2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Madrid, Spain, October 1-5, 2018

Wireframe Mapping for Resource-Constrained Robots*

Adam Caccavale[1] and Mac Schwager[2]

***Abstract*— This paper presents a novel wireframe map structure for resource-constrained robots operating in a rectilinear 2D environment. The wireframe representation compactly represents geometry, in addition to transient situations such as occlusions and boundaries of unexplored regions. We formulate a particle filter to suit this sparse wireframe map structure. Functions for calculating the likelihood of scans, merging wireframes, and resampling are developed to accommodate this map representation. The wireframe structure with the particle filter allows for severe discrete map errors to be corrected, leading to accurate maps with small storage requirements. We show in a simulation study that the algorithm attains a map of an environment with 1% error, compared to an occupancy grid map obtained with GMapping which attained 23% error with the same storage requirements. A simulation mapping a large environment demonstrates the algorithms scalability.**

I. INTRODUCTION

In this paper we present a scalable, memory-efficient algorithm for a single robot to map an indoor 2D environment in a manner useful for navigation. All steps in the algorithm operate on data in its sparse form and no processing is done on a large set of discretized data. This strict adherence to sparsity enables mapping by resource-constrained agents such as microrobots (e.g. [1]), or for the mapping of large environments by a more typical robot. This approach allows a simple robot to handle large amounts of incremental data, constantly growing and evolving the map while dealing with uncertainty. If extended to multi-robot mapping (we are currently working toward this extension), the compact nature of the map, and the tools developed in this paper for merging maps, will enable the transfer of maps between robots.

To simplify the map structure, we assume that the environment can be represented by a set of line segments representing straight walls, which is a common approach taken for indoor mapping [2]–[4]. The main difference in our case is that the corners of these walls will be represented as nodes in a graph, and the walls themselves as edges. The overall structure of the map is a graph embedded in $\mathbb{R}^2$, or more simply, a *wireframe*. We build the wireframe from successive sensor scans in such a way as to encourage a small number of edges and nodes in a connected graph structure, resulting in fewer floating segments and a sparser map than in typical line segment map representations. To handle uncertainty over the space of possible wireframes, we

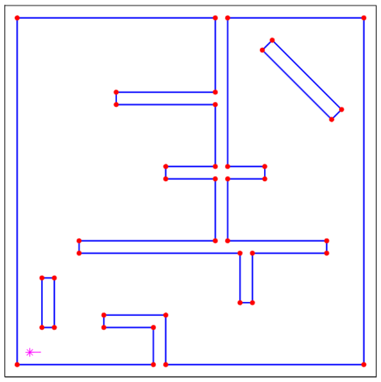

Fig. 1: We consider a mapping algorithm that represents the 2D environment as a wireframe, as shown here. We propose a particle filter architecture to represent uncertainty, and to correct for past errors in identifying walls (blue edges in the wireframe) and corners (red dots in the wireframe). The wireframe representation is highly storage efficient compared to feature-based or occupancy grid map representations.

propose a particle filter architecture in which each particle is a candidate wireframe with an associated likelihood weight. We define a measurement likelihood function for wireframes, and propose procedures for updating the wireframe particles with new sensor measurements, and resampling the particle set consistent with the traditional particle filter architecture. Our wireframe map also uses a novel categorization of nodes as *nominal* nodes, *occlusion* nodes, and *frontier* nodes. These three categories are essential to the measurement prediction and update steps in the particle filter, and also suggest natural integration with simple navigation strategies (e.g. [5]), such as collision avoidance (avoid occlusion nodes and graph edges) and exploration (chase frontier nodes). It is important to note the difference between our wireframe structure (which is mathematically a graph), and the typical pose graph used in simultaneous localization and mapping (SLAM) algorithms (see GraphSLAM [6]). A pose graph is made up of nodes (the robot's past poses) and edges (denoting relative measurements between poses), while our wireframe represents the environment geometry directly, with nodes as corners and edges as walls. It is also important to differentiate this work from that of Rao-Blackwelized particle filters (such as those used in FastSLAM [7]). In FastSLAM each particle represents a candidate robot trajectory. Conversely, in our wireframe particle filter, each particle is a candidate map geometry. We also stress that our method is not a feature based method, as it does not directly store sensor features in

*This research was supported in part by NSF grant CMMI-1562335 and ONR grant N00014-12-1-1000. We are grateful for this support.

[1]A. Caccavale is with the Department of Mechanical Engineering, Stanford University, Stanford, CA 94305 awc11@stanford.edu

[2]M. Schwager is with the Department of Aeronautics & Astronautics, Stanford University, Stanford, CA 94305 schwager@stanford.edu

the environment map. Similarly, our method is not based on occupancy grids, which are particularly storage inefficient, scaling quadratically in the length dimension of the environment. Instead, our method seeks to directly represent the metric geometry of the environment in a memory efficient way. The goal of this algorithm is to efficiently capture a map that is easy to store, communicate to other robots, and use for navigation. Given a bounded environment, the memory storage requirements and computational complexity for our method are constant in time. We show in simulation comparisons that, using the same storage capacity, our method builds a 22% more accurate map than the GMapping algorithm [9].

