Preventing and Correcting Mistakes in Lifelong Mapping

Nandan Banerjee*, Dimitri Lisin, Victoria Albanese, Zhongjian Zhu, Scott R. Lenser,
Justin Shriver, Tyagaraja Ramaswamy, Jimmy Briggs, and Phil Fong

***Abstract*— A Graph SLAM system is only as good as the edges in its pose graph. Critical mistakes in the generation of these edges can instantly render a map inconsistent, misleading, and ultimately unusable. For a lifelong mapping system, where the map is updated continuously, avoiding these errors altogether is infeasible. Instead, we propose a system for detection of and recovery from severe errors in edge generation. Our system remedies both edges created by view observations and edges created by an odometry motion model. For observation edges, we pair a novel method for monitoring ambiguous views with an intelligent graph-merging algorithm capable of rejecting a relocalization in progress. For motion edges, we propose a qualitative geometric approach for detecting structural aberrations characteristic of odometry failures. We conclude with an analysis of our results based on an empirical study of thousands of robot runs.**

I. Introduction

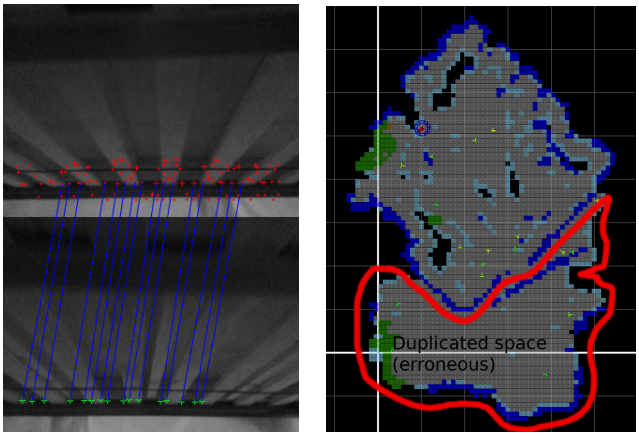
A vision-based Simultaneous Localization and Mapping (SLAM) system is typically partitioned into a front-end which recognizes *views* of the environment and a back-end which maintains a map of the traversable space. In the particular case of a graph SLAM system, this back-end maintains a directed graph where nodes represent unknown robot poses, and edges encode constraints between them. Nonlinear optimizations on this pose graph lead to robust estimates of the structure of the environment and the location of the robot.

The edges in the pose graph contain all known information about the physical layout of the robot's trajectory. In general, there are two types of pose graph edges. *Motion edges* generated from odometry describe the robot's motion between two poses. *Observation edges* denote the re-observation of a known view by the front-end, forming what are often called "loop closures" in the pose graph.

Large mistakes in the generation of pose graph edges can lead to inconsistent maps and nonfunctional navigation on a mobile robot. If the robot builds a new map on every run, these errors will be confined to their originating run, mitigating the damage they may cause. However, in a lifelong mapping scenario, where the robot updates and reuses its pose graph in each run, pose graph errors can permanently disable the robot's navigation.

In this paper we consider a wheeled, mobile, home robot which employs a monocular graph-based SLAM navigation system [1]. The robot's front-end uses camera data to generate and recognize views. Re-observations of views are incorporated into the back-end's pose graph as observation edges. The motion edges of the pose graph are generated from odometry measurements taken by wheel encoders. While the SLAM system tolerates normal sensor noise well, both the camera and wheel encoders occasionally produce critical errors that can cause mapping and navigation failures during lifelong mapping.

*The authors are with the iRobot Corporation, Bedford, MA 01730, USA. {nbanerjee, dlisin, valbanese, zzhu, slenser, jshriver, tramaswamy, jbriggs, pfong}@irobot.com



(a) (b)

Fig. 1: **Ambiguous views** (a) The bottom of a bed frame looks the same viewed from opposite sides (top image - view from one side, bottom image - view from another side, points and lines - represents matches by our detector). (b) These ambiguous views can cause erroneous edges to be added to the pose graph. Addition of an erroneous edge in this case made the occupancy map become very bad: occupancy cells within the red boundary is duplicated space which is not there.

Critical motion edge failures occur when the robot's wheels lose solid contact with the ground. This failure state arises in certain natural configurations of furniture and indoor architecture such as when a moderate upward force nullifies the robot's hold on a plush carpet. These motion edge errors are small at first, but accumulate until the robot's wheels regain their traction. This often results in long erroneous protrusions in the map where the wheels have reported movement but the robot has not actually moved.

Observation edges have more complex modes of critical failure. For example, similar-looking objects in the environment (Fig. 1a) can cause incorrect observations, which in turn become incorrect observation edges in the graph. Such edges will cause inconsistencies in the occupancy map (Fig. 1b), adversely affecting the robot's behavior.

