Efficient Dense Frontier Detection for 2D Graph SLAM Based on Occupancy Grid Submaps

Juraj Oršulić*, Damjan Miklić† and Zdenko Kovačić*

*University of Zagreb
Faculty of Electrical Engineering and Computing, Laboratory for Robotics and Intelligent Control Systems
† RoMb Technologies d.o.o., Zagreb
Email: juraj.orsulic@fer.hr

Abstract—In autonomous robot exploration, the frontier is the border in the world map between the explored space and unexplored space. The frontier plays an important role when deciding where in the environment the robots should go explore next. We examine a modular control system pipeline for autonomous exploration where a 2D graph SLAM algorithm based on occupancy grid submaps performs map building and localization. We provide an overview of the state of the art in frontier detection and the relevant SLAM concepts and propose a specialized frontier detection method which is efficiently constrained to active submaps, yet robust to SLAM loop closures.

I. INTRODUCTION

A fully autonomous mobile robot, able to explore, navigate and perform actions in an unknown environment is the ultimate objective of today’s mobile robotics research. To this end, we consider a single autonomous robot or an autonomous team of robots tasked with exploring the unknown environment. The autonomous exploration problem comprises collecting the data sensed from the environment, using the collected data to build a structured model of the environment, self-localizing in the environment model, high-level planning and scheduling of robot tasks (mission generation and assignment), and path planning and following. All of these need to be performed in real time.

In a common approach known as frontier exploration, the robot maintains information about the border which divides the explored and unexplored space in the environment – the frontier. Elements of the frontier represent places in the environment which the robot may approach and thereby increase the knowledge about the structure of the environment. With the information about the exploration frontier available, mission planning can be described in its simplest version as (boldly) go where no one has gone before.

Many components of the autonomous exploration problem mentioned above are complex enough to be associated with their own field of robotics research, resulting in sophisticated methods and software modules being available for solving them. Namely, building the model of the environment (a map) and self-localization therein may be performed by a module implementing a Simultaneous Localization and Mapping (SLAM) algorithm. For this reason, we consider a control system implementing a modular exploration pipeline as depicted in Figure 1. Map building and localization are performed by a SLAM module. The SLAM results – a map and a robot pose – are processed in a frontier detection module. The detected frontier is then used by a high-level exploration task generation and scheduling module, which creates and selects exploration tasks to be executed according to an exploration strategy, and further requests the path planning and following modules to execute the exploration tasks.

The main contribution of this paper is a piece of the exploration pipeline – a new, efficient frontier detection approach, specialized for use with a 2D graph SLAM algorithm based on occupancy grid submaps [1]. The proposed approach is

---

Fig. 1. Block diagram providing a high-level overview of the proposed control system for performing autonomous exploration. The robot may be equipped with a laser rangefinder, an inertial measurement unit, and wheel encoders for odometry. A Simultaneous Localization and Mapping module uses the sensed environment data to build an environment map and to estimate the pose therein, while a frontier detection module keeps track of the exploration frontier. An exploration task generation and scheduling module assigns exploration tasks to be performed according to an exploration strategy and forwards it to a path planning and following module, which uses the map and localization from SLAM to steer the robot towards the goal defined by the assigned task. Notably, the frontier detector is tightly coupled with the SLAM module, enabling an efficient implementation of frontier detection.
robust to loop closures and exploits the submap structure of the SLAM algorithm in order to quickly perform frontier updates. By providing high frequency incremental frontier updates which enable more responsive planning of exploration objectives, it facilitates a real-time use case on large and complex maps, e.g. the Deutsches museum dataset [1]. All the while the proposed frontier detection algorithm delivers a result at least as good as a naive frontier edge-detection algorithm, i.e. performing edge detection on a completely assembled global map each time after SLAM updates the map by inserting scans or optimizing the pose graph to perform loop closures.

II. RELATED WORK

A. Frontier Exploration as a Prevalent Exploration Method

Frontier exploration in the context of autonomous robotics was first introduced by Yamauchi in 1997 [2], paving the way for many others ([3, 4, 5]). Commonly, elements of the detected frontier are used as navigation goals during planning of exploration tasks. Building on this, there are more complex exploration strategies which attempt to coordinate entire robot teams (6, 7), or use frontiers as sinks in a potential field (8). Frontier detection is therefore a key elementary operation in frontier exploration, and it is important that it be performed as quickly as possible so that exploration can be more efficient [9].

B. State of the Frontier Detection Art

A naive algorithm for frontier detection is to perform edge detection on the complete global map after each map update. However, this approach is not feasible for larger maps and real-time robot operation with such maps, as it presents a significant computational burden.