A map representation is not useful if it cannot handle uncertainty. For example, incorrectly adding an edge between two nodes in a wireframe can drastically change how usefully the map is for navigation. We therefore propose a particle filter architecture to handle the multiple hypotheses resulting from uncertainty in the binary decision regarding the existence of edges in the wireframe. Hence, one of the key contributions of our work is in adapting the particle filter architecture to accommodate a probabilistic distribution over wireframes. Since a wireframe is not a real-valued random variable, as is typically used in a particle filter, we propose all the necessary components of the particle filter specifically suited to deal with wireframe random variables. A key feature of this particle filter architecture is that incorrectly added edges which contradict future scans are eventually removed from the map, hence the method can correct for severe discrete map errors to approach the true environment map over time.

The remainder of this work is organized as follows. Related work is discussed in Sec. II and the problem is mathematically formalized in Sec. III. Section IV describes the main particle filter algorithm, and Sec. V discusses simulation results comparing performance with GMapping. Finally, our conclusions are given in Sec. VI.

II. Related Work

Our work is most closely related to SLAM, although, unlike SLAM, we do not represent the robot trajectory explicitly. The SLAM literature is too vast to provide a adequate survey here, however a comprehensive overview of SLAM variations are described in [8]. Our filter architecture is adapted from the well-known particle filter [6]. Particle filters have been used extensively for mapping, for example FastSLAM [7] which uses a feature-based approach, and GMapping [9] which uses an occupancy grid approach. We compare our simulation results to the method in [9].

Some existing work in SLAM focuses on light-weight mapping solutions. For example, the goal of extreme efficiency is sought by [4] and [3]. Both use a Rao-Blackwellized particle filter for handling uncertainty, with each particle representing a different robot trajectory hypothesis. Both papers use map representations that are sets of unconnected line segments, in contrast to our proposed wireframe in which the line segments are connected at nodes, and are constantly pruned to yield simple wireframe maps. Furthermore, the approach in [3] specifically focuses on horizontal and vertical line segments, while our wireframe can accommodate edges in any orientation. An EKF slam approach (no particle filter) is presented for line segments in [10] and [2]. Specifically, [2] combines line segment features with model-tracking to construct a map of the environment. An unscented Kalman filter is used to estimate line segments, which are added to the map if they meet certain criteria. This paper also uses the term "wireframe," but the term is used differently from our usage here. The structure in that work is not a series of connected line segments but instead a superposition of disconnected line segments. Finally, the prior work with the most-similar map representation is [11]. Line segments are filtered and connected at the ends to form polygonal obstacles. Here, shapes represented by a series of connected line segments are extracted from range data. These shapes are matched to the existing map to both localize the robot and align the scan. However, that work does not focus on correcting errors introduced into the map through a probabilistic filter as we do here.

In our work we assume that a sensor processing layer extracts environment corners from raw sensor measurements, hence our measurement scan consists of noisy environment corner locations relative to our robot. This sensor processing layer can be realize with several existing algorithms, for example the split and merge algorithm proposed by [12] extracting line segments from a 2D Lidar scan [13]. Alternatively, we could use the algorithm in [14] to produce a dense depth map from a Lidar scan, and then extract the corner locations from this dense depth map. Another method could be to use line or corner features to extract corner locations from a monocular camera, a stereo camera, or an RGB-D camera [15], [16].

III. Problem Formulation

We consider a robot tasked with building an environment map modeled as a wireframe. In this section we rigorously define a wireframe, and describe a distribution over wireframes represented as a weighted particle set.

A. Wireframe Map Representation

A key feature of the presented algorithm is the sparse map representation. A labeled directed embedded graph, referred to in this paper as a wireframe, is used as the map representation. Typically, a graph $G = \{V, E\}$ is a set of vertices V and a set of edges E. Our wireframe is "directed" because the edge connecting vertex i to j is not the same as edge connecting vertex j to i, "embedded" because it includes a function that maps each vertex to a point in physical space, and is "labeled" because it includes a function mapping each vertex to a descriptor label. Specifically, a wireframe is a four tuple $W = \{V_W, E_W, \phi, \lambda\}$, where $V_W = \{1, 2, \ldots, n\}$ is a set of n vertices. The edge set consists of pairs of connected vertices, $E_W = \{\ldots, (i, j), \ldots\}$, where $(i, j) \in E_W$ denotes an edge between vertices i and j and represents a wall in the map. Each index $i \in V_W$ is mapped to a point in space $p_i = \phi(i)$ by function $\phi : V_W \to \mathbb{R}^2$. Let $\mathcal{L}$ be the set of possible labels for the nodes. Then each index is mapped to a label $l_i = \lambda(i)$ by function $\lambda : V \to \mathcal{L}$. For this algorithm the set of possible labels is $\mathcal{L} = \{nominal, occlusion, frontier\}$, the purpose of which is explained below.