Another critical failure mode for a home robot's observation edges follows from its size. Because home robots are

typically small, they can be easily picked up and moved. We call this phenomenon "kidnapping." A kidnapped robot does not know where it is, so it must assume that it is in a brand new space and create a new separate component in its pose graph. We use the term **Component** to refer to one fully connected graph and the term **Graph** to denote one or a collection of components that are disjoint. As it maps the space, the robot may recognize a known view, indicating that it is actually in a previously mapped space. In this instance, the robot will attempt to *merge* its current component representing the new space with the component corresponding to the original space.

Conversely, similar objects in separate components may be mistaken for each other by the front end. When the front-end recognizes a view from the wrong component, it will trigger the addition of an invalid observation edge between separate pose graph components. This invalid new edge may merge the previously separate components of the pose graph, causing even more damage to the occupancy map.

Our goal in this paper is to provide three new algorithms which allow the SLAM system to detect and recover from critical edge-creation errors in the SLAM pose graph:

- An algorithm for determining which views are ambiguous, and therefore likely to cause incorrect localization.
- An extension to the SLAM system which allows the robot to re-localize conditionally, with the option to back out of the re-localization in the absence of additional evidence.
- An algorithm for detecting erroneous protrusions in the occupancy map, typically caused by wheel slippage.

II. Related Work

The graph-based formulation for SLAM was first proposed in the seminal work by Lu and Milios [2], and it was subsequently developed to allow much more efficient algorithms [3], [4], [5], [6], [7]. There has been some significant research in graph-based SLAM to improve the robustness, and time and space complexity of front-end and back-end. The emphasis on the accuracy of the pose graph, including the corner cases, has increased with the requirement of robust map representation [8], [9] and segmentation [10] and efficient motion planning algorithms [11], [12]. Current SLAM approaches [13], [14], [15], [16], [17] that build upon the previous research try to make the algorithm more robust by taking relocalization into account.

Several of these frameworks have been tested for specific robot kidnapping situation [18]. ORB-SLAM2 [13] goes into relocalization mode after the kidnapping, but fails to get back to the mapping mode. On the other hand, ProSLAM [15], goes into the relocalization stage and merges the map.

Jensfelt et al. [19] proposed a localization approach using multi-hypothesis tracking using Kalman Filters. Lee et al. [20] proposed a computer vision based mechanism for recovery from kidnapping for home cleaning robots. It focuses on detecting a kidnap and then localizing the robot using a single image and the features from the existing map. The feature matching is done at a coarse level to limit the search area and then at the finer level for accurate pose estimation. The robot's pose is estimated from these matches and their corresponding poses are estimated using a recursive EKF process until it meets the convergence criteria.

In contrast, our framework requires multiple observations of multiple views to relocalize. Furthermore, we propose a mechanism for a conditional localization, pending additional evidence.

Another problem in lifelong mapping that often gets unnoticed is large wheel slippage, causing erroneous map protrusions and affecting the re-usability of maps. Extensive work has been done to incorporate wheel slippage into the dynamic model of the wheeled mobile robot. Various approaches to incorporating wheel slippage into the robot's kinematics model have been proposed [21], [22], [23], [24], [25]. To increase robustness, Reina et al. [26] describe a system using three separate methods for detecting slip in an ensemble. Palacin et. al [27] use an entire secondary odometry system based around an optical sensor to counteract the effects of slippage.

Unfortunately, these methods are not foolproof, and, as we have mentioned in previous sections, the repercussions of a failure of a slip/stasis detector can be huge. In this paper, we propose to detect and remove erroneous protrusion after the fact, thus improving localization, navigation, and mapping on the robot.

Our SLAM system is built on the previous works [1], [28], [29]. We add the following extensions to the system: a method for detecting ambiguous views that could potentially lead to incorrect localization, an ability to conditionally relocalize or back out of a relocalization deemed uncertain, and a method for detecting erroneous protrusions in the occupancy maps.

The rest of the paper is structured as follows. Section III describes our system, including the re-localization and recovery mechanism. Section IV presents the methods for detecting ambiguous views, conditional re-localization, and detecting and removing erroneous protrusions. We show experimental results validating our methods in section V.

III. System Overview

A. SLAM System

Our SLAM system can be divided into the front-end and the back-end [1]. The back-end maintains a pose graph, where nodes are robot poses, and edges are pairwise constraints between them.

The front-end's first job is to use images from the camera to create views, which are 3D point clouds, with associated image descriptors. Each view is associated with a node in the pose graph. The front-end's second job is to observe previously created views, and calculate robot's pose relative to them. Observing a view creates an edge in the pose graph between the robot's current node and the view's node, resulting in a loop closure. The loops allow the graph to be optimized, yielding most likely pose estimates, and correcting errors in odometry.

While the pose graph allows the robot to keep track of its position, it does not help it to plan its path. For that we use an occupancy representation consisting of many overlapping local occupancy maps attached to nodes in the graph [8]. The graph, and various entities associated with its nodes, such as views, local maps, as well as semantic information such as room boundaries [30], make up the robot's *map*.

