution by pruning only those views which neighbor other views of similar orientation.

We now present our algorithm for selecting a set of views in a component that can be pruned without negatively affecting relocalization, or the uniform spatial distribution of views. First, we describe the parameters of the algorithm, then the score calculation, and finally the algorithm itself.

#### A. Parameters

The following are the parameters of our algorithm:

- **MIN\_VIEWS** : Minimum number of views required before the pruning algorithm is run. The value of this parameter should be determined experimentally, to make sure that the SLAM system runs at the required frame rate, does not crash, and leaves enough cycles for other tasks.
- **NN\_THRESHOLD** : Nearest neighbor threshold, if the total number of views nearby is less than the threshold, the view will not be marked for deletion.
- **NN\_VOXEL\_SIZE** : 3-element vector ( $x$ ,  $y$ , and  $\theta$ ) used for constraining the search space while finding neighboring views.

- **SCORE\_THRESHOLD** : If the view score is less than this threshold, the view will be deleted.
- $W_1$  : Used to weight views that are used for relocalizing into another component.
- $W_2$  : Used to weight total observations ratio in the current run.
- $W_3$  : Used to weight total runs where the view was observed at least once.

### B. View management algorithm

Along with the data that helps to observe a view, our view data structure also stores some statistics that are used to determine its score. These include the total number of times the view has been observed in the current robot run, the time when the view was created, the number of runs the view was observed in, and whether the view has been used during relocalization into a component.

---

#### Algorithm 1 Find views for deletion

---

**Require:** Set  $\mathcal{V}$  of all the views

- 1: **if**  $|\mathcal{V}| \leq \text{MIN\_VIEWS}$  **then**
- 2:   return
- 3:  $\mathcal{V}_{keep} \leftarrow \text{newlyCreatedViews}()$
- 4: **for**  $v$  in  $(\mathcal{V} \setminus \mathcal{V}_{keep})$  **do**
- 5:    $score[v] \leftarrow$   

$$\left( W_1 \times reloc + W_2 \times \frac{n\_obs\_cur}{max\_obs} + W_3 \times \frac{n\_obs\_runs}{n\_runs} \right)$$
- 6:   **if**  $score[v] > \text{SCORE\_THRESHOLD}$  **then**
- 7:      $\mathcal{V}_{keep} \leftarrow \mathcal{V}_{keep} \cup \{v\}$
- 8:  $\mathcal{V}_{delete} \leftarrow \mathcal{V} \setminus \mathcal{V}_{keep}$
- 9:  $\text{sort}(\mathcal{V}_{delete}, score)$  # Sort by score
- 10:  $D \leftarrow \emptyset$
- 11: **for**  $v$  in  $\mathcal{V}_{delete}$  **do**
- 12:   **if**  $\text{numNearestNeighbors}(v, D) < \text{NN\_THRESHOLD}$  **then**
- 13:      $\mathcal{V}_{keep} \leftarrow \mathcal{V}_{keep} \cup \{v\}$
- 14:   **else**
- 15:      $D \leftarrow D \cup \{v\}$
- 16:  $\mathcal{V}_{delete} \leftarrow \mathcal{V} \setminus \mathcal{V}_{keep}$
- 17: **return**  $\mathcal{V}_{delete}$

---

Let  $\mathcal{V}_{keep}$  be the subset of views that should be kept and  $\mathcal{V}$  be the set of all views.  $\mathcal{V}_{keep}$  is initially empty, and is then populated with views that were created in the current run and observed at least once. Then a score is computed for the pre-existing views, based on the observation statistics from the current and previous runs, as well as the view’s usefulness for relocalization. Views with scores higher than the threshold are also added to the set  $\mathcal{V}_{keep}$ . The view score is calculated as follows:

$$W_1 \times reloc + W_2 \times \frac{n\_obs\_cur}{max\_obs} + W_3 \times \frac{n\_obs\_runs}{n\_runs}$$

where  $n\_runs$  is the total number of runs when the view was present in the component,  $n\_obs\_runs$  is the number of runs

where the view was observed,  $n\_obs\_cur$  is the number of observations of the view in the current run, and  $max\_obs$  is the maximum number of observations of any view in the current run.  $reloc$  is 1 if the view was used for relocalization and 0 otherwise.  $W_1$ ,  $W_2$ , and  $W_3$  are the weights.