An example wireframe is shown below and illustrated in Fig. 3. For each vertex i the embedding p_i is shown and the label l_i is represented by a symbol. The meaning of these labels will now be explained.

$$\begin{aligned} W = \{ & V : \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}, \\ & E : \{(2, 1), (3, 2), (4, 3), (5, 4), (6, 5), (7, 6), (8, 7), \\ & \quad (9, 8), (10, 9), (11, 10), (12, 11), (1, 12), (13, 14), \\ & \quad (14, 15), (15, 16), (16, 13)\},\ \phi,\ \lambda\}. \end{aligned}$$

Each vertex is tagged with one of the labels in $\mathcal{L} = \{nominal, occlusion, frontier\}$, and examples of each are shown in Figure 2. The first label $nominal$ is applied when two edges can be seen converging at a point (i.e. at a corner). In a fully explored map every vertex will be labeled as $nominal$. The other two point types are used as the map is being constructed, and are a result of limits on the robot's sensing (occlusions, range, etc). The $occlusion$ label is used for vertices at the end of a visible edge that is partially obscuring another edge. The point formed by projecting the occlusion point onto the partially-occluded edge is labeled $frontier$. This special type of point does not exist in the "real" map, it is an artifact of the robot's current limited view. This label indicates that the point is not an endpoint, but rather, the edge on which it sits continues beyond what has been visible up until this point. If the robot were to move in the direction of the frontier point, the next scan would reveal more of the edge and the frontier point would appear to move along the line segment until the entire segment is in view, capped by nominal points at each end.

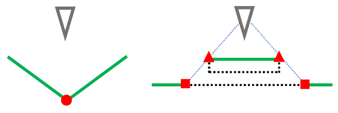

Fig. 2: Visible walls are drawn in green, the robot's field of view is blue, and walls that are occluded are black dashed lines. The nominal point is shown as a red circle (left) and occlusion points and frontier points are shown as triangles and squares respectively (right). Note that frontier points are not corners in the environment, but are a result of partial occlusions that occur along an edge.

Labeling the vertices is helpful for many of the algorithm's steps as described in Sec. IV-B. Specifically, these labels are crucial for predicting a future scan given a wireframe particle, for assessing the likelihood of a scan given a wireframe particle, and for fusing new scans into an existing wireframe particle. Furthermore, if $l_i = frontier$ in the robot's wireframe this indicates an unexplored area, which can be leveraged for directing the robot to efficiently explore an environment, which will be considered in future work.

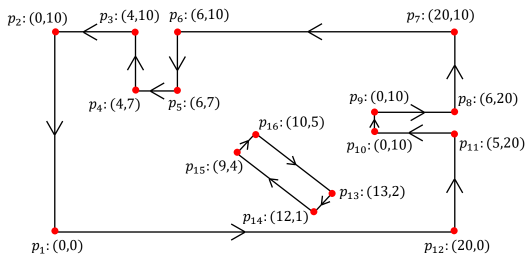

Fig. 3: Complete Wireframe Map - The walls are shown as directed black lines. The embedding in $\mathbb{R}^2$ of each point ($\phi(i) = p_i$) is shown at the point location. Since this is a complete map, the labels of all points are $nomial$ and denoted by red circles. For comparison, Fig. 4 shows an incomplete map with various point types.

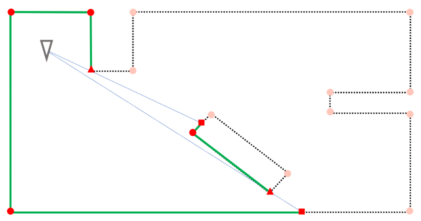

Fig. 4: Partially Completed Wireframe Map - Visible walls are shown as green lines and occluded walls as dashed black lines. Visible nominal points are red circles, occlusion points are triangles, and frontier points are squares. Points out of sight are shown as faded red circles. Dashed blue lines highlight the paired nature of occlusion and frontier points.

The direction of the edges provides a subtle but important advantage for comparing a predicted scan to an actual scan. Laser scan output is ordered, so in a single scan all corners will be detected in a consistent direction (e.g. clockwise). In the scenario where a robot makes a U-turn into unexplored area, ray tracing from the robot's new position would predict that the back of the previously seen wall would be detected. In reality, no walls are infinitely thin so the opposite is true: we would expect to see a wall between us and the previously detected wall. Directed graphs capture this information because the predicted corners will appear in the wrong order when seen from the back of that edge. See Fig. 5 for an illustration of this scenario. This is especially important for the calculation of the likelihood function (see Sec. IV-D), where the difference between the expected and actual graphs decrease the weight of the particle. Edge direction can also be used to inform the merging of graphs by disallowing edges of conflicting directions to be merged [3].

B. Handling Uncertainty - A Particle Filter for Wireframes

A fundamental property for any mapping algorithm is robustness to erroneous measurements. For a wireframe map, errors are manifested from both continuous and discrete sources. The location of the vertices is naturally represented

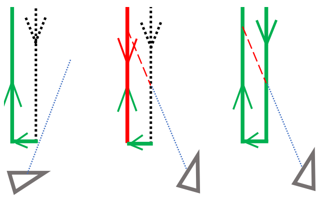
Fig. 5: The robot is moving into unexplored area around a wall (left). Without tracking wall direction, the prediction in red would overlap the previously seen edge (center). The conflicting directions identify this scenario, and the wall likelihood is not penalized. The rightmost image shows what the robot actually sees.

as a continuous random variable, while the presence or absence of connections between vertices (i.e. edges) is naturally represented as a discrete random variable. The mixed continuous/discrete nature of the wireframe suggests that a particle filter is an appropriate methodology for handling the diverse multiple hypotheses that are possible for a wireframe map. Hence, we build the appropriate tools for a particle filter tailored to estimate a wireframe random variable.