B. Hypothesis Filter

When a home robot is kidnapped, it will lose the information of its present location. To cover all cases, the robot can simultaneously build a new pose graph component from scratch and search for views from prior components. Suppose our robot has mapped a single space as a pose graph component C_a. When it is kidnapped, the robot will create a new, empty component C_b which will be used for localization and mapping. If the robot later observes a view corresponding to a previous observation in C_a, it can localize itself in C_a by adding an edge between its current node in C_b and the node for the previously observed view in C_a. This results in a merge between the connected components.

Unfortunately, a view observation can be incorrect. The front-end can mistake similar-looking features of the environment for each other. Even after observing the correct view, errors in pose estimation can arise from image noise, motion blur, and changes in illumination. Therefore, merging C_a and C_b, based on a single observation is risky. Instead, if the robot observes a view from a prior component, we form a *merge hypothesis* that the two components should be joined, but delay the actual merging of the components until we accumulate enough evidence for the merge hypothesis.

Similarly, the robot may detect a new view in its current pose graph component which contradicts its current pose estimate. This may result either from an incorrect view observation or from the accumulation of odometry errors. As before, adding an observation edge immediately risks destruction of the map. The robot would be wise to turn the view observation into a *recovery hypothesis*, and wait to see if more evidence for it presents itself.

The *hypothesis filter* (Fig. 2) is our method for managing merge and recovery hypotheses. The hypothesis filter represents each hypothesis as an extended Kalman Filter (EKF), whose state is updated by robot motion and view observation. We accept a hypothesis by adding edges to the SLAM graph for view observations that support it. For a merge hypothesis, new edges are all between the same two disjoint connected components, whereas for a recovery hypothesis, the edges are all within a single connected component.

Let $\mathcal{N}(z, \Sigma_z)$ denote a landmark observation in SE(2), which is the transformation from the landmark node to the current robot node in the SLAM pose graph.

$$z = {}^{robot}T_{lm} \tag{1}$$

When the first observation z_0 comes across, a new hypothesis is created, with the initial state $\mathcal{N}(G_{0|0}, \Sigma_{G_{0|0}})$ initialized as

$$\begin{aligned} G_{0|0} &= z_0 \\ \Sigma_{G_{0|0}} &= \Sigma_{z_0} \end{aligned} \tag{2}$$

As the robot moves in its environment, the hypothesis state $< G, \Sigma_G >$ is updated with the new robot motion $T_{i|i-1}$

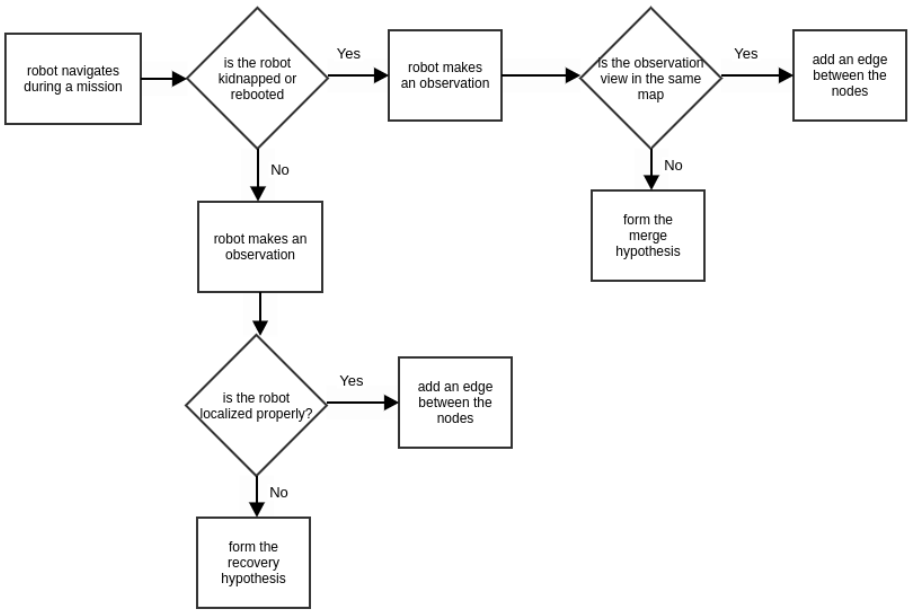


Fig. 2: Hypothesis Filter Flowchart.

through an EKF motion update.

$$\begin{aligned} G_{i|i-1} &= T_{i|i-1} \cdot G_{i-1|i-1} \\ \Sigma_{G_{i|i-1}} &= \mathrm{Adj}_{[T_{i|i-1}]} \cdot \Sigma_{G_{i-1|i-1}} \cdot \mathrm{Adj}^{\top}_{[T_{i|i-1}]} + \Sigma_{T_{i|i-1}} \end{aligned} \tag{3}$$