1) Keidar and Kaminka’s seminal work on efficient frontier detection: Keidar and Kaminka [10] proposed in 2014 several approaches which attempt to perform frontier detection in an efficient manner. The first, Wavefront Frontier Detector (WFD), consists of running two consecutive breadth-first searches (BFS). The first BFS starts at the robot position and continues throughout the free space, until eventually a frontier point is found which belongs to a component of connected frontier points. From there, the rest of the connected component is found by a second BFS along the connected frontier points. While WFD avoids searching the unoccupied space, it still searches all observed free space in each iteration, which may degenerate into a full map search as exploration progresses.

The second approach to frontier detection proposed by Keidar and Kaminka, the Fast Frontier Detector (FFD), does not use the map built by SLAM, but rather constructs the contour of each laser scan using Bresenham’s line algorithm, and uses the constructed contour to detect the frontier and store it in a specialized data structure. Quin and Alempijević [9] note that FFD has to be executed after each scan, which results in many wasteful calculations if frontier updates are required only occasionally, and that Bresenham’s line algorithm can cut across unobserved space and miss some frontier cells. The proposed approach does not require execution after each processed scan, supporting a use case where frontier updates are required only occasionally.

FFD is also notable for introducing the concept of active area – a bounding box positioned in the map around the robot position, circumscribing the last scan the map was updated with. The frontier update step is sped up by restricting it to the active area. Keidar and Kaminka also applied this concept to the WFD detector, yielding the incremental WFD (WFD-INC) algorithm, which requires non-trivial auxiliary data structures for frontier point maintenance. The proposed algorithm uses a similar concept of active submaps.

2) Impact of loop closure in SLAM on frontier detection: Loop closure is an event when the SLAM algorithm recognizes that the robot has revisited the same place, and then makes a correction using this information which reduces the error caused by drift in localization along the whole loop. The frontier detector has to be able to efficiently cope with the map changes induced by loop closure corrections. These map changes may not be confined to the active area - in fact, loop closure may result in widespread changes all over the map. While an efficient frontier detection algorithm should avoid reassembling and iterating throughout the entire global map in every iteration, constraining the algorithm to only the active area makes it difficult to be robust to loop closures. WFD-INC addresses loop closure events by evicting the detected frontier and performing frontier detection from scratch using the new loop-corrected map.

To efficiently address loop closures, the frontier detection algorithm needs to get intimate to a certain degree with the implementation of the SLAM algorithm. For example, Keidar and Kaminka additionally proposed an implementation of WFD-INC for GMapping (a particle filter-based SLAM [11]) called incremental parallel WFD (WFD-IP). WFD-IP performs in parallel separate WFD-INC frontier detection for each particle (each particle having its own map), and outputs the appropriate frontier of the current best particle. Like WFD-INC, the proposed method uses the internals of the SLAM algorithm in order to perform frontier detection faster while also being robust to loop closure.

Quin and Alempijević [9] introduce two frontier detection methods: naive active area (NaiveAA), which is the naive approach confined to the active area, and a version of WFD called Expanding WFD (EWF) which steers the WFD breadth-first search into newly discovered free areas. EWF assumes that the entropy for each cell can only decrease over time. This is not true in the general case for the complete global map when considering effects of loop closure – observed areas can get moved around in the global map during loop closure and leave unexplored space in their wake. However, the entropy decrease assumption is almost surely true for single submaps in submap-based SLAM, and we exploit this fact in the proposed approach.

3) Other approaches: Senarathne and Wang [12] use an oriented bounding-box based inexact approach.
Umari [13] uses rapidly-exploring random trees (RRT) to perform sparse frontier detection by building a tree inside the free space in the SLAM-built map. When the algorithm crosses the frontier while trying to expand the random tree, a single frontier point is detected. However, using the implementation of the algorithm provided in [13] does require reassembling the global map in each iteration. Also, this algorithm is not robust to loop closure, since the built RRT tree does not follow the results of pose graph optimization.

Experiments were performed with a modified version of the RRT frontier detection algorithm in which the tree nodes were bent according to the displacement of the closest submap in the optimized pose graph in an attempt to make the algorithm robust to loop closure. However, narrow corridors in the map have proven problematic as the map size increases, because the probability of extending the tree into a narrow corridor dramatically decreases as the map canvas size increases. An inspiration for the proposed method was attempting to perform dense local frontier detection (opposed to sparse, as with RRT), while trying to follow the global “dance” of the pose graph as it is optimized.