In our particle filter, the distribution over wireframes is approximated by a finite set of weighted wireframe particles, each with a weight proportional to its likelihood given the history of scans. The set of M weighted particles is $\mathcal{W} = (W_1, w_1),\ (W_2, w_2),\ \ldots,\ (W_M, w_M)$. We sometimes show the weights with time superscripts, but these will be left off unless needed for clarity. Let S be the sensor output (which is also a wireframe). The measurement likelihood function is a measure of how well a given scan is explained by a particle, denoted i is $p(S \mid W_i)$. This value is used to update the particle's weight at each iteration $w_i^{new} = \eta w^{old} p(S \mid W_i)$, where η is a normalizing factor. An example of a set of wireframe particles is shown in Fig. 6.

The steps for the particle filter follow those shown in Fig. 7. First, a scan S is obtained. To calculate a scan's likelihood relative to particle i, a predicted scan is generated from the robot's current position and particle W_i. The likelihood is calculated and the weight is updated to be $w_i^{new} = \eta w_i^{old} p(S \mid W_i)$. The scan and particle are then merged, and this process is repeated for each particle. Finally, a new set of particles are selected through the resampling process described in Sec. IV-F.

IV. Algorithm

The following section describes the major steps in the wireframe mapping process. The flow of information and sequence of actions are shown in Fig. 7. All pseudocode can be found at `https://msl.stanford.edu/wireframe-mapping-resource-constrained-robots`.

A. Move, Scan, and Odometry

At each time step, t, the robot first moves according to an external motion command (e.g. from a human operator), then it takes a sensor measurement. The algorithm is agnostic to the type of sensor. We assume that an existing sensor preprocessing step extracts the relative positions of the corners from the visible scene, with additive Gaussian noise. These corner positions form the vertex set of a wireframe that we call a *scan*. These vertices are connected counterclockwise to form the edge set, and labels for each vertex are assigned to complete the wireframe scan, $S^t = \{V^t_{scan}, E^t_{scan}, \phi^t_{scan}, \lambda^t_{scan}\}$. The labels are assigned by checking for two vertices that are colinear (within a tolerance) along the same line of sight from the robot. This indicates that the closer of the two is an $occlusion$, and the farther of the two a $frontier$. If there is only one vertex along a line of sight, it is $nomial$. Since the scan is taken in the robot's coordinate frame, it is rotated and translated according to the robot's current estimated position derived from an onboard odometry measurement. The inevitable error from the odometry estimate is later accounted for in the algorithm in Sec. IV-C.

B. Predict Wireframe

In order to calculate the likelihood of a given scan, the robot first needs to be able to determine what it expects to measure. Hence a predicted wireframe scan is computed for each wireframe particle. First, all known vertices within the robot's sensing range are sorted counterclockwise and points visible from the robot's position are identified. Using this information, the visible sections of edges are found. An edge that is partially occluded will terminate in a $frontier$ vertex, and its occluding vertex in the foreground is labeled appropriately as $occlusion$. Every edge has two vertices, so only the portion of the edge that is visible is added to the predicted scan. A single edge may be broken into multiple edges with $frontier$ vertices if, e.g., it is obscured by a shorter edge in front of it.

The complexity of evaluating if a single point is visible to the robot is at best $\mathcal{O}(n)$ where n is the number of vertices (or edges, $|E| <= 2|V|$ because $E \subset V \times V$). This check will need to be performed for each vertex so the naive implementation will be $\mathcal{O}(n^2)$. This can be sped up using a binary search tree to achieve $\mathcal{O}(n \log(n))$ [5].

C. Wireframe Matching and Correspondences

The measured scan and predicted scan will not align perfectly due to noise in the scan, as well as position errors from odometry. A modified version of the iterative closest point (ICP) method [17] is used to generate the rotation and translation which will best align the wireframes. Not only can this be used to align the scan and current wireframe, but it can also be used to update the robot's position estimate. An additional benefit of ICP is that in calculating the shift, the correspondences between points are established by a closest distance criteria where no single vertex can correspond to more than one other vertex. Similarly, correspondences between edges are determined. If both end points correspond, then the edge corresponds. If not, the minimum offset between the segments and their relative angle determine if the edges correspond. Unlike vertices, edges can have duplicate correspondences. However, the matching algorithm

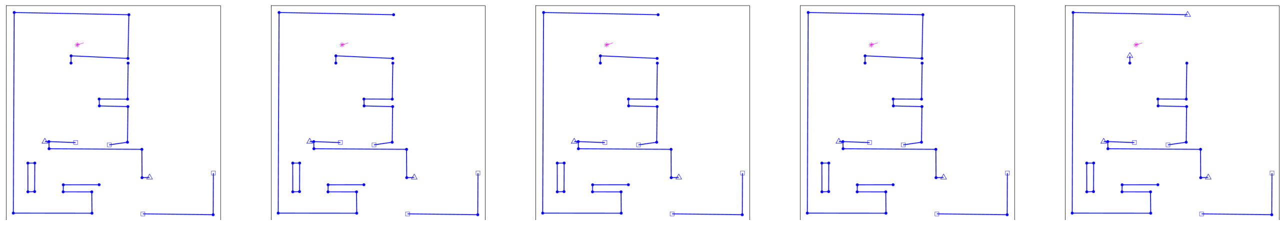

Fig. 6: This figure shows an example of 5 wireframe particles. Most of the walls are in similar positions, but there are noticeable absences due to the resampling process removing edges with low-likelihood weights.