Given the updated state $G_{i|i-1}$ and the observation Z_i, the residual (innovation) R_i is given by

$$\begin{aligned} R_i &= z_i \cdot G^{-1}_{i|i-1} \\ \Sigma_{R_i} &= \mathrm{Adj}_{[R_i]} \cdot \Sigma_{G_{i|i-1}} \cdot \mathrm{Adj}^{T}_{[R_i]} + \Sigma_{z_i} \end{aligned} \tag{4}$$

The measurement update combines the motion estimates of $G_{i|i-1}$ and the observation z_i to produce a refined motion $G_{i|i}$. This is similar to edge combination in [1]. The Kalman gain is given by

$$K_{i|i-1} = \Sigma_{G_{i|i-1}} \cdot (\Sigma_{G_{i|i-1}} + \Sigma_{z_i})^{-1} \tag{5}$$

Using the Kalman gain $K_{i|i-1}$, the refined state is given by

$$\begin{aligned} \Sigma_{G_{i|i}} &= (I - K_{i|i-1}) \cdot \Sigma_{G_{i|i-1}} \\ &= \Sigma_{G_{i|i-1}} - \Sigma_{G_{i|i-1}} \cdot (\Sigma_{G_{i|i-1}} + \Sigma_{z_i})^{-1} \cdot \Sigma_{G_{i|i-1}} \\ &= (\Sigma^{-1}_{G_{i|i-1}} + \Sigma^{-1}_{z_i})^{-1} \\ G_{i|i} &= G_{i|i-1} \oplus (K_{i|i-1} \cdot \ln(R_i)) \\ &= \exp(K_{i|i-1} \cdot \ln(R_i)) \cdot G_{i|i-1} \\ &= \exp(\Sigma_{G_{i|i}} \cdot \Sigma^{-1}_{z_i} \cdot \ln(R_i)) \cdot G_{i|i-1} \end{aligned} \tag{6}$$

As new observations come in, the residuals R of the hypotheses present in the hypothesis filter are first computed (Eqn. 4). If the residual magnitude in Mahalanobis distance is above a pre-determined threshold for all the present hypotheses in the system, then a new hypothesis is created. If not, then the observation is used in the measurement update step for the conforming hypothesis.

There is still the question of when a hypothesis should be accepted. We do this simply by setting a threshold on the number of observations contained in the hypothesis, and the number of views that were observed. If a particular hypothesis meets these criteria, then we accept it, and add the corresponding edges to the graph.

It is also helpful to keep the number of hypotheses from becoming too many. We impose an expiration criterion: if no observations have been added to a hypothesis for some period of time, we delete it. Other criteria can also be used, for e.g., distance traveled by the robot while no observations were added to a hypothesis. Additionally, when we accept a hypothesis, we delete all the other competing hypotheses.

IV. Map Corruption Prevention

A. Ambiguous Views

We present an ambiguous view detector on top of the hypothesis filter to help prevent accepting poor recovery hypotheses. A view is defined as ambiguous if in a robot run, that particular view is observable from multiple locations in an environment leading to rejected observations from the SLAM back-end, while at the same time the robot is certain about its positional estimate w.r.t a recently accepted "trusted" view observation. For a given view V_y, view V_x is "trusted" if V_x was created in the system before V_y. Views marked as ambiguous are discounted in the recovery hypothesis filter. These views can also be discounted from the merge hypothesis filter. Fig. 1 shows an example of ambiguous views and map corruption due to bad recovery hypothesis acceptance.

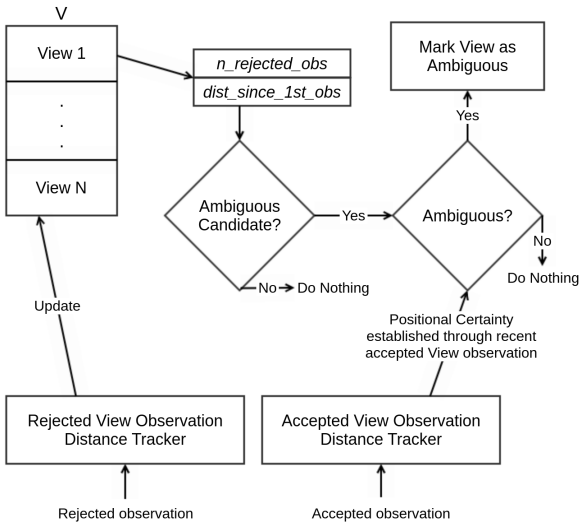


Fig. 3: Ambiguity detection flow chart.

The ambiguity tracker (Fig.3) contains two observation **distance trackers** - a rejected view observation distance tracker, and an accepted view observation distance tracker. The distance tracker (DT) keeps track of the distance the robot has traversed since a particular observation was made. Observations in the tracker that exceed a particular distance threshold are removed. Accepted observations go into the accepted observation DT, and rejected observations go into the rejected observation DT.

