2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Macau, China, November 4-8, 2019

Trust But Verify: A Distributed Algorithm for Multi-Robot Wireframe Exploration and Mapping

Adam Caccavale and Mac Schwager

Abstract— **This paper presents a novel distributed mapping algorithm for multiple resource-constrained robots operating in a rectilinear 2D environment. The algorithm is built upon the sparse wireframe map representation and updating framework in [1]. We propose an exploration strategy based on the labeling of the vertices in the wireframe map, combined with a map-merging interrupt routine that is activated when robots enter into communication range with one another. The maps are not naively merged, but instead the receiving robot verifies the received information before it is assimilated by attempting to drive to the location where the other robot was when communication was established. The robots do not share a global coordinate frame, so prior to a merge the relative map alignment is determined. This is achieved using the random sample consensus (RANSAC) framework with a custom feature which leverages the structure inherent in the wireframe map representation. This results in a lower rate of false-positive matches compared to another state-of-the-art feature used in point cloud alignment, the 4-point congruent set (4PCS). We show our feature to be more robust to false-positive alignments, a common occurrence when attempting to align sparse structures such as wireframe maps. We present high fidelity simulation results in a ROS-Gazebo environment with lidar-equipped TurtleBots[1] to highlight the benefits of our algorithm.**

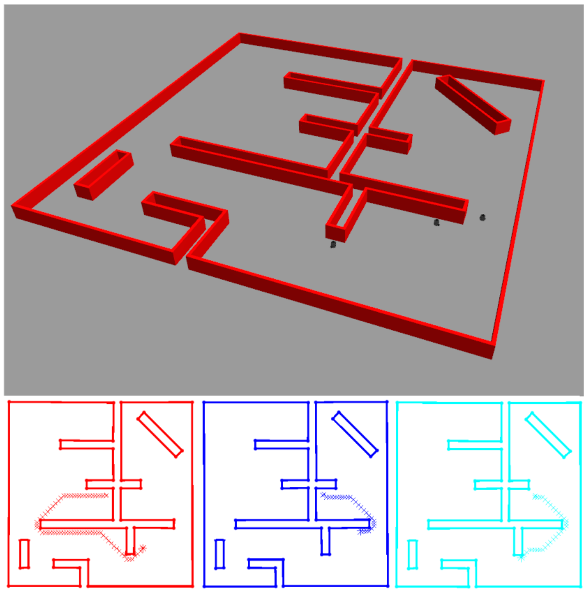
Fig. 1: Above: three TurtleBots collaboratively mapping a 15m$\times$15m maze environment in the Gazebo simulator using our decentralized wireframe mapping strategy. Below: the three robots' maximum likelihood particles (i.e. wireframes).

I. Introduction

In this work we present a decentralized control algorithm for resource-constrained robots to map an indoor 2D environment. Our algorithm builds upon the wireframe mapping framework introduced in [1]. This prior work establishes a scalable, memory-efficient method for a single robot to handle large amounts of incremental data to construct a map structured in a manner useful for navigation. This paper builds upon the wireframe representation by proposing a scalable cooperative exploration strategy that is fully distributed, as well as a distributed map merging algorithm that allows robots to benefit from the maps of other robots encountered serendipitously in the course of exploration. We simulate the robot dynamics and sensor readings in ROS-Gazebo to illustrate the performance of this algorithm in realistic mapping scenarios.

All steps in the algorithm operate on map data in its sparse wireframe form, and no processing is done on large data sets, e.g., from raw lidar scans, occupancy grids, or dense point clouds. Maintaining this strict adherence to sparsity enables mapping by resource-constrained agents such as microrobots (e.g. [2]), or for the mapping of large environments by more typical robots. Similarly, the sparsity of the wireframe map representation leads to lower bandwidth communication requirements among the robots compared to other cooperative mapping algorithms that use occupancy grids or point cloud map representations.

The wireframe map structure is an embedded labeled directed graph, with vertices representing corners in the map and edges indicating walls. The vertices are labeled based on their occlusion status as the map is constructed, and the edges are directed based on order of end-point detection (which is determined by the rotation of the laser). We use these properties both for aligning shared maps communicated between robots, and for directing the robots' exploration of the environment.

One key advantage to this sparse map representation is that, even with tight communication limitations, robots can transmit what information they have discovered to other robots. This ability to communicate allows the group of robots to more efficiently map an environment than a single robot following the same exploration algorithm. In the multi-robot case, the cooperation is implicit—no explicit joint

This research was supported in part by NSF grant CMMI-1562335 and National Robotics Initiative grant 1830402. We are grateful for this support.

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

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

[1]The TurtleBot3 burger configuration www.turtlebot.com

planning is performed, nor are assumptions made about the other robot's behavior. Upon receiving shared map data from another robot, a robot will not immediately incorporate the map, but instead first "verify" it by attempting to drive to the location where the other robot was when the communication was established. The other robot does not stay fixed for this verification, but continues with its own exploration algorithm, allowing for asynchronous behavior.