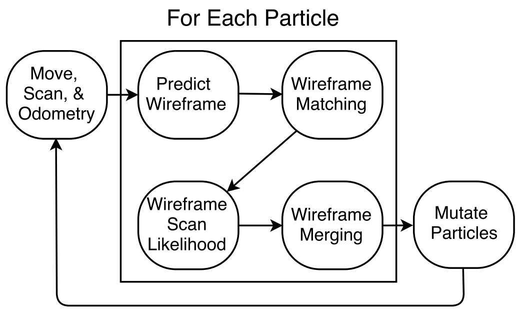

Fig. 7: High-Level Algorithm Flow

is still susceptible to large errors due to noise, which are then corrected through the particle filter. Fig. 8 shows the outcome of a incorrect large shift due to a failure in the matching algorithm.

D. Wireframe Scan Likelihood

The likelihood function takes in a wireframe particle W_i and a scan S and returns the likelihood $p(S \mid W_i)$, which is used to update the particle's weight. We propose to use a likelihood function which treats each vertex measurement as an independent Gaussian distributed random variable, and the presence of each edge as an independent Bernoulli random variable. For the vertices, we denote $g_i(\cdot) = \mathcal{N}(p_i, \Sigma)$ as the measurement likelihood for a vertex i, with mean p_i (the vertex position in the wireframe particle), and known covariance matrix Σ. For the edges, we denote $b(e_{scan}, e_{particle})$ as the Bernoulli distribution for the existence of an edge. Specifically, $b(1,0) = p_{fp}$ is the likelihood of seeing an edge in the scan that is not in the particle (a false positive), hence $b(0,0) = 1 - p_{fp}$ is the true negative likelihood. Similarly, $b(0,1) = p_{fn}$ is the false negative likelihood and $b(1,1) = 1 - p_{fn}$ the true positive likelihood. We assume the Bernoulli parameters p_{fp} and p_{fn} are known. Therefore, the overall likelihood of a scan is given by $p(S \mid W_i) = \prod_i g_i(p_{i,scan}) \prod_{\{i,j\}} b(e_{scan}(i,j), e_{predict}(i,j))$, where $e_{scan}(i,j) = 1$ if $\{i,j\} \in E_{scan}$ and 0 otherwise, and $e_{predict}(i,j) = 1$ if $\{i,j\} \in E_{predict}$ and 0 otherwise. Furthermore, a likelihood for an individual edge can be captured by taking the product of the two vertices' Gaussian likelihoods multiplied by the edge's Bernoulli likelihood. These individual edge likelihoods are leveraged to prune the edges most likely to be a result of noise from the wireframe as detailed in section IV-F.

Unfortunately, often times there is not a complete correspondence between points in a scan S and a predicted scan P. In order to calculate the wireframe likelihood in this case, temporary points are added to S whenever there is a point in P that does not have a match. If the point is located between a frontier and occlusion point (with no edge behind), then this is considered to be unexplored space. In order to not penalize the map likelihood when seeing genuinely new edges, the temporary point is placed directly onto the new vertex. However, if not in a gap, the temporary point is projected onto the closest edge in S. This is done for both wireframes, resulting both having the same number of points with correspondences fully established. Consider, for example, the situation where there is a full edge in P, but only half is visible in S due to an occlusion. This means there will be a frontier point halfway along the length of the edge in S, that will not have a correspondence to a point in P. The temporary point added will be projected onto the full edge in P, and this has the effect of only penalizing the frontier point for is distance from the edge, which is desired.

E. Wireframe Merging

After scanning the environment, the algorithm must decide how to incorporate the new information into the current wireframe. The correspondences between vertices previously calculated make it easy to determine which points should be merged, if the two are within a certain threshold then the points are combined.

If a new vertex falls along an edge, the locations of the other vertices it is connected to is important to determining the outcome of the merge. If there is no neighbor, then the vertex is simply removed and the two adjacent colinear edges are joined. If the neighbor is along the same edge, this shorer edge is absorbed into the other. If the neighbor is colinear but extends past the endpoints (i.e. a portion of the line segments overlap), then one edge is formed spanning the two most distant endpoints.

It is important to note that this is a union, and therefore if there is an edge present in the scan and not in the current wireframe (or vice versa), the resulting merged wireframe will always contain this new edge. Without the process described in section IV-F, any erroneously sensed edges would be permanent additions to the map.

Throughout the merging process we track which of the original edges are merged into each final edge in order to propagate the individual edge likelihoods. If two edges with likelihoods $L1$ and $L2$ are merged to form a third edge, then the merged likelihood is taken to be $L_{merged} = 1 - (1 - L1)(1 - L2)$. This equation comes from the insight that if two walls appear in the scan, and are believed to be parts of the same wall, then their probabilistic contribution to the new likelihood is represented by one minus the probability that neither of those two edges actually exists.

F. Particle Mutation (Resampling)

In a typical particle filter [6], the particles are randomly sampled according to their weights. The chosen particles are then propagated forward via noisy dynamics model, resulting in a diverse set of hypotheses. However, in this algorithm the combination of correcting for odometry errors through scan matching, the static environment, and the way the merging process combines vertices means that propagating the particles forward via dynamics would not cause the particles to naturally diversify. We propose alternative method to overcome this issue. This method is based on the individual edge likelihood values calculated in section IV-D.