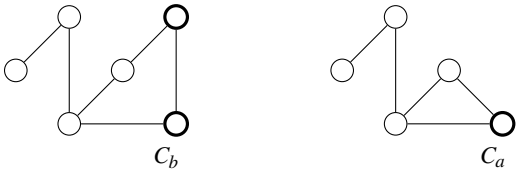
The rejected observation DT updates a structure V containing views with their number of rejected observations and distance since the first observation periodically. For all views present in V, a check is performed with a preset threshold criteria based on the number of rejected observations and distance since the 1st observation to filter out **ambiguous view candidates**. The more the number of rejected observations for a particular view, the more the chances of that view being ambiguous.

For all ambiguous view candidates, a second check is performed to determine if they are **ambiguous views**. This is done with the aid of the accepted observation distance tracker which provides information about "trustworthy" (as defined earlier) views to ascertain positional certainty of the robot. The accepted view distance tracker is used to keep track of recent observations of trustworthy views. Once a view is determined to be ambiguous, it is marked as such so that it is discounted in the recovery / merge hypothesis filter.

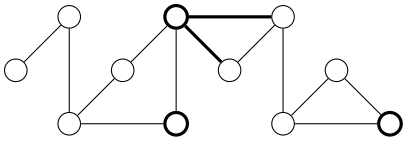
B. Conditional Merge

Acceptance criteria for a merge hypothesis present a trade-off. If they are too permissive, the robot may localize into a wrong component, or it may localize into the right component, but in the wrong place. If the acceptance criteria are too strict, then the robot may take a long time to localize, during which it will not be able to perform many tasks, such as navigating to a specific room.

(a) Original graph G with two separate connected components



(b) Copy of the graph G', with the two components conditionally merged after observing one view twice.



(c) Original graph G with the two components fully merged after observing two views.

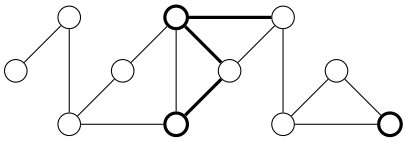


Fig. 4: **Conditional merge.** Let us say we require observing one view for a conditional acceptance of a hypothesis, and two views for full acceptance. Part (a) shows a SLAM graph G consisting of two connected components: C_a, corresponding to the current new component and C_b corresponding to a previously existing component. The view nodes are shown as thick circles. Part (b) illustrates a conditional acceptances after the robot observes one view in C_b twice. It shows G', the copy of G, with two added observation edges from the conditionally accepted hypothesis (thick lines). Part (c) illustrates a full acceptance after observing two views in C_b. It shows the new observation edges added to the original graph G

We propose to get the best of both worlds by having two sets of acceptance criteria: a permissive set for a *conditional* merge of the two components, which can be undone, and a strict set for a full merge, which cannot be undone. The idea is for the robot to be able to conditionally merge two components quickly, and start behaving as though it is localized, while continuing to collect additional evidence for the merge. If sufficient evidence for full acceptance presents itself, the conditional merge becomes a full merge. Otherwise, we undo the conditional merge.

This is easier said than done, because once two SLAM graph components C_a and C_b are joined, they cannot be easily untangled. The merged graph will be optimized as a whole, affecting the poses of the robot, the views, the occupancy map, and other entities associated with its nodes. Even if we were to keep track of the new edges used to join the connected components, simply removing them would not bring C_a and C_b to their original state.

To get around this problem, we make a copy of the entire SLAM graph. Let G be the SLAM graph, where $C_a \subset G$ and $C_b \subset G$ (Figure 4a). If we have a merge hypothesis h_c, which is conditionally accepted, then we create G', which is a copy of G, and where C'_a and C'_b are copies of C_a and C_b respectively. We add the edges from h_c into G', joining C'_a and C'_b (Figure 4b). Now, we will use G' for read-only operations, such as looking up the robot pose, or path planning, making the robot behave as though it is localized in the map C_b. However, we still perform all the write operations, such as adding new nodes and edges, and optimization on G, to keep C_a and C_b consistent with their respective original states. Every time G is modified, we copy it into G' and add the edges from h_c, merging C'_a with C'_b.

To fully accept h_c, we simply add the edges corresponding to its observations to G (Figure 4c). On the other hand, to undo the conditional merge, all we have to do is stop adding the edges from h_c to G'. If there are multiple merge hypotheses present, only one merge hypothesis is allowed to be in a conditionally merged state.

C. Erroneous Protrusion Detection

We present a detection mechanism for erroneous protrusions caused due to a failure to detect slip / stasis in a robot. This detector works in the occupancy grid map space.