The views that are currently in the set of  $\mathcal{V}_{delete}$  containing  $\mathcal{V} \setminus \mathcal{V}_{keep}$  are sorted by view score in ascending order. The number of nearby views for each view in  $\mathcal{V}_{delete}$  is checked, and if it is less than the **NN\_THRESHOLD**, the view from  $\mathcal{V}_{delete}$  and added to  $\mathcal{V}_{keep}$ . This constraint ensures a uniform spatial distribution of the views. In the end  $\mathcal{V}_{delete}$  is the set of views that can be pruned without affecting the system’s performance. The pseudo-code for the algorithm described above is presented in Algorithm 1. Depending on the system, one might also want to add an upper bound on the total number of views per map, which can be determined experimentally based on the area of the space that is being mapped.

## V. EXPERIMENTS AND RESULTS

We have performed extensive experiments and analysis to evaluate the algorithm’s performance and tune its parameters. We have collected run-time logs using a proprietary vacuum cleaning robot platform in several environments at different times of day with varied lighting conditions. The logs contain timestamped odometry readings, gyro readings, and camera images. Since most robot runs start from a dock / charging station, our logs start from the same location. We ran our visual SLAM system off-line on the logs, saving the map at the end of each run, so that it could be loaded for the next run. A saved map contains information about the SLAM graph, the views, and other supporting data. Trying to make an observation from a set of 2000 views would take more time than from a set of 200 views. Since this is dependent on the candidate view selection algorithm, CPU utilization results will vary based on what algorithm is being used. It follows that with increasing number of views, leading to an increase in the amount of time needed for making observations necessary for loop closures, the accuracy of the system will degrade. Because these are system dependent, and due to space constraints, we do not show any data related to this. Instead we concentrate on data related to the metrics that help us choose the ideal views for removal.

### A. Evaluation criteria

In Section IV we described the properties of the SLAM system that we have to preserve despite removing some of the views. The most important of those is the ability to relocalize in an existing map under various appearance changes of the environment. It is also important for the robot to be able to relocalize quickly, as opposed to wandering around for a long time looking for familiar views.

Thus we use the following criteria for measuring the robot’s ability to relocalize:

- *Distance between cross-observations* is the distance the robot has to travel before observing a view created during a previous run.

- *Fraction of cross-observed frames* is the fraction of the video frames in which the robot observed a view from a previous run.
- *Relocalization distance* is the distance the robot has to travel before relocalizing, i. e. before observing the required number of views in another component (from a previous run in the same environment).

We compute these values for every robot run, and average them across all runs.

TABLE I: View score weight selection ( $W_2 = 1$ , score threshold of 0.25 times max\_score)