III. PREREQUISITES

A. Simultaneous Localization and Mapping

The term SLAM was coined by Leonard and Durrant-Whyte in 1991 [14]. As shown in the block diagram of the exploration pipeline in Figure 1, a SLAM algorithm uses sensor data to build a map and perform localization, which is further used in frontier detection, exploration task planning and execution. There is a wealthy trove of SLAM methods developed to this day, which can be roughly grouped into methods based on filtering and methods based on graph optimization.

We will focus on graph SLAM, which represents poses and detected features as nodes in a graph, while the correspondences which impose constraints on the poses of the respective nodes are represented as edges. Various optimization methods may be used to minimize the residual error of all constraints, e.g. the Ceres solver [15].

Submaps are small local maps which are merged into a global map. One of earlier approaches to SLAM using submaps is [16], with further examples being [17] and a graph SLAM approach using histograms of submap features [18].

B. Cartographer

The proposed frontier detection method was designed for use with Cartographer, an open-source multi-robot multi-trajectory 2D and 3D graph SLAM based on occupancy grid submaps, developed by Google (Hess, Kohler, Rapp in 2016 [11]). Cartographer’s approach of optimizing the poses of all scans and submaps follows Sparse Pose Adjustment [19] and uses the Ceres solver [15] for optimizing the pose graph using the Levenberg–Marquardt algorithm (LMA).

Cartographer submaps are spatially and temporally compact occupancy grid maps made from a short, continuous series of rangefinder sensor measurements (laser scans) taken during traversal of a short section of the robot trajectory. It is desired that the size of the submaps be small enough such that the localization drift is not perceptible when looking at one submap at a time. Building the submaps and the trajectory (without loop closures) is handled by Cartographer’s local trajectory builder component, which maintains a pair of active submaps that the laser scans are inserted into according to a local pose obtained by performing scan matching against the older (larger) submap from the active pair.

When a predetermined number of scans is inserted into a submap, it is marked as finished, and a new submap is created to take its place in the active submap pair. Importantly, once a submap is finished, its occupancy grid is immutable from that point onward.

Cell occupancy probabilities are clamped to the interval [0.1, 0.9] and are stored linearly mapped onto the space of unsigned 16-bit integers. Scan insertion into submaps is performed as Bayesian updates of the cell occupancy probabilities (see (3) in [1]). The cells corresponding to laser hit points are updated with “occupied” observations, while the intermediate points (obtained by casting rays from the laser rangefinder origin to hit points) are updated with “free” observations. It is also important to note that on a level of a single submap, the cell entropy (i.e., the uncertainty of the occupancy probability) can be assumed to be monotonically decreasing. If there are moving objects in the environment (e.g. a door which was previously closed is now open), it is theoretically possible that an observed cell may afterwards land in the narrow occupancy probability interval near 0.5 where it would be again considered unobserved. However, the probability of this is negligible, and it is considered that the cells of an active submap cannot become unobserved once they are observed.

When loop closures are detected by the constraint builder component of Cartographer, pose constraints between the corresponding trajectory nodes and submaps are added as edges into the pose graph. Afterwards, optimization is periodically invoked in order to find a new solution—a set of global submap and trajectory node poses—which minimizes the residual costs of the constraints. As discussed, for frontier detection, this implies that when pose graph optimization is performed, the submaps can and do get displaced and rotated (i.e. undergo rigid transformations), although their occupancy grids are immutable after they are marked as finished. The proposed frontier detection approach attempts to take advantage of these properties of Cartographer.

IV. FRONTIER DETECTION

A. Definitions

Rigid transformation $T^{b}_{a} \in \text{SE}(3)$ is the pose of the coordinate system $b$ relative to the coordinate system $a$. The $\text{SE}(3)$ group operation of pose composition is written as multiplication, e.g. $T^{a}_{b} T^{b}_{c}$. The transform inverse $(T^{a}_{b})^{-1}$ is equal to $T^{b}_{a}$. The projection of a point with coordinates expressed in $b$, $p^{b} = [x \ y \ z]^{T} \in \mathbb{R}^{3}$, to the corresponding point $p^{a}$ in coordinate system $a$ is denoted as $p^{a} = T^{a}_{b} p^{b}$. 
The global map coordinate system is denoted with $g$. The solution of pose graph optimization are poses of graph members expressed with respect to $g$.