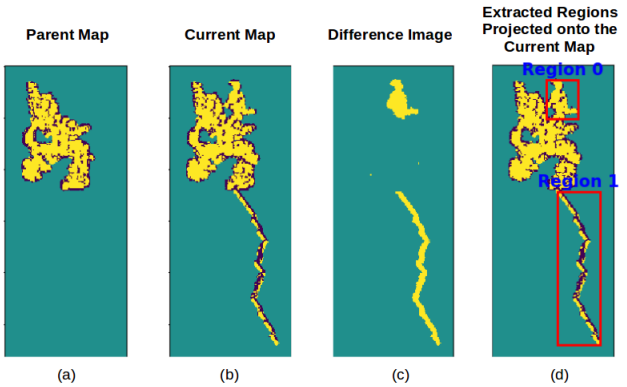


Fig. 5: **Erroneous Protrusion Detection.** (a) Initial version of the map. (b) Updated map after run, with new space and protrusion. (c) Image difference between map *(a)* and *(b)* detects newly added areas. (d) Classifier detects region 0 as new space and region 1 as erroneous protrusion.

The detector compares a snapshot of the map at the start of a robot's run (parent map), and a snapshot of the map at present time (current/child map) by performing an image difference operation to segment out the "new" space. The segments are labeled, and run through an erroneous

Fig. 6: **Recovery hypotheses statistics.** Most robot runs contain recovery hypotheses, but the vast majority is rejected.

Fig. 7: **Merge hypotheses statistics.** In the majority of runs the robot had to relocalize in the map. 18.6% of the merge hypotheses were conditionally accepted and later rejected, preventing possible map corruption.

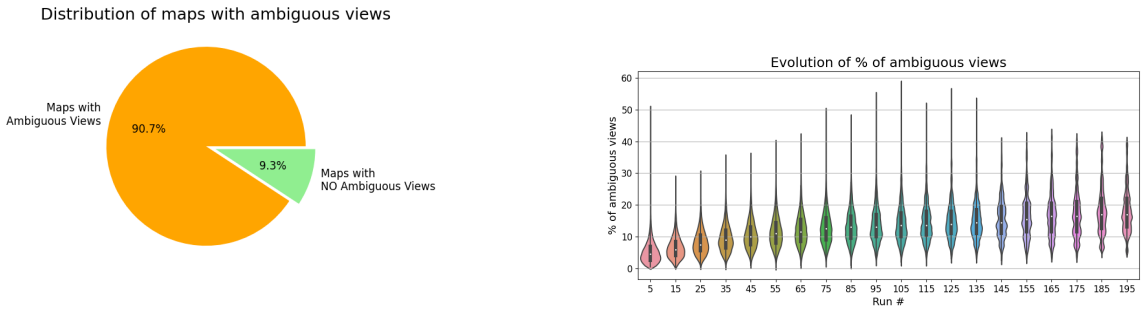


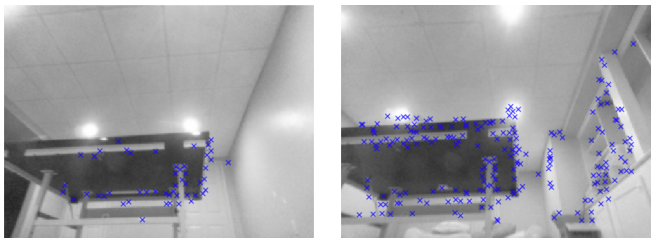
Fig. 8: **Ambiguous views statistics.** A vast majority of maps contain ambiguous views. The typical number of views per map is about 15-18%.

protrusion classifier to determine whether the segment is actually new space discovered in the environment, or an erroneous protrusion (Fig.5). Detected protrusions can be removed from a map, or the map forgotten and the robot can go back to using the parent map.

Feature Selection, Training and Classification: Training data consisting of a mix of parent and child maps with new regions and erroneous protrusions are collected. Regions are segmented and labeled as new regions or protrusions. A set of geometric features such as the average distance transform, bounding box area, area of region, major axis length, minor axis length, and perimeter are computed for each segmented region. For feature selection, ANOVA F-values and their corresponding p-values are computed for all of the features; individual features are looked at as independent variables and can also be augmented with other feature variables. Features with p-values less than 0.05 are selected. Classic machine learning methods such as decision trees, random forests, or support vector machines (SVM) can be used.

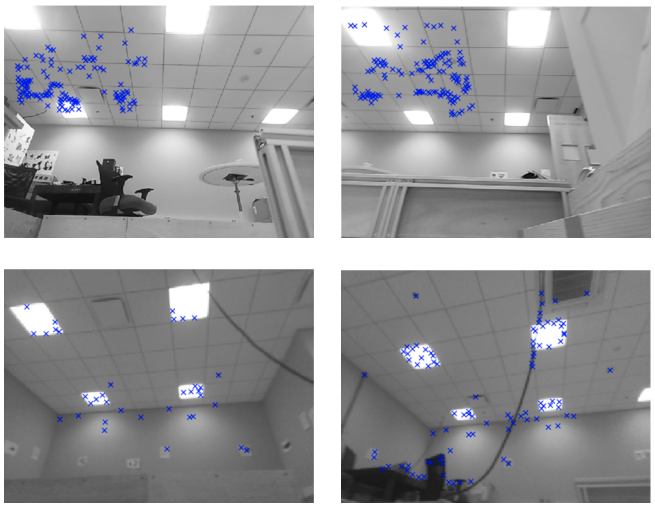
V. Experiments and Results

In our pursuit to ensure prevention and correction of mistakes due to poor slip / stasis detection, scene ambiguity, and bad relocalization in people's homes, we realized that it is infeasible to simulate all the possible scenarios in a laboratory setting or simulated environments. Instead, we rely on experimental data containing ground truth information for