$W_1$	$W_3$	Avg. distance between cross observations	Fraction of cross observed frames	Growth rate
0	0	1.429	0.054	0.7
0	0.5	0.606	0.106	14.45
0	1	0.604	0.107	15.2
0	1.5	0.607	0.106	14.4
0	2	0.572	0.107	14.65
0	2.5	0.602	0.107	14.2
<b>0</b>	<b>3</b>	<b>0.727</b>	<b>0.081</b>	<b>4.1</b>
0.5	0	1.093	0.057	1.15
0.5	0.5	0.61	0.104	15.05
0.5	1	0.598	0.109	14.45
0.5	1.5	0.618	0.105	14.85
0.5	2	0.603	0.106	14.7
<b>0.5</b>	<b>2.5</b>	<b>0.699</b>	<b>0.085</b>	<b>3.45</b>
<b>0.5</b>	<b>3</b>	<b>0.748</b>	<b>0.086</b>	<b>3.85</b>
1	0	1.196	0.058	1
1	0.5	0.673	0.104	14.45
1	1	0.611	0.104	14.35
1	1.5	0.615	0.104	15.05
1	2	0.841	0.077	2.45
<b>1</b>	<b>2.5</b>	<b>0.794</b>	<b>0.082</b>	<b>3.65</b>
<b>1</b>	<b>3</b>	<b>0.673</b>	<b>0.086</b>	<b>4.15</b>
1.5	0	1.095	0.061	1.2
1.5	0.5	0.565	0.108	14.35
1.5	1	0.603	0.105	14.5
1.5	1.5	0.957	0.067	1.3
1.5	2	0.914	0.073	2
<b>1.5</b>	<b>2.5</b>	<b>0.781</b>	<b>0.082</b>	<b>3.3</b>
<b>1.5</b>	<b>3</b>	<b>0.711</b>	<b>0.084</b>	<b>4.25</b>
2	0	1.183	0.059	0.95
2	0.5	0.67	0.105	15.1
2	1	1.211	0.059	1.15
2	1.5	0.942	0.068	1.55
2	2	0.844	0.075	2.3
<b>2</b>	<b>2.5</b>	<b>0.689</b>	<b>0.086</b>	<b>3.45</b>
<b>2</b>	<b>3</b>	<b>0.774</b>	<b>0.085</b>	<b>4.25</b>

The other goal of our algorithm is to limit the growth of the number of views in the graph. To measure that, we define the *view growth rate*  $G$  as follows:

$$G = \frac{v_n - v_2}{n - 1}$$

where  $v_i$  is the number of views at the end of the  $i$ -th run, and  $n$  is the total number of runs. We start from the second run because in the first run the robot creates a brand new map, rather than relocalizing into an existing map.

### B. View score weights selection

We divide the algorithm parameters into two subsets: the weights for computing the view scores, and the parameters

for counting the nearest neighbor views. It is not practical to tune both subsets simultaneously, because the scoring function and the nearest neighbor check work against each other. Therefore, first, we disable the nearest neighbor check and tune the view score weights using the distance between cross-observations, the fraction of cross-observed frames, and the view growth rate.

We want to choose the weights such that the distance between cross-observations is low, the fraction of cross-observed frames is high, and the growth rate is low. The values in Table I were generated by running the robot logs of environment A sequentially 22 times. We used a threshold of 5.0 for the growth rate, 0.8m for the distance between-cross observations, and 0.075 for the fraction of cross-observed frames. From the table we see that the weights of (0, 1, 3), (0.5, 1, 2.5), (0.5, 1, 3), (1, 1, 2.5), (1, 1, 3), (1.5, 1, 2.5), (1.5, 1, 3), (2, 1, 2.5), and (2, 1, 3) are good choices.

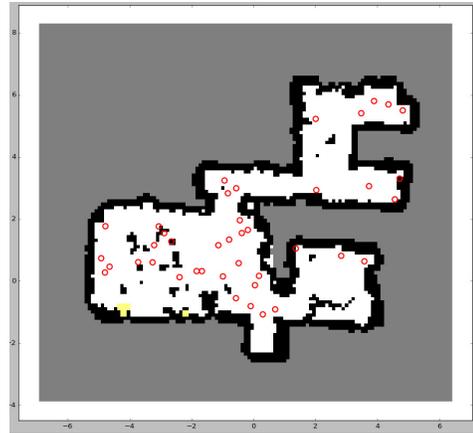


Fig. 5: Uneven distribution of views after 20 runs in an environment with only view score based pruning.