The resampling process begins by sampling with replacement from the integers between 1 and c (the number of particles). The number j ($\in \{1, ..., c\}$) has a probability of w_j of being selected each draw. Each time j is drawn, the jth wireframe will be propagated forward. So far this is consistent with the traditional particle filter resampling algorithm. However, instead of the particle's state being propagated forward by a dynamics model, the wireframes are modified according to the following process.

First, edge candidates for deletion are culled based on if they are within sight (or predicted sight) of the robots location. This is to prevent edges from being deleted across the map, when they might have been correct. Next any remaining edge that is above a certain threshold is maintained. This is to ensure that if an edge doesn't truly exist, this fact is verified though multiple scans. The number of particles to be formed by mutating this particle is determined by the number of times is had been randomly sampled. If only sampled once, the state of the particle will remain unchanged. For ever sample after that, each candidate edges will be randomly kept or eliminated based on its individual likelihood.

G. Wireframe Memory Efficiency

This mapping algorithm is built around using the minimalist wireframe data structure. This structure contains $|V|$ index variables, $2|E|$ edge variables, $2|V|$ xy pairs in ϕ, and $|V|$ labels in λ) for a total of $4|V|+2|E|$ variables. If there are M particles the total number of values will be $M(4|V|+2|E|)$. When implemented computationally, this can be reduced by $|V|$ by storing the vertices implicitly by ordering the xy pairs of ϕ in order, e.g. $\{(x_1, y_1), (x_2, y_2), ..., (x_{|V|}, y_{|V|})\}$. To highlight just how sparse the wireframe structure is, consider a map that spans the $15 \times 15m$ small simulation environment from Sec. V. This map contains 42 points ($|V|$) and has 42 edges $|E|$, for a total of 210 variables per particle. This can now be compared to occupancy grids, a common map representation [6].

With a fine enough resolution, any occupancy grid can represent any shape. This representation is easily used for navigation as a check to see if a location is obstacle free is a simple query. However, this representation has a serious drawback when concerning scalability. As the size of the environment grows, the map size (for two dimensions) grows quadratically, even if the map has no obstacles. Assuming a course resolution of 0.2m, the occupancy grid would have to store 1125 values. A resolution of 0.1 would require 2250 values. This problem is only exasperated when going to three dimensions. Improvements on the scalability of occupancy grid based maps have been made, for example the octree structure [18], but these methods compromise the ease at which this map type can be used for navigation. To create an equitable scenario, a resolution was chosen to make the total memory of the maps approximately equivalent so the quality of the output can be directly judged.

GMapping's occupancy grid values are each eight bits, so with a grid resolution of $0.2m$ the occupancy grid requires $1125 * 8 = 9000\ bytes$. The wireframe memory requirement is more complicated. While the wireframe map is not discretized, there is a threshold for deciding if two corners should be merged so this was set equal to the grid resolution of $0.2m$. With this resolution, the pairs of output values from $\phi(i)$ can be stored as two shorts (4 bytes). Each label is one of three options, and as such only requires 2 bytes. The indices are implicitly stored as the order of the outputs of $\phi(i)$, so no additional storage is required. And finally, 2 values are stored for each edge, where each value is an integer which is at most the number of vertices (in this case 42). Each of these can then be stored as 5 bytes. The net result of this is $|V| = |E| = 42$, so the total storage per wireframe is $2 * 4 * 42 + 2 * 42 + 2 * 5 * 42 = 840\ bytes$. With ten particles the total is almost equal, at 8400. For the larger environment a merge threshold of $0.4m$ was used so the occupancy grid would requires $200/0.4 * 200/0.4 * 8 = 2 * 10^6\ bytes$ The resulting wireframe map has 293 vertices and 298 edges and uses 6 particles, resulting in $6*((2*4+2)*293+2*5*298) = 35460\ bytes$, a reduction by a factor of 56.

V. Simulation and Results

Simulations were run to highlight the efficiency and effectiveness of the algorithm in two different environments. As the robot moves through the environment it is constantly incorporating noisy scans into its current wireframe. As the robot moves, odometry noise causes the robot to believe it is translated and rotated from its true position. This has a major effect when a new scan is taken in the robot's true coordinate frame, but shifted to align with the map based on the robot's incorrect belief about its own coordinate frame. 10 particles were maintained for the first simulation run, though this is likely more than needed. For the second (larger) simulation, only 6 particles were used.

To overcome the error due to odometry, the rotation and translation output from ICP is used to update the robot's believed position. However, noise in the corner measurements

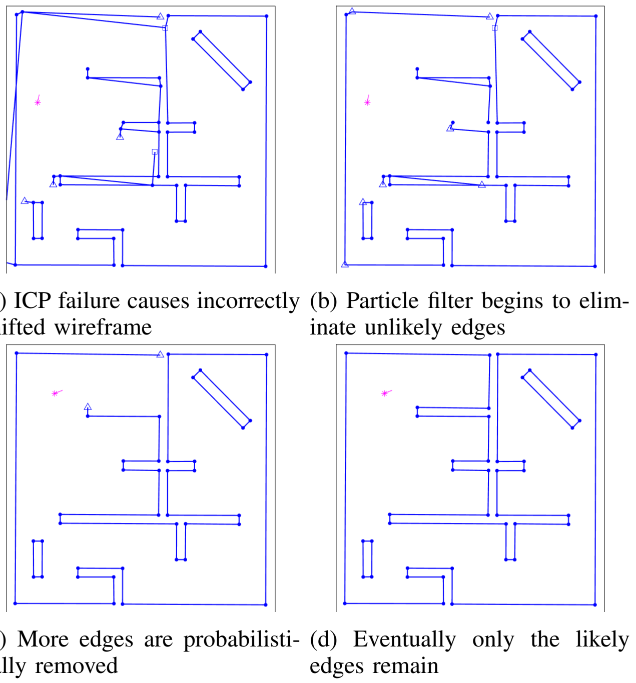
(a) ICP failure causes incorrectly shifted wireframe (b) Particle filter begins to eliminate unlikely edges (c) More edges are probabilistically removed (d) Eventually only the likely edges remain

Fig. 8: This series of figures show various particles after a sudden influx of incorrect edges caused by a failure in scan matching. The particle filter is able to robustly recover in only a few steps.