(a) (b)

Fig. 9: **Ambiguous views** The ping pong table looks the same viewed from opposite sides. (a) is the view and (b) is an observation. This ambiguous view (a) was detected by our ambiguity tracker and marked as ambiguous.



(a) (b)

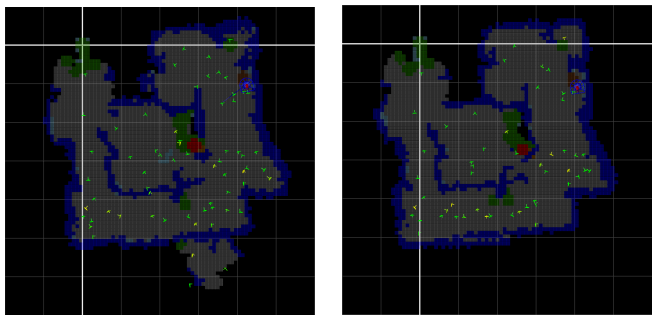
Fig. 10: **Ambiguous views** The office ceilings look very similar viewed from opposite sides. (a) denote the views and (b) denote observations. These ambiguous views (a) were detected by our ambiguity tracker and were marked as ambiguous.

tuning / training parameters, and for overall global statistics regarding the performance of our proposed algorithms, we rely on analyzing robot telemetry from thousands of robots all over the world.

In Fig. 6, we show the number of recovery hypotheses generated in 10,000 robot runs in different household environments. A vast majority of runs result in at least one recovery hypotheses created, of which only 1.1% are accepted.

Overall, the robot started a run localized 43.2% of the time, the rest of the time, the robot had to relocalize, either because of a kidnap or a reboot (Fig. 7). Note that 18.6% of merge hypotheses were accepted conditionally, and later rejected, thus preventing potential poor relocalizations.

For detecting ambiguous views, we tuned our ambiguity tracker parameters based on a sampling of robot runs with both good and bad accepted recovery hypotheses. We ran the ambiguity tracker on the handful of real world samples that we had collected containing ambiguous views - bottom of the bed frame (Fig. 1), ping pong table (Fig. 9), office ceilings (Fig. 10), etc and saw a remarkable drop in corrupted maps



(a) (b)

Fig. 11: **Map corruption prevention using ambiguity tracker.** (a) Occupancy map with an erroneous protrusion at the bottom without our ambiguity tracker, caused by adding edges resulting from erroneous observations. (b) With our ambiguity tracker, view from Fig. 9a was marked as ambiguous and the occupancy map corruption was prevented.

from those environments. Fig. 9 shows ambiguous views detected by our ambiguity tracker, and Fig. 11 shows a map corruption prevention due to a rejected bad recovery hypothesis because of additional ambiguous view information.

We have analyzed ambiguous view detection in 500,000 maps. Fig. 8 (left) shows that there were almost 90.7% maps containing ambiguous views. Over multiple runs the total number of ambiguous views per map hovers around 15%-18% (Fig. 8 (right)).

In our system, we used a decision tree classifier using the average distance transform, area, and minor axis length features to classify protrusions/new space. Out of 500,000 maps, the erroneous protrusion detector found abnormal protrusions in 4200 maps, or 0.92%. A sampling of 1352 maps containing detected protrusions (1208) and detected new space (144) was user labeled and compared. The confusion matrix (Fig. 12) shows the recall of protrusion detection was 94.05%, and the precision for protrusion detection was 99.58%, with a false positive rate of 0.42%.

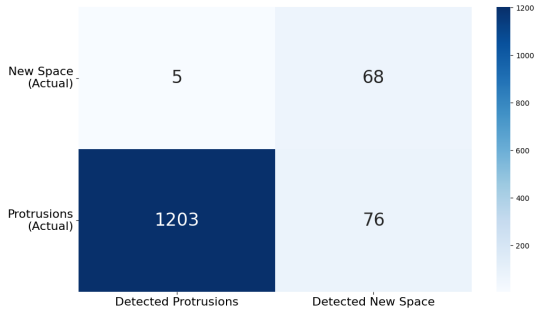


Fig. 12: **Erroneous Protrusion Detector confusion matrix.** We have evaluated our method on 1352 maps, where erroneous protrusions and actual newly discovered space were hand-labeled.