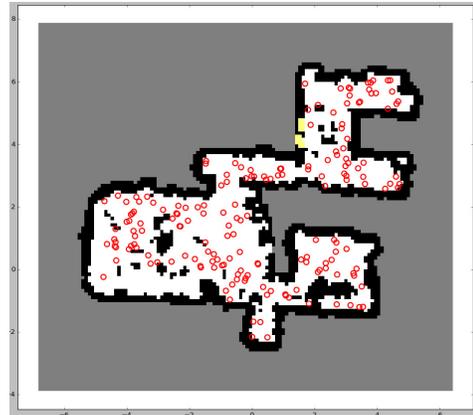


Fig. 6: Uniform distribution of views after 20 runs in the same environment as the previous figure after applying the nearest neighbor constraint.

### C. Nearest neighbor threshold selection

The nearest neighbor function returns the number of neighboring views present in a voxel of size specified by NN\_VOXEL\_SIZE surrounding the view that is being queried. After views are marked for deletion based on the view score, the nearest neighbor check is performed. The

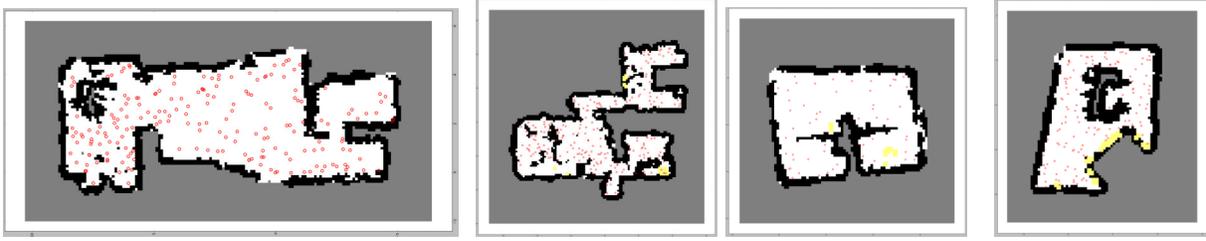


Fig. 7: Occupancy maps with overlaid views of different environments A, B, C, and D (from left to right). Env A and Env B are large with around 650 and 500  $ft^2$  of navigable area respectively. Env C and Env D have about 300  $ft^2$  of navigable area. Logs from Env A and Env B have more texture on walls and furniture than logs from Env C and Env D. All logs were collected with varied lighting conditions, both during daytime and nighttime.