No global coordinate frame is assumed, so upon receiving map information from another robot, a robot first aligns the two maps. Since the wireframe structure is a graph in the mathematical sense, the problem could be treated as an instance of the subgraph isomorphism problem, which is known to be NP-complete [3]. Therefore a heuristic must be used. Treating the embedded vertices as a set of points allows for any point cloud matching technique (e.g. Iterative Closed Point) to be applied. However, most of these methods seek a local minimum and fail if the initial alignment is not close. Another common requirement of existing point cloud alignment algorithms is that a local normal vector can be found, or that there is a high density of points. Neither of these conditions are met when matching wireframes. Furthermore, when considering only point clouds, the existence or absence of walls between the points is ambiguous, which can lead to false-positive alignments. This paper introduces a new feature for wireframe map alignment that leverages the directed edges of the graph structure to achieve matches that are rotationally invariant, and are more robust under sparsity and symmetry than existing point cloud matching methods.

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. Sec. IV describes how the robots interact as they explore the environment, and Sec. V discusses the simulation results. Finally, our conclusions are given in Sec. VI.

II. Related Work

We use the wireframe representation for mapping, rather than full SLAM, that is, we do not explicitly estimate the robot trajectories, nor do we detect loop closure. However one could formulate a full SLAM algorithm using the wireframe map representation. The SLAM literature is too vast to provide an adequate survey here, however a comprehensive overview of SLAM methods are described in [4]. Section II of [1] details other sparse map representations and how they differ from the wireframe.

Within the field of SLAM, there are many notable multi-robot mapping papers including [5]–[11]. These approaches all vary in their advantages and disadvantages, though they all use established map representations and therefore do not have the same sparsity and navigability benefits that result from the wireframe framework.

The filter architecture used in both the wireframe map update and the verification procedure for proposed map alignments between robots is adapted from the well-known particle filter [12]–[14]. Particle filters have been used extensively for mapping, for example FastSLAM [15] which uses a feature-based approach, and GMapping [16] which uses an occupancy grid approach. The wireframe map is directly compared in simulation to [16], and is shown to give a higher quality map given equivalent memory capacity.

The wireframe representation assumes 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 is implemented by post-processing the output of split and merge, an algorithm proposed by [17] which extracts line segments from 2D lidar scans. Alternatively, we could use the algorithm in [18] 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 [19], [20].

There is also existing work related to the high-level control algorithm proposed in this paper. The most similar work is frontier-based exploration with multiple robots, which directs robots to the boundary between explored and unexplored space [21]–[23]. A small, but important, difference between what we do and this established idea is that our "frontier points" are at the end of line segments in our partially-constructed map—points where we know the true map continues beyond what we have perceived. This allows exploration without maintaining an explicit record of explored vs. unexplored space.

To merge maps, a feature is generated for each vertex based on the lengths and angles between relative parent, grandparent, child, and grandchild nodes (the order of which is determined by the scan direction). We argue that this feature information is more discriminating than simply using the embedded locations of the vertices such as [24]. The authors of [25] also use a feature based on lengths and angles, but it requires calculating the relative distance and angle between every vertex in the graph which is not scalable. Map alignments are then found by matching points with similar features and determining the best correspondence (and hence transform) through the RANSAC algorithm [26]. RANSAC is used in many algorithms and applications.

III. Problem Formulation

We consider an arbitrary number of robots tasked with building an environment map modeled as a wireframe. In [1] we rigorously define the wireframe representation and describe a framework for handling uncertainty in the map update using a distribution over wireframes represented as a weighted particle set. Noisy sensor data is collected as each robot moves throughout the map and particle filters update the map estimates over time. For this work it is assumed there is no separate odometry error, only noise in the sensing. To reduce drift due to odometry error, the technique described in the prior work of using the shift between each new scan and the existing map to update the robot's estimated position can be applied. The robots do not share a global coordinate frame, and the alignment of maps is achieved in a distributed manner. Each robot has a limited communication range

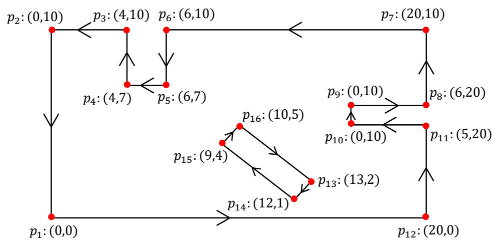

Fig. 2: Complete Wireframe Map - The walls are shown as directed black lines. The embedding in $\mathbb{R}^2$ of each point $(\phi(i) = p_i)$ is shown at the point location. Since this is a complete map, the labels of all points are *nominal* and denoted by red circles. For comparison, Fig. 3 shows a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

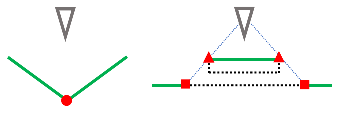

Fig. 3: Visible walls are drawn in green, the robot's field of view is blue, and walls that are occluded are black dashed lines. The nominal point is shown as a red circle (left) and occlusion points and frontier points are shown as triangles and squares respectively (right). Note that frontier points are not corners in the environment, but are a result of partial occlusions that occur along an edge.

through which the robot's unique ID, maximum likelihood particle (i.e. wireframe map), and current position (in its own frame) are transmitted.