VI. Conclusions

We have proposed two techniques for preventing map corruption: ambiguous view detection and conditional localization. We have also presented a method for correcting

mistakes in the map after fact by detecting and removing erroneous protrusions in the occupancy grid. We have demonstrated the efficacy of these methods in keeping the maps usable in lifelong mapping scenarios.

REFERENCES

[1] E. Eade, P. Fong, and M. E. Munich, "Monocular graph SLAM with complexity reduction," *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 3017–3024, 2010.

[2] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997.

[3] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[4] F. Dellaert and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006.

[5] F. Dellaert, "Factor graphs and gtsam: A hands-on introduction," Georgia Institute of Technology, Tech. Rep., 2012.

[6] M. Kaess, A. Ranganathan, and F. Dellaert, "isam: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378, 2008.

[7] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g 2 o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3607–3613.

[8] M. Llofriu, P. Fong, V. Karapetyan, and M. Munich, "Mapping under changing trajectory estimates," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1403–1410.

[9] N. Banerjee, D. Lisin, J. Briggs, M. Llofriu, and M. E. Munich, "Lifelong mapping using adaptive local maps," in *2019 European Conference on Mobile Robots (ECMR)*. IEEE, 2019, pp. 1–8.

[10] A. Kleiner, R. Baravalle, A. Kolling, P. Pilotti, and M. Munich, "A solution to room-by-room coverage for autonomous cleaning robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 5346–5352.

[11] D. Goel, J. P. Case, D. Tamino, J.-S. Gutmann, M. E. Munich, M. Dooley, and P. Pirjanian, "Systematic floor coverage of unknown environments using rectangular regions and localization certainty," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 1–8.

[12] I. Vandermeulen, R. Groß, and A. Kolling, "Turn-minimizing multi-robot coverage," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 1014–1020.

[13] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[14] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, pp. 1–17, 2021.

[15] D. Schlegel, M. Colosi, and G. Grisetti, "Proslam: Graph slam from a programmer's perspective," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–9.

[16] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison, "Slam++: Simultaneous localisation and mapping at the level of objects," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 1352–1359.

[17] J. McCormac, R. Clark, M. Bloesch, A. Davison, and S. Leutenegger, "Fusion++: Volumetric object-level slam," in *2018 international conference on 3D vision (3DV)*. IEEE, 2018, pp. 32–41.

[18] M. Colosi, S. Haug, P. Biber, K. O. Arras, and G. Grisetti, "Better lost in transition than lost in space: Slam state machine," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 362–369.

[19] P. Jensfelt and S. Kristensen, "Active global localization for a mobile robot using multiple hypothesis tracking," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 5, pp. 748–760, 2001.

[20] S. Lee, S. Lee, and S. Baek, "Vision-based kidnap recovery with slam for home cleaning robots," *Journal of Intelligent & Robotic Systems*, vol. 67, no. 1, pp. 7–24, 2012.

[21] I. Motte and G. Campion, "A slow manifold approach for the control of mobile robots not satisfying the kinematic constraints," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 6, pp. 875–880, 2000.

[22] W.-S. Lin, L.-H. Chang, and P.-C. Yang, "Adaptive critic anti-slip control of wheeled autonomous robot," *IET control theory & applications*, vol. 1, no. 1, pp. 51–57, 2007.

[23] R. Balakrishna and A. Ghosal, "Modeling of slip for wheeled mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 1, pp. 126–132, 1995.

[24] S. Jung and T. C. Hsia, "Explicit lateral force control of an autonomous mobile robot with slip," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 388–393.

[25] D. Stonier, S.-H. Cho, S.-L. Choi, N. S. Kuppuswamy, and J.-H. Kim, "Nonlinear slip dynamics for an omniwheel mobile robot platform," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 2367–2372.

[26] G. Reina, L. Ojeda, A. Milella, and J. Borenstein, "Wheel slippage and sinkage detection for planetary rovers," *IEEE/Asme Transactions on Mechatronics*, vol. 11, no. 2, pp. 185–195, 2006.

[27] J. Palacin, I. Valganon, and R. Pernia, "The optical mouse for indoor mobile robot odometry measurement," *Sensors and Actuators A: Physical*, vol. 126, no. 1, pp. 141–147, 2006.

[28] D.-N. Ta, N. Banerjee, S. Eick, S. Lenser, and M. Munich, "Fast nonlinear approximation of pose graph node marginalization," in *Robotics and Automation (ICRA), 2018 IEEE International Conference on*, May 2018.

[29] N. Banerjee, R. C. Connolly, D. Lisin, J. Briggs, and M. E. Munich, "View management for lifelong visual maps," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 7871–7878.

[30] M. Narayana, A. Kolling, L. Nardelli, and P. Fong, "Lifelong update of semantic maps in dynamic environments," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020.