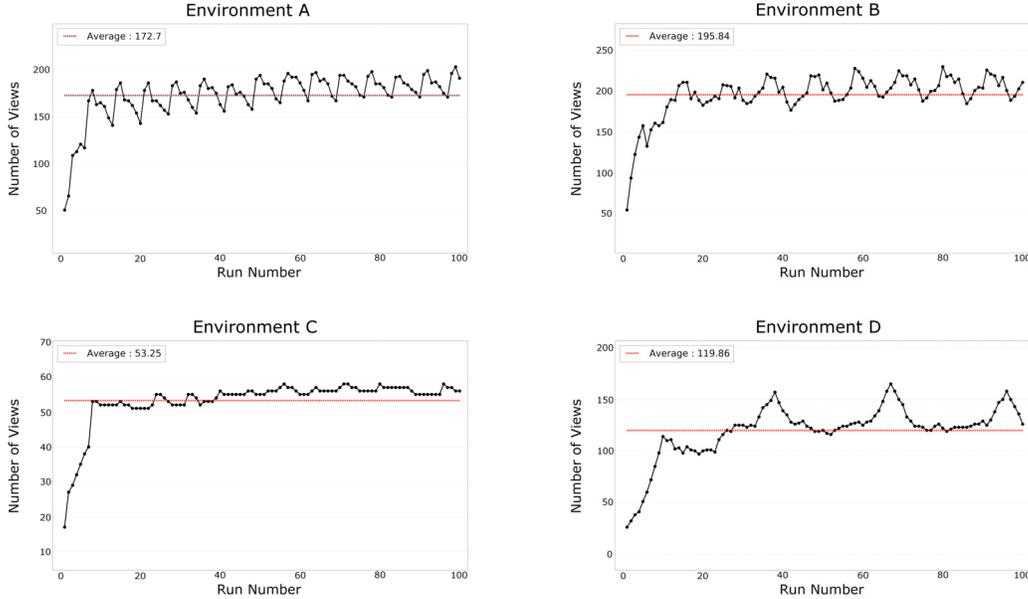


Fig. 8: The growth of views over 100 runs in the four environments from Fig. 7. The total number of views stabilize after 5 to 10 runs.

nearest neighbor constraint is needed to ensure a good spatial distribution (see Fig. 6) of views across the entire map. Without this additional constraint, maps can end up with an uneven spatial distribution of views (see Fig. 5).

A small voxel size will mean that a view might end up having too few neighbors, and a high nearest neighbor threshold will mean too many minimum neighboring views. In either case, the nearest neighbor constraint would restrict pruning. A uniform spatial distribution would be maintained but the growth rate of views would be high. Whereas a very large voxel size will mean too many neighbors, and a small nearest neighbor threshold will restrict the number of views. In these scenarios, the nearest neighbor constraint wouldn't ensure a uniform spatial distribution but would restrict the growth rate. So, the choice of the nearest neighbor voxel parameters and the nearest neighbor threshold needs to be carefully chosen so as to have a good uniform distribution and a reasonable growth rate.

We looked at the relocalization distance and the growth rate metrics to determine the appropriate nearest neighbor voxel size and threshold. In Table II, the view score is obtained using  $W_1 = 1.5$ ,  $W_2 = 1$ ,  $W_3 = 3$ , and a view score threshold of 1.375. Nearest neighbor threshold is varied from

TABLE II: Nearest neighbor threshold selection

NN_THRESH	VOXEL_PARAMS (x, y, theta) [m, m, rad]	Relocalization Distance [m]	Growth Rate
1	1, 1, 1	13.78	4.05
1	1, 1, 2	14.615	3.15
<b>1</b>	<b>2, 2, 1</b>	<b>9.349</b>	<b>4.2</b>
1	2, 2, 2	11.256	3.95
<b>3</b>	<b>1, 1, 1</b>	<b>9.183</b>	<b>4.45</b>
<b>3</b>	<b>1, 1, 2</b>	<b>11.583</b>	<b>4.05</b>
3	2, 2, 1	9.417	3.55
3	2, 2, 2	10.409	4
5	1, 1, 1	11.527	5
<b>5</b>	<b>1, 1, 2</b>	<b>9.658</b>	<b>4.4</b>
5	2, 2, 1	10.841	4
5	2, 2, 2	12.939	3.95

1 to 5, and the voxel size is varied from (1m, 1m, 1 rad) to (2m, 2m, 2 rad). The relocalization distance and the growth rates are compared with the values from using just the view score without the nearest neighbor constraint. The average relocalization distance is 11.78m and growth rate is 4.25 without using the nearest neighbor constraint. Growth rates within a tolerance of 0.2 (4.05 to 4.45) and a lower relocalization distance ( $\leq 11.78m$ ) is used for selecting the