A. Wireframe Map

As detailed in [1], a wireframe is a labeled embedded graph. Specifically, it is a 4-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. Each index $i \in V_W$ is mapped to a point in space $p_i = \phi(i)$ where function $\phi : V_W \rightarrow \mathbb{R}^2$. See Fig. 2 for an example.

Let $\mathcal{L}$ be the set of possible labels for the nodes. Each index is mapped to a label $l_i = \lambda(i)$ where function $\lambda : V \rightarrow \mathcal{L}$. The set of possible labels is $\mathcal{L} = \{nominal, occlusion, frontier\}$. Note that partially occluded edges (or edges truncated by a limited field of view) end in a vertex labeled as $frontier$ (see Fig. 3). These vertices will not appear in the final map but are artifacts of the limited view. In the previous work these labels assisted with building the map, but now are also leveraged to direct the exploration strategy as described further in Sec. IV.

B. Wireframe Particle Filter

The errors that occur in the wireframe representation are due to both continuous and discrete sources. The corner positions may be shifted by noise from where their ground truth locations are (continuous), while edges may be mistakenly introduced or left out (discrete). In [1], a particle filter framework is used to handle both types of errors. Unlike a traditional particle filter, the wireframe map itself is treated as a particle and the weighted set of particles approximates the belief over the space of all possible wireframes. The traditional steps for calculating the likelihood of scans, updating the weights, and resampling were customized to accommodate this unique type of particle. This process of maintaining a hypothesis and updating its weight as new measurements are taken will play a key role in the verification of proposed map alignments.

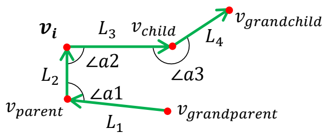

Fig. 4: The feature vector for point v_i is defined by 4 lengths and 3 angles calculated from it's 2-hop neighbors. For this example $f_i = [L_1, L_2, L_3, L_4, a_1, a_2, a_3]$.

C. Wireframe Feature

The core idea behind the wireframe representation is to minimize the stored detail while maximizing the utility (e.g. for navigation). The fact that points are indistinguishable from each other makes the correspondence problem challenging, especially without an estimate of the initial map alignment. In order to align shared map information between robots, a rotation-invariant feature is needed. Furthermore since many indoor environments are heavily symmetrical the feature must be robust to false positive matches.

To create a feature with these properties, the direction of the edges are leveraged to differentiate between otherwise identical vertices. For each vertex v_i a feature vector of length 7 is constructed. The wireframe structure is a directed graph so the ordered list of nodes within two hops can be found: $(v_{grandparent}, v_{parent}, v_i, v_{child}, v_{grandchild})$. The first four elements in the feature vector are the distances between each pair of consecutive points (i.e. the edge lengths). The angles formed between each consecutive edge are the last 3 feature vector values bringing the length up to 7 (see Fig. 4). To compare feature vectors f_i and f_j they are first subtracted. The difference in angle or length is "discounted" for each generational step away from the labeled vertex to allow for the compounded effect of noise (see Algo. 2).

Map alignment could alternatively be achieved using the 4-Points Congruent Set (4PCS) method [24]. Here each feature is made up of 4 points so alignments can be calculated from a single feature (so only one is drawn each RANSAC iteration). If 4 points fall within a certain radius a feature is formed based on the relative lengths from each point to the intersection. The authors develop a method to quickly find matches between two groups of these 4-point sets. While very quick, this method suffers from many false positives when matching wireframes. See Fig. 5 which demonstrates

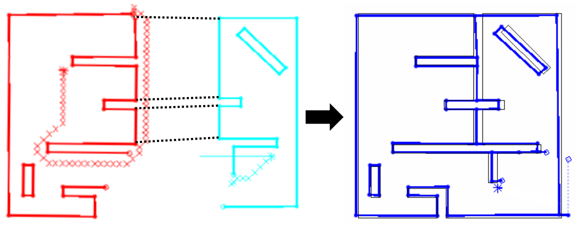
Fig. 5: The blue map on the right is the result of the red and cyan maps merging with a bad alignment due to 4PCS being "fooled" by the high degree of symmetry. The vertices on the right side of the wall (in the cyan map) are mistaken for those on the left side (in the red map). The dashed lines show the mistaken correspondences. The addition of directed edge information in the proposed feature prevents these mistakes.

where the 4PCS makes a "perfect" alignment between the chosen vertices, but not between the maps.