Submap is an occupancy grid of resolution (i.e. cell size) $r$, which is typically 0.05 cm. The occupancy probabilities of grid cells are initially unobserved i.e. unknown (exactly 0.5). Submaps are constructed by insertion of $n_{scans}$ sequential laser scans, where parameters $r$ and $n_{scans}$ are predetermined fixed parameters. A submap is marked as finished when $n_{scans}$ scans have been inserted into the submap.

The set of active submaps contains the submaps which are not yet finished.

The local submap coordinate system has the origin fixed, for example, to the robot pose in the first scan inserted into the submap. The cells of a submap occupancy grid are indexed in a 2D matrix using a 2-integer tuple: $S_{k,l}^{si}$ is the occupancy probability value of the cell $(k,l)$ in the submap $si$. As the submap grows in size, the 2-integer tuples corresponding to the same cell in the submap may change (which is an implementation-specific detail), but the position of the cell in the local submap coordinate system must remain the same.

Global submap pose $T_{g}^{si}$ is the global pose of the origin of the local coordinate system of a submap $si$. Since submaps are members of the pose graph, submap poses are part of the optimized pose graph solution.

Occupancy classification – in the frontier detection algorithm, the cell occupancy probability values are not used directly. The probability values are first classified according to the following thresholding rule:

$$\text{class}(p = S_{k,l}^{si}) := \begin{cases} 
\text{free} & p < 0.5 \\
\text{occupied} & p > 0.5 \\
\text{unobserved} & p = 0.5 
\end{cases} \quad (1)$$

Observed cells are occupancy grid cells that are not unobserved.

Local frontier point is the center of an unobserved occupancy grid cell which is adjacent to a free cell in the same submap.

Local frontier of a submap is the set of its local frontier points. See the red points in the first two pictures in Figure 2 for an example.

Stabbing query refers to looking up the corresponding cell in a given submap for a given global point and the global submap pose. More precisely, for a submap $si$ and a global point $p^g \in \mathbb{R}^3$ expressed in $g$, to find the occupancy grid cell $(k,l)$ in $si$ whose center coordinates in the local coordinate system of the submap $si$ are closest to $(T_{g}^{si})^{-1} p^g$. Further, performing a stabbing query test means checking if the corresponding cell $S_{k,l}^{si}$ is an unobserved cell, in which case the test passes.

Global frontier point is the center of an unobserved global occupancy grid cell adjacent to a free global map cell. A valid global frontier point must pass the stabbing query test against all other submaps.

Global frontier is the set of all valid global frontier points at a given time. See the third picture in Figure 2.

Perimeter of the global or local frontier is the number of frontier points in the respective set.

It may be noted that the frontier points could alternatively have been defined as centers of free cells adjacent to unobserved cells. We have chosen to define frontier points as centers of unobserved cells as above in order to simplify the stabbing query test.

B. The frontier detection algorithm

There are two kinds of events which occur during SLAM execution which are of interest for frontier detection: a submap update event, where a scan is inserted into the active submaps; and a pose graph optimization event which also occurs periodically, but less often. Algorithm 1 describes handling of submap update events, while Algorithm 2 describes handling of pose graph optimization events.

1) Handling of submap updates: For each laser scan processed in SLAM, both submaps in the active pair are updated, making submap updates the most frequently occurring type of event. A submap update can only affect the frontier in the area covered by the active submaps, therefore the frontier detection algorithm can be constrained to this area in order to maximize efficiency.

The first step in handling a submap update of an active submap is performing dense local frontier detection on the new version of the submap occupancy grid (Algorithm 1 lines 6-10).
vectorized, allowing for high performance on modern CPUs. Our implementation relies on Eigen [20] for vectorization, where we used matrix block algebra and Hadamard products to implement thresholding, classification and Boolean logic for edge detection.

Computing and storing the set of local frontier points for an active submap produces a set of candidates for the global frontier. Every local frontier point is projected into the global map frame according to the current global pose of the submap against which the failing submap hint in most cases immediately produces a negative result. The reason for using a naive approach for local frontier detection is twofold. First, since the submaps are bounded in size (controlled by the fixed parameter $n_{scans}$), and the number of active submaps is constant, the time complexity of local frontier detection is not affected by the size of the global map or the size of the dataset. Second, probability thresholding, classification (line 6) and edge detection (line 8) can be

In other words, on the local submap level, we have opted to perform a naive edge detection approach.

The reason for using a naive approach for local frontier detection is twofold. First, since the submaps are bounded in size (controlled by the fixed parameter $n_{scans}$), and the number of active submaps is constant, the time complexity of local frontier detection is not affected by the size of the global map or the size of the dataset. Second, probability thresholding, classification (line 6) and edge detection (line 8) can be
2) The algorithm for handling submap updates may assume that no graph optimization has occurred since the last submap update event, so all existing global frontiers of finished submaps are valid (except for the situation described below).