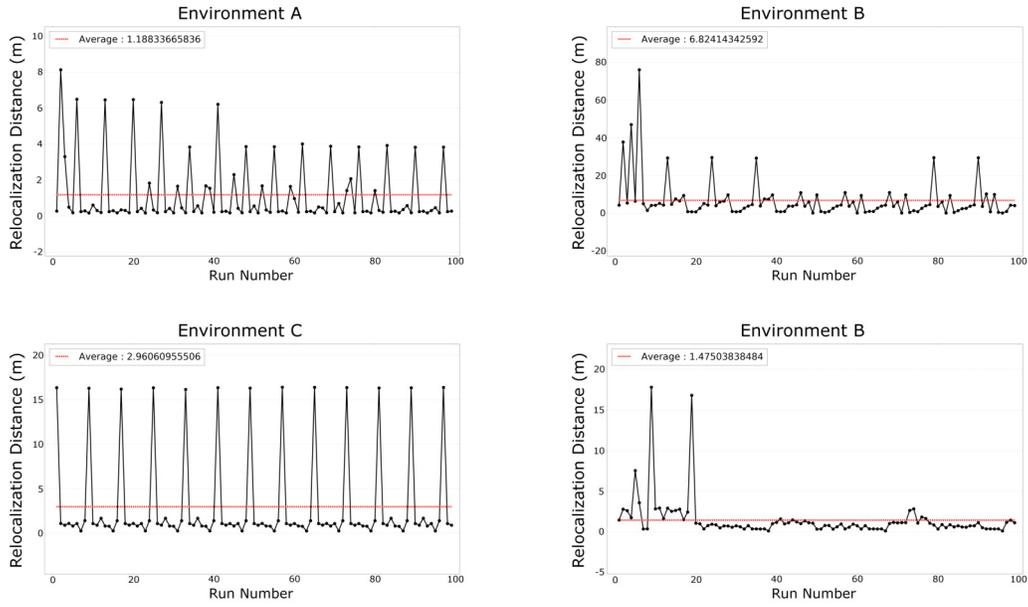


Fig. 9: Plot of the relocalization distance, the lower the better. The relocalization distance is usually low in most cases, and in some maps there are large spikes in the beginning which is due to the system taking time to observe views from a loaded map due to very different lighting conditions.

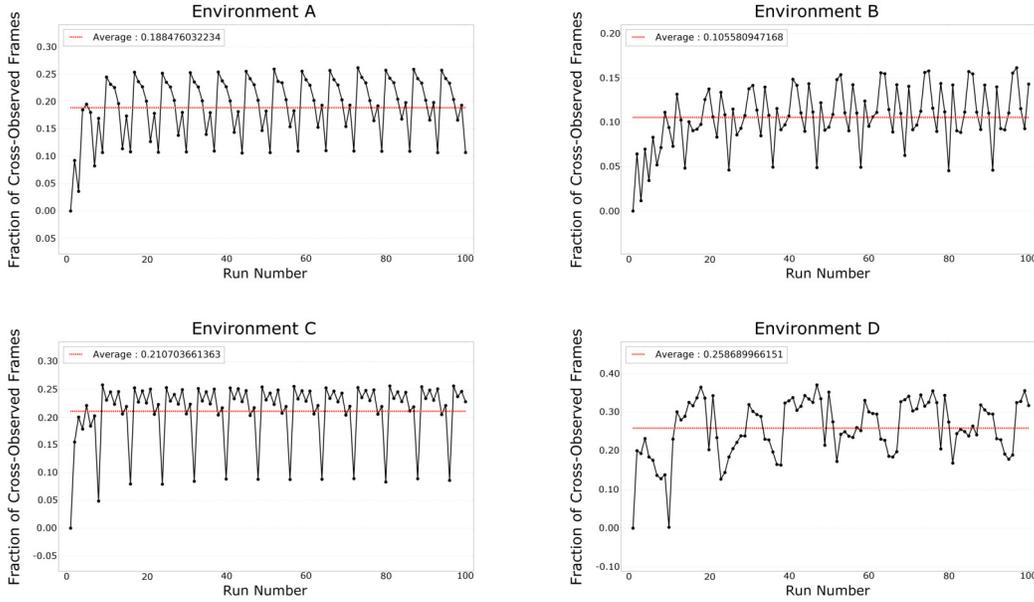


Fig. 10: Plot of the fraction of cross observed frames, the higher the better. This plot gives an idea of how the fraction of cross observations change over multiple runs with the view management system in place.

parameters. Using these, we find (1, 2m, 2m, 1 rad), (3, 1m, 1m, 1 rad), (3, 1m, 1m, 2 rad), and (5, 1m, 1m, 2 rad) to be suitable nearest neighbor threshold and voxel size candidates.