IV. Algorithm

The goal of this algorithm is to provide a way for the robots to completely explore the map while sharing information in a scalabale way. The wireframe map construction naturally embeds information about unexplored areas through the vertex labeling—frontier points are known to not be true vertices but simply markers of where we have seen up to along a wall that continues. The high-level control flow is shown in Algo. 1.

Algorithm 1 Control Flow for Robot r_i

```
1:  Robots in communication range, $\mathcal{C}$
2:  Queue of robots with proposed map alignments, $\mathcal{Z}$
3:  List of stored transformations $T$
4:  $\mathcal{C} \leftarrow$ getNearbyRobots($r$)
5:  $\mathcal{Z} \leftarrow \mathcal{Z} \bigcup$ alignMaps($\mathcal{C} \setminus \mathcal{Z}$)
6:  if map is empty then // EMPTY MAP
7:      drive forward
8:  else if $\mathcal{Z}$ not empty then // MAP VERIFICATION
9:      drive towards first robot in $\mathcal{Z}$, $r_j$
10:     if $\|position(r_i) - position(r_j)\|_2 < thresh$ then
11:         bAccept $\leftarrow$ judgeMapAlignment($\mathcal{Z}$.pop())
12:         if bAccept then
13:             mergeWireframes($r_i$, $r_j$)
14: else if frontier point set not empty then
15:     // EXPLORATION STATE
16:     drive towards closest frontier point
17: else// DEFAULT STATE
18:     Drive to edge with lowest likelihood
```

The robot moves straight until it finds a wall. The robot will then simply move towards the closest frontier point, which by nature of the map updating will "lead" the robot into unexplored area. Each merging of the scan will push the frontier point further away, until it vanishes when the contour is fully viewed. Eventually there will be no more frontier points and at this stage the robot can choose among several options - driving across the map to a vertex randomly sampled from its current map, "sweep" through empty area looking for previously unseen interior obstacles, follow the outer perimeter, or seek out areas of the map that have low-likelihood edges. The last approach is what is currently implemented in our simulations, though there are advantages to the sweep/ perimeter following if this work is extended to topological mapping. This is discussed further in Sec.VI.

If a robot enters into communication range it transmits its unique ID, maximum-likelihood particle (i.e. map with the highest relative weight), and current position in its own coordinate frame. The receiving robot will attempt to determine the transformation between the two maps, and if successful will use this to estimate the location of the other robot. The aligning robot will then drive to where it has estimated the other robot to be. This will take the robot to a location that is common to both maps. This way, if the map alignment is a false positive, it is highly likely conflicts will be detected along the way. By going a little out of its way the robot can ensure the map to be merged does not conflict with its own map.

If there are multiple robots in communication range, or if another robot is encountered while the first robot's alignment is being verified, then the information received from these additional robots are stored in a queue to be verified one at a time. No action is required by the transmitting robot for verification so this process is asynchronous (though the other robot may be doing the same process in reverse, depending on its current state). The most recent verified transformation is stored for each robot encountered.

Due to the randomness inherent in the RANSAC process, a robot might finish aligning and verifying another's map before the other robot has done the same. This is not a problem because the process is asynchronous, though if the robots happen to still be in communication range this transform is shared with the other robot to hot-start their next RANSAC procedure.

An important practical consideration to avoid robots getting stuck in a behavior loop is demonstrated by the following scenario. Consider two robots that enter into communication range with maps that are prone to mis-alignment. The bad alignment is calculated, which changes the path the robot was going to follow so it can instead verify this alignment. After a few steps, the verification procedure rejects the map and the robot changes its target back to its previous destination. However, the robot is still in communication range with the other, and it will immediately repeat this bad alignment and again be forced to deviate from its path to verify the alignment. To avoid this undesirable behavior loop the robots store any rejected alignments with other robots for a short period of time. Any future proposed alignments that are too similar are immediately rejected.

A. *Map Alignment*

Two pairs of corresponding points are required to define a 2D transformation. In practice, three points is much more robust to noise especially with respect to the rotation. If the

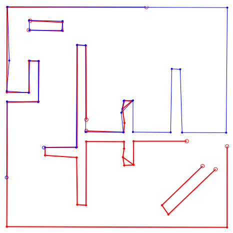

Fig. 6: Correct alignment is found using our custom feature with RANSAC, overcoming significant symmetry in point locations and noisy scan measurements that resulted in spurious line segments in both the blue and red maps.

initial shift and rotation are known to be small then optimizations that find local minima such as iterative closest point [27] can be used. However no assumptions are made about the relative transformation between the robots so a different approach is employed. The RANSAC framework is used in conjunction with the custom wireframe feature described in Sec. III-C. Each vertex is labeled with seven values, four lengths and three angles, with an order determined by the direction of the wireframe edges.