3) The cell occupancy probabilities of active submaps have decreasing entropy, which means that only previously unobserved cells can become observed, and not vice-versa. This implies that updates to active submaps can invalidate the global frontiers of intersecting submaps. In lines [14][17] it is tested if the new versions of active submaps cover existing valid global frontiers of the intersecting finished submaps. This is done by performing stabbing query tests of the global frontier points against the active submaps. If the stabbing query test fails, the newly covered global frontier points are removed from the set of global frontier points (line 17), thus preserving the invariant that all global frontier points at a time are valid. The global frontiers of finished submaps whose global frontier points got removed are also marked as updated (line 18) for incremental publishing of frontier updates (line 22).

Bounding boxes of finished submaps are stored inside a tree data structure which enables fast queries of submaps which intersect with a given bounding box (e.g. looking up finished submaps intersecting with an active submap, line 5). When a submap is marked as finished, its bounding box is inserted into the tree structure (line 19). Our implementation uses the Boost implementation of R-trees [21] for storing global axis-aligned bounding boxes of finished submaps.

It can also be noted that not all submap update events have to be handled in order to guarantee a correct result – any non-final submap update can be skipped. A robotic exploration system which does not require real-time frontier updates after every scan, but rather occasionally, can invoke the algorithm for handling submap updates only when a submap is finished. This can significantly reduce the computational effort of keeping the frontier up to date, up to a factor of \( n_{\text{scans}} \).

4) Handling of graph optimization events: When graph SLAM performs optimization, a new solution is produced for poses of all graph members. For frontier detection, this means that the submap poses have changed. This invalidates the global bounding boxes of submaps and the entire global frontier, all of which has to be recomputed in Algorithm 2. However, advantage is taken of the fact that the local frontiers have already been computed for all submaps, so all that needs to be done is to re-project the local frontier points to the global coordinate system \( g \) and re-test them.

The global bounding boxes are recomputed in lines [1][3] while the rest of Algorithm 2 recomputes the global frontier similarly to Algorithm 1. Each local frontier point is re-projected according to the corresponding new global submap pose (line 8) and the stabbing query test against the intersecting submaps is performed (line 11). If the stabbing query test is passed, the re-projected and re-tested frontier point is inserted into the new global frontier set (line 12).

Note that during re-testing, it is advisable to first try testing against the failing submap hint, if it exists (comment in lines [9][10]. This speeds up rejection of points which fail the test by first testing against the same submap which caused the same local frontier point to fail the test earlier. Even though recomputing the entire global frontier in Algorithm 2 is certainly not a computationally lightweight operation, it is still more efficient than any other non-submap aware approach which would require iterating through the entire reassembled global map, and thus have a time complexity proportional to the area of the global map. Recomputing the global frontier in Algorithm 2 has a time complexity proportional to the perimeter of local frontiers.

To summarize, Algorithm 1 assumed that all global frontier points before a submap update event were valid and up-to-date. Graph optimization violates this assumption by displacing the submaps according to submap poses of the new pose graph solution, Algorithm 2 restores this invariant by recomputing the global frontier.

V. ALGORITHM ANALYSIS
A. Soundness and completeness

Let us first note that the process of merging submaps into a global map according to a pose graph solution cannot result in an unobserved cell in the global map if any of the corresponding submaps are observed. In other words, if a global map cell is unobserved, the corresponding cells in all submaps are also unobserved. Also, let \( LF \) and \( GF \) denote the sets of local and valid global frontier points of all submaps.

For discussing completeness, we will consider a valid global frontier point in \( GF \), i.e. the center of an unobserved global map cell adjacent to a free global map cell. Due to the map merging process, all submap cells corresponding to the global frontier cell have to be unobserved as well. Next, the adjacent free global map cell is also free in at least one submap, whose merging caused the free global map cell to be marked as such. In that submap, the unobserved cell next to the free cell is a local frontier and is in \( LF \). Therefore, each valid global frontier point corresponds to a local frontier point in at least one submap (i.e. \( GF \subseteq LF \) projected to \( g \)), and an algorithm which computes the set of global frontier points \( GF \) by taking the set of all local frontier points \( LF \) as input is thus complete, providing it does not incorrectly discard any valid global frontier points in the process. The rule for discarding local frontier points when they fail the stabbing query test can only result in observed cells being correctly discarded from the global frontier, and therefore the property of completeness is preserved. Also, the naive edge detection algorithm used for computing the local frontiers is trivially valid.