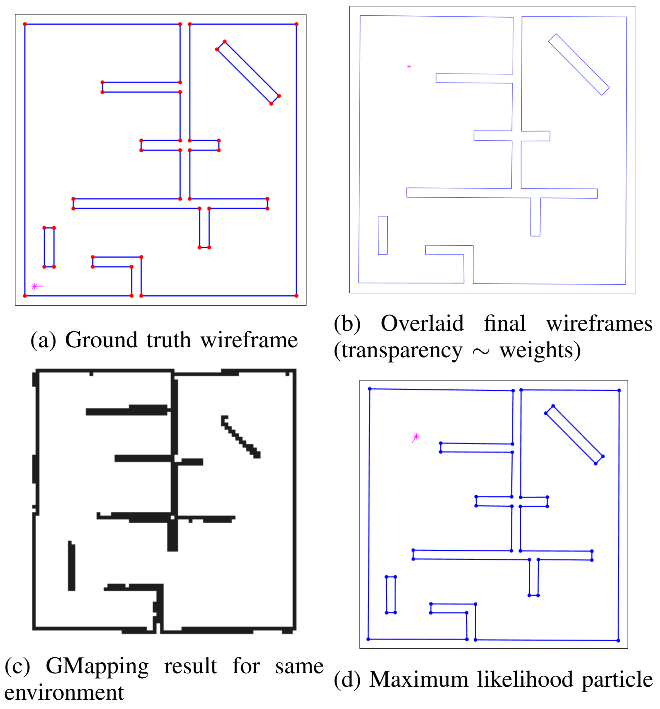
(a) Ground truth wireframe (b) Overlaid final wireframes (transparency $\sim$ weights) (c) GMapping result for same environment (d) Maximum likelihood particle

Fig. 9: These figures show the wireframe map and Gpmappings occupancy grid. Note the artifacts and inefficiencies in storage inherent to GMapping's methodology

can cause this process to fail, and the resultant scan and map are misaligned. See Fig. 8 for an example of this occurring, and the issue being fixed through the robustness of the particle filter. Fig. 9 shows the final output of the algorithm compared to Gmapping (a common mapping platform based on an occupancy grid-based Rao-Blackwelized particle filter [9]), and Fig. 10 shows the algorithm performed in a larger environment.

A. *In-depth Comparison with Existing Methods*

In order to create an objective metric for comparing the wireframe algorithm's map and GMapping's occupancy grid map fairly against ground truth the occupancy grid's resolution was set equal to the threshold below which two points are merged in the wireframe maps. This value is the effective "resolution" for the wireframe. This resolution was then used to transform the wireframe maps into an occupancy grid-type map where each cell's value depends on whether its center location falls inside an obstacle.

The evaluation metric for comparing the map outputs is a cell-by-cell comparison against the ground truth. Since the ground truth and the wireframe's algorithm outputs are both wireframe structures, they are first discretized using the same resolution as the occupancy grid. The score for a given graph is $f_{score}(map) = \sum_{i=1}^{N} \mathcal{I}(map_{truth}(i) == map(i))$. This score is effectively counting the number of cells considered to be erroneous. Overall the GMapping map had 11,955 errors (23.2%) and the wireframe algorithm's map had only 779 errors (1.51%). Furthermore, the wireframe map is qualitatively a much better representative of the ground truth.

Another common map representation is a collection of features. An example of a technique that uses this approach is FastSLAM [7], where a particle filter is also used to handle uncertainty. Here, unlike in the method presented by the authors, each particle is a robot path and a set of means and covariances for each feature. If the features are line segments, there would be 4 variables needed to represent the mean of each end point and 3 unique variables for the covariance matrix resulting in a total of 7 variable per feature. The robot pose (3 variables) is tracked at each timestep so for M particles each with Q features over T timesteps there are $M(3T+7Q)$ variables required (note that this will grow linearly with time). Assuming the number of features is equal to the number of edges in the calculation above, and that there is the same number of particles, after 10 time steps there would already be 324 values to store. While this is similar to the wireframe, this will continue to grow each iteration of the algorithm. Furthermore this calculation is making the best-case assumption that there are no two features along the same wall, which is unlikely. In the wireframe model line segments are continuously merged, preventing this issue.