For each pair of points, the infinity norm of the weighted difference between the two vectors ($\|w \circ (f_1 - f_2)\|_\infty$) is compared to a threshold to determine if two features match. Here w is a vector of weights which is elementwise multiplied with the subtracted feature vectors to discount the elements that are more hops in the graph away from the vertex. E.g. the difference between the edge length of the child-to-grandchild node is penalized less than the vertex-to-child node distance. Sets of three are drawn from the resulting pool of potential matches, and based on these correspondences the rotation and translation is found [28]. After applying this candidate transformation the number of overlapping vertices between the aligned maps are counted to give a measure of the quality of map alignment. This is repeated for several iterations and the transformation corresponding to the largest number of matched points is returned. An example of two maps merged with this technique is shown in Fig. 6.

After an alignment is verified, it is stored for that robot. The next time these robots encounter one another, this saved transform is used to hot-start the RANSAC search for map alignment. Future alignments are only verified and merged if the proposed merge would increase the number of edges or push the bounds of the robot's map. The previously stored transform is not assumed to be perfect, and other transforms are still considered and compared which allows for mistakes to be corrected. However the previous "best" transform is always at least considered.

B. Map Merge Verification

The wireframe alignment algorithm returns the best transform found, but since each RANSAC iteration only evaluates a subset of the features it may be incorrect. To prevent a poor alignment from introducing catastrophic map errors, the receiving robot first travels to the location of the other robot which is estimated based on the transform from the map

Algorithm 2 Align Maps

```
1:  Given maps $W_1$ and $W_2$ to align
2:  $\gamma$ discount factor for second hops
3:  $w = [\gamma, 1, 1, \gamma, \gamma, 1, \gamma]$ weight vector
4:  $\beta$ threshold for accepting match
5:  $F_1$, $F_2$ sets of feature vectors
6:  for each map $W_i$ do // Match Features
7:      for each vertex $v$ in $W_i$ do
8:          if 2-hop neighbors exist then
9:              $F_i.add(getFeature(v))$ // see Fig. 4
10: for all $f_i$ in $F_1$ do
11:     for all $f_j$ in $F_2$ do
12:         if $\|w \circ (f_i - f_j)\|_\infty < \beta$ then // compare features
13:             $matches.append(f_1, f_2)$
14: for N iterations do // RANSAC
15:     $(Vpair_1, Vpair_2, Vpair_3) \leftarrow draw3(matches)$
16:     $Tf \leftarrow findTransform(Vpair_1, Vpair_2, Vpair_3)$
17:     $count = countOverlap(Tf, W_1, W_2)$
18:     if $count > maxCount$ then
19:         $maxCount = count$
20:         $Tf_{best} = Tf$
21: return $Tf_{best}$
```

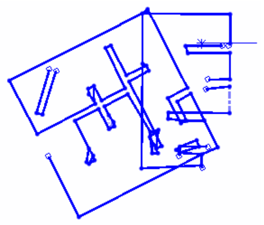

Fig. 7: Here two maps were incorrectly aligned. Due to many overlapping walls no path to the other robot's position could be found and this map merge was subsequently rejected.

alignment as well as the communicated position in the other robot's local frame. If no path to the other robot exists, this immediately identifies the map alignment as incorrect (see Fig. 7 for an example where this would occur). Otherwise, the measurements taken en route will provide evidence in favor or against accepting the map alignment. The evaluation is performed with the tools we already developed to judge the current particles in a probabilistic fashion.

Each robot already maintains a set of N weighted particles used for rejecting errors in the map. The core idea is that these particles, each of which is a map hypothesis, are used to predict the scan, and how well the particle matches the measurement strengthens/weakens our confidence that this particle is accurate. This same process can be used to judge the merged map. As shown in Fig. 8, the merged map can be treated as an additional special particle that is separate from the "regular" particles. This special particle will not be resampled (i.e. mixed with the others), nor will it be updated (i.e. new measurements will not be merged into this map hypothesis). Instead, its weight will be adjusted as future

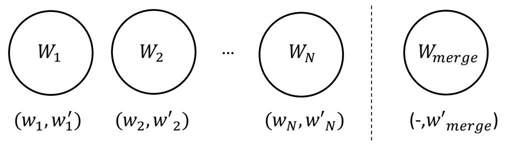

Fig. 8: On the left of the dashed line each circle represents a single particle (W_i). Each particle maintains 2 weights. The first (w_i) is used for the standard particle filter update. The second (w_i'), is referred to as the merge weight, is used to judge the map alignment. On the right of the dashed line is the single merge particle. It only has a merge weight.

scans support/contradict the information contained in this special particle as the robot travels the map.