For soundness, we will consider a global frontier point returned by the proposed algorithm, and suppose that it is not valid. This could be either because the returned global frontier point is not an unobserved cell in the global map, or because the returned global frontier point is not adjacent to a free cell in the global map. The first case is not possible because the frontier point would have failed the stabbing query test against the submap that contains an observed cell which resulted in
marking the corresponding global cell as observed. The second case is theoretically possible – for example, if there were a plurality of submaps with corresponding occupied cells rather than free, so the merging process results in adjacent global map cells being marked as occupied instead of free. However, this case is of no practical significance, because the chance that this will happen without the unobserved frontier cell also getting covered up by observed cells (which in turn will make it fail the stabbing query test) is negligible. To exclude this case, the stabbing query test could be modified to also look at the adjacent cells in submaps, instead of just checking if the single cell corresponding to the frontier is unobserved. This would not increase the theoretical time complexity, but would make the implementation unnecessarily more elaborate and slow.

B. Computational complexity

1) Handling of submap updates: In [Algorithm 1 line 5] the complexity of querying the R-tree for submaps intersecting with the updated submap \( s_i \) is \( O(\log |S| + |S_{\cap s_i}|) \), where \( S \) is the set of all submaps, and \( S_{\cap s_i} \) is the set of submaps intersecting with the submap \( s_i \). This also includes the complexity of inserting a finished submap bounding box into the R-tree.

In lines [6][8] naive local frontier detection runs in \( O(A(s_i)) \), where \( A(s_i) \) is the area of submap \( s_i \), i.e. the number of cells in the submap.

The number of local frontier points in \( s_i \), i.e. the perimeter of its local frontier \( LF_{s_i} \), will be denoted as \( P(LF_{s_i}) \), while the global frontier of the submap \( s_i \) and its perimeter will be denoted as \( GF_{s_i} \) and \( P(GF_{s_i}) \), respectively.

For each detected local frontier point in the updated submap, the complexity of performing the stabbing query test against the intersecting submaps (line [11]) is \( O(|S_{\cap s_i}|) \), yielding the complexity of this step \( O(P(LF_{s_i}) \cdot |S_{\cap s_i}|) \).

Invalidating global frontiers of the intersecting submaps (lines [14][17]) runs linearly with respect to their perimeter, i.e. in time \( O(P(\bigcup_{s_j \in S_{\cap s_i}} GF_{s_j})) \).

The total time complexity of handling an update of submap \( s_i \) thus equals (simplified assuming \( P(LF_{s_i}) \geq 1)\):

\[
O(\log |S| + A(s_i) + P(LF_{s_i}) \cdot |S_{\cap s_i}| + P(\bigcup_{s_j \in S_{\cap s_i}} GF_{s_j}))
\]  

(2)

2) Handling of pose graph optimization events: The first step in [Algorithm 2] is to rebuild the R-tree (lines [1][5]), which runs in \( O(|S| \log |S|) \). This complexity also covers looking up the intersecting submaps for each submap (line [6]).

Next, every local frontier point is projected to the global coordinate frame \( g \) and the stabbing query test is performed against the intersecting submaps (lines [8][11]). A pessimistic bound would entail testing every local frontier point against all other submaps, i.e. a time complexity of \( O(|S| \cdot P(LF)) \). The pessimism of this bound can be reduced by assuming that the points which fail the stabbing query test will do so on the first test, performed against the failing submap hint. If their perimeter, equal to \( P(LF) - P(GF) \), is denoted as \( P(FF) \), this assumption yields the time complexity of \( O(|S| \cdot P(GF) + P(FF)) \).

The total time complexity of handling a pose graph optimization event thus depends on the number of submaps and the local and global frontier perimeters:

\[
O(|S| \cdot (\log |S| + P(GF)) + P(FF))
\]  

(3)

We have managed to avoid having the two-dimensional map area in the time complexity of the global operation of handling pose graph optimization by taking advantage of submaps and their immutability.

VI. EXPERIMENTAL RESULTS

Our implementation of frontier detection, available online on Github[2] has been tested on an Intel i7 6800K CPU running Ubuntu 18.04. A demo video of processing Google’s Deutsches museum dataset [1] is available[3]. We are using the Google Cartographer offline node, which processes a ROS bag dataset as fast as the CPU can handle (around 4-5x realtime), making it ideal for benchmarking the impact of performing frontier detection on SLAM performance.