There are other drawbacks associated with feature-based SLAM methods as well. While FastSLAM can lead to good localization, the map it constructs is certainly less sparse but also less easily used for navigation. If the goal, as in this paper, is to allow a simple robot to quickly map out an

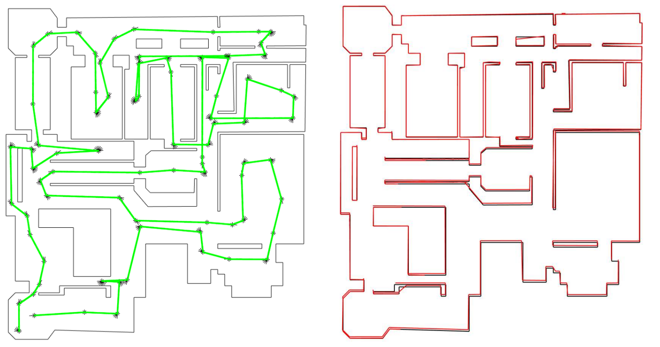

Fig. 10: The left image shows the large ground truth environment (approximately $200m \times 200m$) with the robot's trajectory in green. The right image shows the algorithm output in red overlaying ground truth in black

environment, it is not desirable for the robot to be performing complicated path planning along the way. The algorithm described in [5] calculates the list of visible vertices, and this information alone is enough for basic navigation and obstacle avoidance. Calculating the visible vertices is already part of the algorithm presented in this paper, so this method can immediately be adopted. To extend this method to guide the robot so that it explores the whole environment, the frontier points can be used as a moving waypoint since they indicate areas that have yet to be explored.

B. Large Simulation

A larger simulation was run using a room layout based on the floor plan from the Museum of Natural History in New York City (see Fig. 10). This environment is roughly $200 \times 200m$, and contains more intricate room layouts than the smaller simulation environment. Furthermore this map contains large empty rooms and long featureless corridors, situations that might pose a challenge to the algorithm. Unlike with the smaller simulation that served to highlight the advantage of this method versus common map representations, a direct comparison was not made with Gmapping as the occupancy grid requires vastly more storage than a the wireframe representation of comparable resolution (see Sec. IV-G). This simulation is instead intended to qualitatively demonstrate the algorithm's performance in a much larger environment, and indeed the output and ground truth are very similar. To quantitatively compare the wireframe output with the ground truth, the output wireframe was finely discretized into points and at each the minimum distance to the ground truth wireframe was found. The average of these distances is $0.33m$.

VI. Conclusion

In this paper we propose a map representation, the wireframe, and a unique particle filtering algorithm where the weighted wireframe maps are the particles. This methodology enables the robot to very efficiently map out rectilinear environments. The particles themselves are the maps, and this unique setup enables the robot to overcome sensor noise, odometry error, and abnormal disturbances with minimal computational resources. This is demonstrated through simulations with the output of our method comparing favorably, both in storage requirements and quality to the output of GMapping.

Ongoing work extending this methodology to multi-robot mapping is currently underway. The extreme compression of data through the map representation makes it ideal for sharing maps between robots. Furthermore we intend to investigate expanding this method to three dimensional environments where the computational savings and ease of low-computation navigation are even more critical.

References

[1] A. T. Baisch, O. Ozcan, B. Goldberg, D. Ithier, and R. J. Wood, "High speed locomotion for a quadrupedal microrobot," *The International Journal of Robotics Research*, vol. 33, no. 8, pp. 1063–1082, 2014.

[2] A. P. Gee and W. Mayol-Cuevas, "Real-time model-based slam using line segments," in *International Symposium on Visual Computing*. Springer, 2006, pp. 354–363.

[3] B.-W. Kuo, H.-H. Chang, Y.-C. Chen, and S.-Y. Huang, "A light-and-fast slam algorithm for robots in indoor environments using line segment map," *Journal of Robotics*, vol. 2011, 2011.

[4] S. Y. An, J. G. Kang, L. K. Lee, and S. Y. Oh, "Slam with salient line feature extraction in indoor environments," in *2010 11th International Conference on Control Automation Robotics Vision*, Dec 2010, pp. 410–416.

[5] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Santa Clara, CA, USA: Springer-Verlag TELOS, 2008.

[6] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[7] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *In Proceedings of the AAAI National Conference on Artificial Intelligence*. AAAI, 2002, pp. 593–598.

[8] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, Dec 2016.

[9] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, Feb 2007.

[10] A. Garulli, A. Giannitrapani, A. Rossi, and A. Vicino, "Mobile robot slam for line-based environment representation," in *Proceedings of the 44th IEEE Conference on Decision and Control*, Dec 2005, pp. 2041–2046.

[11] D. Wolter, *Spatial Representations for Mapping*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 19–52.

[12] T. Pavlidis and S. L. Horowitz, "Segmentation of plane curves," *IEEE Transactions on Computers*, vol. C-23, no. 8, pp. 860–870, Aug 1974.

[13] V. Nguyen, S. Gächter, A. Martinelli, N. Tomatis, and R. Siegwart, "A comparison of line extraction algorithms using 2d range data for indoor mobile robotics," *Autonomous Robots*, vol. 23, no. 2, pp. 97–111, Aug 2007.

[14] F. Ma, L. Carlone, U. Ayaz, and S. Karaman, "Sparse sensing for resource-constrained depth reconstruction," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 96–103.

[15] C. Harris and M. Stephens, "A combined corner and edge detector," in *In Proc. of Fourth Alvey Vision Conference*, 1988, pp. 147–151.

[16] R. Nevatia and K. R. Babu, "Linear feature extraction and description," *Computer Graphics and Image Processing*, vol. 13, no. 3, pp. 257 – 269, 1980.

[17] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, Feb 1992.

[18] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.