The weight of a particle is only informative relative to the weights of all the other particles. Since we do not want this special particle affecting the "regular" particle filter, a second set of weights (called merge weights) is introduced. There will be $N+1$ merge weights and the weight for the special particle is called w_{merge}. The form of the weight update is $w_i = \eta w_i p(S|W_i)$, where η is a normalization factor and $p(S|W_i)$ is our likelihood function which is a measure of how well the scan is explained by the i^{th} particle.

Unlike the "regular" particles the merged particle is never resampled and therefore its weight is never reset. Instead it is updated by $w_{merge} = \eta_{merge} w_{merge} p(S|W_{merge})$. This normalization factor is one over the sum of all the merge weights. Each iteration all merge weights corresponding to the N "regular" particles are set to $(1-w_{merge})/N$. In other words, the remaining probability is split among the regular particles. In practice this means that the merge particle's weight is boosted when predictions based on its map prove correct and penalized when not, similar to the process for the regular particle filter. This value is normalized by the relative performance of the regular particles, whose merge weights provide a measure against which to judge the merged maps.

Upon arriving at the other robot's stored location the merge particle's weight is judged relative to the other merge weights. The alignment is accepted if the merge particle's weight is no smaller than the maximum merge weight. The other robot's shared map is then simply treated as a new scan and is merged into each real particle. The merged particle and all the merged weights are discarded.

There is an important practical consideration regarding the likelihood function. It is not desired to treat newly explore area as a contradiction to the existing map, even though newly discovered edges are not predicted. Directions from the robot with no known edges are called "gaps", and edges/vertices appearing in the scans that fall into these gaps are not penalized as false positives. Otherwise any new exploration by the robot would be immediately penalized. However, this has the effect of putting the merge particle at a disadvantage. There will always be some noise in the locations of edge endpoints, and there are almost always more edges in the merged particle. This means the regular particles are not penalized for unanticipated edges in gaps

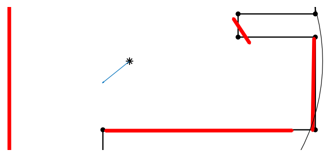

Fig. 9: A typical scan artifact (beveled corner) resulting from realistic sensor limitations. The scan is red and the true walls are black. The circle shows the sensor's field of view.

while in the merge particle these edges may be expected and therefore subject to a likelihood penalty due to noise. To prevent this, only the Bernoulli distribution (based on the existence/non-existence of edges) is used for calculating the likelihood when updating the merge weights. Only the binary prediction about the existence of edges affects this weight, not the offset of vertices that is usually due to noise. Since even relatively small map misalignments, like the one shown in Fig. 5, introduce several incorrect edges disregarding the noise proves to still be an effective judge of map alignment.

C. Wireframe Navigation and Exploration

After Algo. 1 calculates a goal point for the robot to reach (e.g. a frontier point or the stored previous location of another robot), a simple A* search over a temporarily-superimposed grid is used to generate a collision-free path for the robot to reach the goal. However, any online planning algorithm with work for this, including sampling based methods such as RRT* [29], to classical methods like the bug algorithm [30], or methods based on computational geometry. Examples of this last category include visibility graphs which are guaranteed to find the shortest free path in $\mathcal{O}(n^2 \log |V|)$ time, or trapezoid decomposition, which creates an efficiently searchable structure in ($\mathcal{O}(n \log |V|)$) that can be used for collision free navigation [31]. Once a trajectory is generated, the robot can use its low-level control system to follow this path. In our Gazebo simulations, a simple PID controller is used to guide the TurtleBots along their planned paths.

V. Results

The advantages of the wireframe map representation compared to standard techniques are established in [1]. The simulations results in this paper establish the effectiveness of the decentralized control strategy for both environment exploration and collaboration. Simulations were run with first MATLAB then ROS-Gazebo. The ROS-Gazebo simulations allow the algorithm to be challenged by more realistic sensor noise. Fig. 9 highlights a common issue when extracting lines from a noisy laser scan. Where there should be two right angles the end is cutoff. Furthermore, the dynamics of the robots are now simulated in addition to the sensing.

The first simulation establishes a baseline with a single robot, as well as demonstrates that though this algorithm is developed for groups of robots the exploration strategy relying on frontier points is effective for a single robot as well. A maze was generated in a 15m$\times$15m space, which is large compared to the small TurtleBot. Fig. 10 shows the

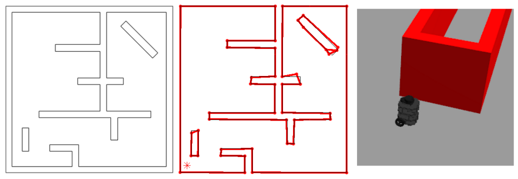
Fig. 10: Left: ground truth wireframe map (15m×15m). Middle: final maximum likelihood map of a single robot. Right: Screenshot from the simulation of the TurtleBot.

ground truth wireframe map, the result of a single robot exploring the environment, and a screenshot of this robot as it is mapping. All simulated robots maintain 5 particles, have 5m sensing radius, and a 8m communication radius.