In order to minimize the impact on SLAM performance, the frontier detection algorithm is running in a separate thread. The algorithm tries to process all submap updates, while it can adaptively skip non-final submap updates in case the processing speed of frontier detection falls behind SLAM. For this reason, the wall clock frequency of incremental frontier updates has been measured. The results are given in Table 1.

A. Additional notes on implementation of the frontier detection algorithm

An optimization that has been included in our implementation is performing additional stabbing query tests against a fixed number of temporally close i.e. sequential submaps using unoptimized submap poses into the local frontier edge detection condition in [Algorithm 1 line 8] (for example, against 4 previous submaps). This would have the result of permanently “baking in” a negative test result against these submaps,

[1]: https://github.com/larics/cartographer_frontier_detection
[2]: https://github.com/larics/cartographer
[3]: https://goo.gl/62zEUy

| Dataset | cartographer_paper_deutches_museum.bag | duration: 1912s |
|---------|---------------------------------------|-----------------|
| Testing setup details | Intel i7 6800K @ 3600 MHz, Ubuntu 18.04, ROS Melodic | |
| POSGRAPH.constraint_builder.sampling_ratio = 0.1 | |
| MAP>BUILDER.num_background_threads = 10 | |
| Cartographer offline node, exit before final optimization | |
| No RViz visualization | |
| Wall clock SLAM processing time (not including final optimization) | |
| with frontier detection: 434s (4.4x realtime on average) | |
| without frontier detection: 360s (5.3x realtime on average) | |
| Frequency of frontier updates | |
| Average: 78 Hz (13 ms) Std. deviation: 22 ms | |
| SLAM events | |
| Total submap update events: 37816 | |
| Skipped submap update events: 3900/37816 | |
| Pose graph optimization events: 355 | |

Our implementation of frontier detection, available online on Github[4] has been tested on an Intel i7 6800K CPU running Ubuntu 18.04. A demo video of processing Google’s Deutsches museum dataset [1] is available[4]. We are using the Google Cartographer offline node, which processes a ROS bag dataset as fast as the CPU can handle (around 4-5x realtime), making it ideal for benchmarking the impact of performing frontier detection on SLAM performance.

In order to minimize the impact on SLAM performance, the frontier detection algorithm is running in a separate thread. The algorithm tries to process all submap updates, while it can adaptively skip non-final submap updates in case the processing speed of frontier detection falls behind SLAM. For this reason, the wall clock frequency of incremental frontier updates has been measured. The results are given in Table 1.
since these points would permanently be erased from the local frontiers. A benefit of discarding points early from local frontiers is speeding up recomputation of the global frontier in Algorithm 2 since there will now be fewer candidates in the local frontier which have to be re-projected and re-tested. Also, it is expected that the localization drift between sequential submaps is small, and that pose graph optimization will not produce a significant relative displacement between sequential submaps. Using the unoptimized poses may actually prevent detecting a false frontier resulting from slight misalignment of sequential submaps introduced by pose graph optimization when closing loops.

We have employed a few cosmetic improvements which result in (subjectively) aesthetically better frontiers. The first improvement is treating cells with a very uncertain “free” probability (i.e., \( S_{k,l}^{i,j} \in [0.5 - \varepsilon, 0.5] \), where \( \varepsilon := 0.04 \) is an arbitrarily chosen small value) as “unobserved” cells. This prevents detecting a false frontier around single false long-distance laser readings, which cause insertion of a false ray of “free” cells into the submap. This also has the beneficial effect of driving the robot exploration system to get a “better look” at areas with uncertain free cells, since they are considered unexplored. The second change we have made is simple smoothing of local frontiers by adjusting the definition of a local frontier to be the center of an unobserved cell with \( \geq 2 \) free adjacent cells and \( \geq 2 \) unobserved adjacent cells, where adjacent cells are cells in the Moore 8-neighborhood.

VII. CONCLUSION AND FUTURE WORK

We have described, implemented and tested an efficient frontier detection algorithm that is specialized for 2D graph SLAM based on occupancy grid submaps. Our algorithm is efficiently constrained to the area of active submaps, yet robust to loop closure by recomputing the global frontier after pose graph optimization. Importantly, the time complexity of recomputing the global frontier is a function of the frontier perimeter, and not of map area.

Future work: Some cited state of the art (e.g., [10]) performs grouping of continuous frontier points into segments, which is useful for selecting navigation objectives of exploration tasks. Another kind of post-processing of the detected frontier which might be of interest is reachability analysis: detecting frontier points unreachable by the robot which do not make sense as navigation objectives, such as frontier points behind glass, closed doors or behind walls (e.g. from false laser readings, or slight misadjustment of walls in different submaps). Nonetheless, a performant algorithm for frontier detection is the basis for any such further improvements.