#### D. Lifelong mapping runs

We show the results of the proposed view management algorithm by running it on four different environments (see Fig. 7) for 100 runs (see Fig. 8, 9, 10, 11). The weights used for computing the view score were (1.5, 1, 3) with a score threshold of 1.375, and a nearest neighbor voxel size of (1m, 1m, 2 rad) with a nearest neighbor threshold of 5. A minimum views threshold was set at 25, as the system

can easily handle a low number of views and pruning is not necessary. The periodicity that is seen in some of the plots is due to cycling of some of the logs during the sequential 100 runs. For these environments, and in general environments below  $1000 \text{ ft}^2$ , we have seen the total number of views usually stabilize within 300 with our algorithm. Without running our algorithm, the total number of views can go up to 1500 views increasing the CPU usage while observing views. The runtime of the algorithm on our mobile robot platform is around 300 ms with a system containing up to 500 views on a 1.2 GHz quad core cellphone grade processor.

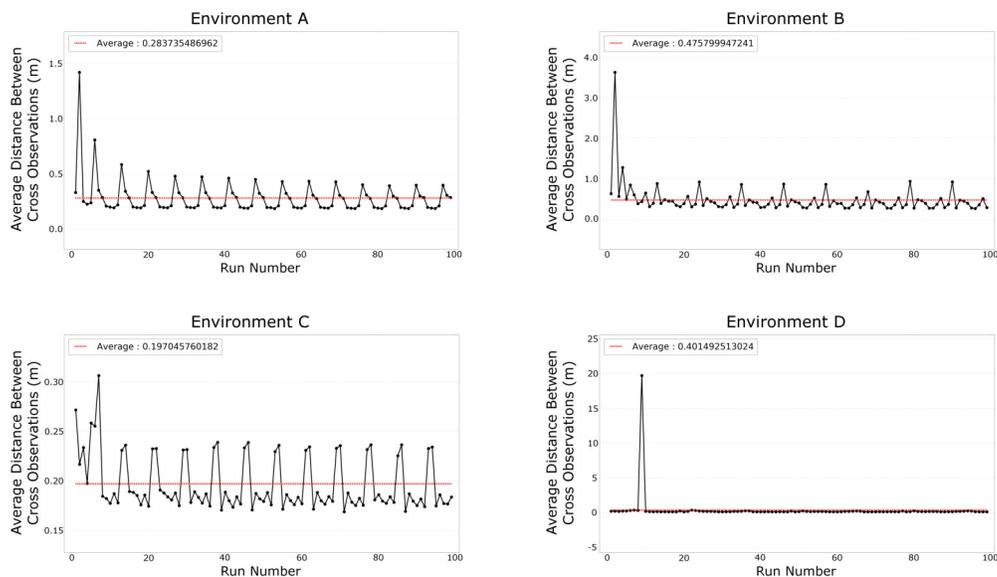


Fig. 11: The average distance between cross observations, the lower the better. In the four environments, it starts off a little higher and then it makes observations from views from the loaded map every 0.5m. In environment D, the average distance between cross observations for the 9th run is very high because the lighting conditions were drastically different from the previous runs and there were only a handful of views created in earlier runs that the robot could observe in the 9th run. Once the robot created new views in that lighting condition, it didn't have any problems thereafter.

## VI. CONCLUSIONS AND FUTURE WORK

In this paper we have addressed the problem of view management in a visual SLAM system, which arises during lifelong visual mapping on a resource-constrained mobile robot platform. We have presented an efficient and robust algorithm for deciding which views can be removed. We have also devised formal criteria for measuring how well the robot is able to relocalize. We have then used them to demonstrate experimentally that our algorithm is capable of limiting the growth of the number of views in the SLAM system without compromising its ability to relocalize, despite changes in the appearance of the environment.