The second simulation was run using the same 15m×15m environment, but this time with 3 robots and MATLAB as the simulator. The focus of this simulation was to test the effectiveness of both the cooperation and the proposed alignment feature. Fig. 11 shows the maximum likelihood map for each of the three robots upon completing the task. During this run, every time two maps were compared for alignment we used the 4PCS method as well as the wireframe-based feature method. The 4PCS method was as likely to succeed as to fail. Judging a group of four points drawn from a set of sparse corners is simply not discriminating enough (any rectangle can be matched 2 ways and any square can be matched 4 ways), and most results were false positives. While the verification method was effective at preventing these incorrect alignments from being incorporated into the map, this caused the robots to spend longer ruling out bad alignments instead of exploring. The wireframe-based feature is not immune to all of these false-positives, but has two advantages. The first is that some matches can be ruled out using the edge information. The second is that the stricter standard for what constitutes a matched pair results in fewer pairs to consider. Even if some incorrect alignments are chosen during RANSAC, it is more likely that the correct alignment is also considered (which should have more overlapping vertices and therefore be chosen).

In order to characterize the effect of the number of robots and sensing range, simulations were run on the 15m×15m map with 1, 2, 3, 5, and 8 robots with sensing ranges of 1.5m, 3m, and 5m. For all runs the communication range was 7m and the average vertex-detection noise was 40mm. Table I shows the average number of iterations for the first robot to complete its map. Since the robot's step size is constant, the number of iterations is a measure of distance traveled. This demonstrates that the extra distance traveled to verify map alignments is more than compensated for by the shared

TABLE I: Average Simulation Iterations to Completion

Avg. Iters.	1 robot	2 robots	3 robots	5 robots	8 robots
$1.5m$ range	342	250	212	172	134
$3m$ range	258	230	143	101	79
$5m$ range	197	138	80	65	63

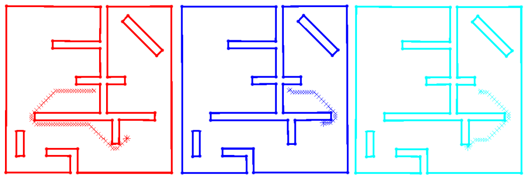
Fig. 11: Each robot's maximum likelihood particle (wireframe map) upon completion of the MATLAB simulation.

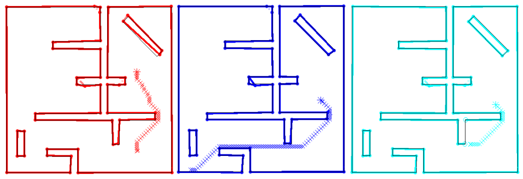
Fig. 12: Each robot's maximum likelihood particle (wireframe map) upon completion of the Gazebo simulation. The increased noise is clear, but the resulting maps are still very close to ground truth (underlaid in black).

information. There are diminishing returns to increasing the number of robots (for a given map size), which is intuitive as the robot's have less new information to share in a given merge and more robots with which to share. The data show that a longer sensor range leads to quicker simulations. This is likely explained by the fact that it is more work to map an area therefore it is more likely that the shared information is new to the other robot.

The number of robots, starting locations, and wall layout of the final simulation were identical to the second, except now the simulator used is ROS-Gazebo. The final maps contain more noise, and more realistic sources of noise, than the MATLAB-simulated results. Evidence of scans incorrectly seeing beveled corners are clear. After mistakenly seeing the beveled edge, often the correct edges are seen as the robot approaches this thin obstacle (this is due to an increased density of laser points falling on closer edges). This has the effect of “trapping” the incorrect cut-off edge inside of the correct edges. This incorrect edge will never be predicted to be seen (since it is always now behind a wall) so it will never be penalized by the weight update and therefore is unlikely to ever be removed. This is not a serious problem for two reasons. The first is that the purpose of this map representation is for the robot to have a lightweight tool that can be used for navigation, which is not affected by these internal edges. Furthermore, if they did somehow become a problem, the robot can identify them once there are no frontier points left in its map. The absence of frontier points means there are no more open contours. Any edge with an endpoint with more than two neighbors can be evaluated by counting the number of intersections along a line segment connected from middle of the tested edge (not counted as an intersection) to a point outside the bounds of the map. If the number of intersection is odd then the edge can be removed (this procedure is know as the point-in-polygon algorithm).

These simulations demonstrate the effectiveness of the algorithm, but the benefit should scale with the size of the map. As the map becomes larger, the maximum distance for scan verification does not increase (as this is only related to the communication radius). Therefore the amount of information gained vs. trajectory detour increases.