REFERENCES

[1] W. Hess, D. Kohler, H. Rapp, and D. Andor, “Real-time loop closure in 2D lidar slam,” in Robotics and Automation (ICRA), 2016 IEEE International Conference on. IEEE, 2016, pp. 1271–1278.
[2] B. Yamauchi, “A frontier-based approach for autonomous exploration,” in Computational Intelligence in Robotics and Automation, 1997. CIRA’97. Proceedings., 1997 IEEE International Symposium on. IEEE, 1997, pp. 146–151.
[3] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, “Collaborative multi-robot exploration,” in Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065). IEEE. [Online]. Available: https://doi.org/10.1109/robot.2000.844100
[4] L. Freda and G. Oriolo, “Frontier-based probabilistic strategies for sensor-based exploration,” in Proceedings of the 2005 IEEE International Conference on Robotics and Automation. IEEE. [Online]. Available: https://doi.org/10.1109/robot.2005.1570713
[5] J. Wettach and K. Berns, “Dynamic frontier based exploration with a mobile indoor robot,” in ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics), June 2010, pp. 1–8.
[6] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, “Coordinated multi-robot exploration,” IEEE Transactions on Robotics, vol. 21, no. 3, pp. 376–386, jun 2005. [Online]. Available: https://doi.org/10.1109/tro.2004.839233
[7] J. Faigl and M. Kulich, “On determination of goal candidates in frontier-based multi-robot exploration,” in 2013 European Conference on Mobile Robots. IEEE, sep 2013. [Online]. Available: https://doi.org/10.1109/ecmr.2013.6698844
[8] R. Shade and P. Newman, “Choosing where to go: Complete 3d exploration with stereo,” in 2011 IEEE International Conference on Robotics and Automation. IEEE, May 2011. [Online]. Available: https://doi.org/10.1109/ICRA.2011.5980121
[9] P. Qin, A. Alempijevic, G. Paul, and D. Liu, “Expanding wavefront frontier detection: An approach for efficiently detecting frontier cells,” in Australasian Conference on Robotics and Automation. Australasian Robotics and Automation Association, 2014.
[10] M. Keidar and G. A. Kaminka, “Efficient frontier detection for robot exploration,” The International Journal of Robotics Research, vol. 33, no. 2, pp. 215–236, Oct 2013. [Online]. Available: https://doi.org/10.1177/0278364913494911
[11] 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, 2007.
[12] P. Senarathne, D. Wang, Z. Wang, and Q. Chen, “Efficient frontier detection and management for robot exploration,” in Cyber Technology in Automation, Control and Intelligent Systems (CYBER), 2013 IEEE 3rd Annual International Conference on. IEEE, 2013, pp. 114–119.
[13] H. Umari and S. Mukhopadhyay, “Autonomous robotic exploration based on multiple rapidly-exploring randomized trees,” in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, Sep 2017. [Online]. Available: https://doi.org/10.1109/IROS.2017.802319
[14] J. J. Leonard and H. F. Durrant-Whyte, “Simultaneous map building and localization for an autonomous mobile robot,” in Intelligent Robots and Systems’ 91.Intelligence for Mechanical Systems, Proceedings IROS’91. IEEE/RSJ International Workshop on. Ieee, 1991, pp. 1442–1447.
[15] S. Agarwal, K. Mierle, and Others, “Ceres solver,” http://ceres-solver.org.
[16] S. Williams, G. Dissanayake, and H. Durrant-Whyte, “An efficient approach to the simultaneous localisation and mapping problem,” in Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292). IEEE. [Online]. Available: https://doi.org/10.1109/robot.2002.1013394
[17] J. Strom and E. Olson, “Occupancy grid rastorization in large environments for teams of robots,” in Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on. IEEE, 2011, pp. 4271–4276.
[18] M. Bosse and R. Zlot, “Map matching and data association for large-scale two-dimensional laser scan-based slam,” The International Journal of Robotics Research, vol. 27, no. 6, pp. 667–691, 2008.
[19] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Lintkaitis, and R. Vincent, “Efficient sparse pose adjustment for 2d mapping,” in Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on. IEEE, 2010, pp. 22–29.
[20] G. Guennebaud, B. Jacob et al., “Eigen v3,” http://eigen.tuxfamily.org, 2010.
[21] “Boost R-tree spatial index,” https://www.boost.org/doc/libs/1_68_0/libs/geometry/doc/html/geometry/spatial_indexes/introduction.html