We have described the procedure we used for tuning the algorithm's parameters. One possible future research direction is to apply machine learning techniques to automate that process, and possibly improve performance further.

## VII. ACKNOWLEDGEMENTS

We would like to thank Philip Fong, Renaud Moser, Martin Llofriu, and Emily Pittore for their advice, and their work on the overall system.

## REFERENCES

- [1] E. Eade, P. Fong, and M. E. Munich, "Monocular graph SLAM with complexity reduction," *IEEE IROS*, pp. 3017–3024, 2010.
- [2] K. Konolige, J. Bowman, J. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua, "View-based maps," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 941–957, 2010.
- [3] "iRobot's new Ava 500 puts robotics in heart of the enterprise," <https://www.forbes.com/sites/jenniferhicks/2013/06/10/irobots-new-ava-500-puts-robotics-in-heart-of-the-enterprise/#3506cd972e1b>, accessed: 2018-09-14.
- [4] K. Konolige and J. Bowman, "Towards lifelong visual maps," in *IEEE IROS*, 2009, pp. 1156–1163.
- [5] N. Sünderhauf and P. Protzel, "Towards a robust back-end for pose graph slam," in *IEEE ICRA*, May 2012, pp. 1254–1261.

- [6] F. Dellaert and M. Kaess, "Factor graphs for robot perception," *Foundations and Trends in Robotics*, vol. 6, no. 1-2, pp. 1–139, 2017. [Online]. Available: <http://dx.doi.org/10.1561/23000000043>
- [7] D.-N. Ta, N. Banerjee, S. Eick, S. Lenser, and M. Munich, "Fast nonlinear approximation of pose graph node marginalization," in *IEEE ICRA*, May 2018.
- [8] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *VISAPP International Conference on Computer Vision Theory and Applications*, 2009, pp. 331–340.
- [9] S. Hochdorfer, M. Lutz, and C. Schlegel, "Lifelong localization of a mobile service-robot in everyday indoor environments using omnidirectional vision," in *IEEE International Conference on Technologies for Practical Robot Applications (TEPRA)*, November 2009.
- [10] W. Hartmann, M. Havlena, and K. Schindler, "Predicting matchability," in *IEEE CVPR*, June 2014, pp. 9–16.
- [11] S. Buoncompagni, D. Maio, D. Maltoni, and S. Papi, "Saliency-based keypoint selection for fast object detection and matching," *Pattern Recognition Letters*, vol. 62, pp. 32–40, 2015.
- [12] M. Dymczyk, T. Schneider, I. Gilitschenski, R. Siegwart, and E. Stumm, "Erasing bad memories: agent-side summarization for long-term mapping," in *IEEE IROS*, October 2016.
- [13] P. Mühlfellner, M. Bürki, M. Bosse, W. Derendarz, R. Philippsen, and P. Furgale, "Summary maps for lifelong visual localization," *Journal of Field Robotics*, vol. 33, no. 5, pp. 561–590, 2015. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21595>
- [14] M. Bürki, I. Gilitschenski, E. Stumm, R. Siegwart, and J. Nieto, "Appearance-based landmark selection for efficient long-term visual localization," in *IEEE IROS*, Oct 2016, pp. 4137–4143.
- [15] M. Dymczyk, S. Lynen, M. Bosse, and R. Siegwart, "Keep it brief: Scalable creation of compressed localization maps," in *IEEE IROS*, Sep. 2015, pp. 2536–2542.
- [16] W. Churchill and P. Newman, "Practice makes perfect? managing and leveraging visual experiences for lifelong navigation," in *IEEE ICRA*, May 2012, pp. 4525–4532.
- [17] —, "Experience-based navigation for long-term localisation," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1645–1661, 2013. [Online]. Available: <https://doi.org/10.1177/0278364913499193>