VI. Conclusion

In this paper we propose a distributed algorithm for an arbitrary number of robots to efficiently map an environment. A sparse representation and update framework are extended to include control and collaboration. To achieve this, a new feature is proposed that leverages the directed edges of the wireframe to avoid the pitfalls of false-positive matches introduce by storing only sparse data. This algorithm is shown to work in a high fidelity simulation environments, where realistic scan errors are handled by the robustness of the particle filter. Single agent and multi-agent simulations are run, and while both result in a complete map of the environment, the multi-robot situation is faster and more robust. As an additional byproduct, the relative robot frames are determined in the course of executing the algorithm.

Ongoing work is underway to implement this methodology on our in-house designed Ouijabot robots [32] to further demonstrate the effectiveness of this algorithm in realistic situations. We also intend to extend this methodology to three dimensional environments where the ease of low-computational navigation and sparse environment representations become even more critical. Finally, this algorithm can also be extended to environments that have disjoint subregions (e.g. some doors are closed creating groups of rooms between which the robots cannot travel). In this situation the robots in each area can construct a wireframe of their reachable environment while using the techniques in [33] to further build a topological map of the larger space.

References

[1] A. Caccavale and M. Schwager, "Wireframe mapping for resource-constrained robots," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018, pp. 1–9.

[2] 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.

[3] D. Eppstein, "Subgraph isomorphism in planar graphs and related problems," in *Graph Algorithms And Applications I*. World Scientific, 2002, pp. 283–309.

[4] 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.

[5] J. Dong, E. Nelson, V. Indelman, N. Michael, and F. Dellaert, "Distributed real-time cooperative localization and mapping using an uncertainty-aware expectation maximization approach," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 5807–5814.

[6] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 376–386, June 2005.

[7] N. Atanasov, J. Le Ny, K. Daniilidis, and G. J. Pappas, "Decentralized active information acquisition: Theory and application to multi-robot slam," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 4775–4782.

[8] V. Indelman, "Cooperative multi-robot belief space planning for autonomous navigation in unknown environments," *Autonomous Robots*, vol. 42, no. 2, pp. 353–373, 2018.

[9] B. Schlotfeldt, D. Thakur, N. Atanasov, V. Kumar, and G. J. Pappas, "Anytime planning for decentralized multirobot active information gathering," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1025–1032, April 2018.

[10] S. Liu, K. Mohta, S. Shen, and V. Kumar, "Towards collaborative mapping and exploration using multiple micro aerial robots," in *Experimental Robotics*. Springer, 2016, pp. 865–878.

[11] S. Thrun and Y. Liu, "Multi-robot slam with sparse extended information filters," in *Robotics Research. The Eleventh Intern. Symposium.*, vol. 15. Springer Berlin Heidelberg, 2005, pp. 254–266.

[12] J. S. Liu and R. Chen, "Sequential monte carlo methods for dynamic systems," *Journal of the American statistical association*, vol. 93, no. 443, pp. 1032–1044, 1998.

[13] P. Del Moral, "Nonlinear filtering using random particles," *Theory of Probability & Its Applications*, vol. 40, no. 4, pp. 690–701, 1996.

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

[15] 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.

[16] 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.

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

[18] 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.

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

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

[21] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative multi-robot exploration," in *2000 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, April 2000, pp. 476–481.

[22] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*. IEEE, 1997, pp. 146–151.

[23] ——, "Frontier-based exploration using multiple robots," in *Proceedings of the second international conference on Autonomous agents*, vol. 98, 1998, pp. 47–53.

[24] D. Aiger, N. J. Mitra, and D. Cohen-Or, "4-points congruent sets for robust pairwise surface registration," in *2008 ACM Transactions on Graphics (TOG)*, vol. 27, no. 3, 2008, p. 85.

[25] S. Belongie, J. Malik, and J. Puzicha, "Shape context: A new descriptor for shape matching and object recognition," in *Advances in neural information processing systems*, 2001, pp. 831–837.

[26] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[27] 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.

[28] O. Sorkine, "Least-squares rigid motion using svd," *Technical notes*, vol. 120, no. 3, p. 52, 2009.

[29] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, vol. 104, no. 2, 2010.

[30] F. Bullo and S. L. Smith, *Lectures on Robotic Planning and Kinematics*, 2015.

[31] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf, "Computational geometry," in *Computational geometry*. Springer, 1997, pp. 1–17.

[32] Z. Wang, G. Yang, X. Su, and M. Schwager, "Ouijabots: Omnidirectional robots for cooperative object transport with rotation control using no communication," in *Distributed Autonomous Robotic Systems (DARS)*. Springer, 2018, pp. 117–131.

[33] A. Caccavale and M. Schwager, "A distributed algorithm for mapping the graphical structure of complex environments with a swarm of robots," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 1459–1466.