Autoexplorer: Autonomous Exploration of Unknown Environments using Fast Frontier-Region Detection and Parallel Path Planning

Kyung Min Han and Young J. Kim

***Abstract*— We propose a fully autonomous system for mobile robot exploration in unknown environments. Our system employs a novel frontier detection algorithm based on the fast front propagation (FFP) technique and uses parallel path planning to reach the detected front regions. Given an occupancy grid map in 2D, possibly updated online, our algorithm can find all the frontier points that can allow mobile robots to visit unexplored regions to maximize the exploratory coverage. Our FFP method is six∼seven times faster than the state-of-the-art wavefront frontier detection algorithm in terms of finding frontier points without compromising the detection accuracy. The speedup can be further accelerated by simplifying the map without degrading the detection accuracy. To expedite locating the optimal frontier point, We also eliminate spurious points by the obstacle filter and the novel frontier region (FR) filter. In addition, we parallelize the global planning phase using the branch-and-bound A*, where the search space of each thread is confined by its best knowledge discovered during the parallel search. As a result, our parallel path-planning algorithm operating on 20 threads is about 32 times faster than the vanilla exploration system that operates on a single thread. Our method is validated through extensive experiments, including autonomous robot exploration in both synthetic and real-world scenarios. In the real-world experiment, we show that an autonomous navigation system using a human-sized mobile manipulator robot equipped with a low-end embedded processor that fully integrates our FFP and parallel path-planning algorithms.**

I. Introduction

Exploring unknown regions is a primitive research motivation for embodied agent navigation problems. Indeed, seeking unknown space is a common objective to drive many types of mobile robot platforms that operate in diverse environments, ranging from exploration in extreme environments like Mars exploration rovers [1] to consumer-level products such as home cleaning bots [2] that affects our daily life.

Specifically, map building is one of the most fundamental mobile robot tasks where the exploring agent should be aware of visited places to discover a new destination that is likely to provide abundant information about the navigating environment. As such, it is evident that locating new places and prioritizing their visiting orders based on reasonable criteria would improve the agent's navigation performance in terms of its exploration efficiency.

While numerous studies have been conducted with regards to the efficient navigation strategies, the standard approach is detecting *frontier region* in the occupancy grid map in 2d, which is defined as the boundary of an unexplored free region, followed by global motion planning to generate an optimal path to reach the detected frontier regions. Needless to say, exploration accuracy and efficiency are two essential features for evaluating the procedure of frontier point detection.

In this paper, we propose a simple and fast approach for autonomous exploration in unknown environments while maintaining its exploration accuracy sufficient for practical applications in mobile robot navigation problems. Our method exploits the idea of fast front propagation (FFP) algorithm [3][4] which quickly propagates front grid-cells to identify the boundaries between the known and unknown regions in a given 2d grid map. In FFP, the scanning (i.e., propagation) procedure is dependent on the number of map cells, making the algorithm runs in linear time in terms of the number of map cells in the map. As a result, our method is much faster than the wavefront detection (WFD) [5] algorithm. Moreover, we have achieved further acceleration by hierarchically exploiting the regular structure of the 2d grid map, allowing the frontier detection algorithm insensitive to the online map update frequency. Thanks to this speed-up, we can globally locate frontier regions by processing the entire map images even if the size of the map grows up to cover a very large scale environment –e.g. *Deutsche museum* dataset [6] covering an area consisting of $5.4K \times 3.3K$ cells at $5cm$ resolution.

In addition, we introduce two novel filters, namely frontier region (FR) filter and obstacle filter, to eliminate spurious frontier points detected by the FFP algorithm to minimize visiting redundant sites. Afterward, the filtered points are efficiently handled by the global, parallel path-planner, which distributes the path plan tasks to multiple processors based on the branch-and-bound technique to early-terminate unnecessary search works. We observed that employing both the two filters and parallel path-planning improved the planning time up to 32 times compared to a serialized planner.

In summary, our main contributions in this paper are:

- We propose a novel linear-time algorithm for frontier detection that runs faster than the WFD by $6 \sim 7$ times without degrading the detection accuracy. This performance can be further accelerated by two folds by simplifying the input maps.
- We propose two novel filters (FR filter and obstacle filter) that can eliminate spurious frontier points to accelerate the path planning step.
- Our global path-planner is effectively parallelizable using the branch and bound, showing scalable performance in terms of the number of participating CPU threads.
- We show a practical application of our algorithm solving a real-world 2d space exploration problem using a human-sized mobile manipulator running entirely on the embedded multi-core CPUs.

The authors are with the department of computer science and engineering at Ewha womans university in Korea $\{hankm|kimy\}@ewha.ac.kr$

II. Related Work

The idea of locating unexplored regions to visit is also known as *frontier point detection* problem, which was first introduced by Yamauchi [7]. His work defines the boundary regions between free space and unexplored space as *frontiers*. Later on, Yamuchi's work impacted many following research works.

For instance, Burgard et al. [8] expands the idea of covering frontier cells with a multi-robot team. This method exploits the mobile robots' current position to assign an appropriate target goal to each robot. As a result, the robots collaborate on exploring unknown spaces more efficiently than operating a single robot for the same task.

[5] revisits the frontier detection problem by introducing wavefront frontier detector (WFD). WFD uses two bread-first search (BFS) schemes in order to locate frontier boundaries. Beginning from the robot's position, their algorithm employs a BFS scan over the unoccupied cells until a frontier cell is detected. Then, a new BFS is executed to find the neighboring frontier point regions starting from the detected frontier point. In Keidar's work, WFD is expanded to the fast frontier detector (FFD) to achieve further speed-up. However, since the method relies on the incoming scanning, it resorts to a database structure to globally manage previously detected frontier points. [9] proposed two variants of WFD called EWFD and NaiveAA, which could be categorized into another branch of WFD.

Locating frontier regions in the entire occupancy grid map could become a time-consuming process when covering a large space. A workaround solution is limiting the examining space to the local region that the current scanning process reaches. [10] introduced an efficient updating strategy that copes with only the newly modified map cells. RRT exploration [11] ameliorates the efficiency issue employing rapidly exploring random tree (RRT), yet [12] reports that RRT exploration has a limited performance in narrow corridor cases.

Osulic's dense frontier detector (DFD) [12] offers a submap based approach built on top of SLAM cartographer [6]. DFD exploits the properties of pose graph SLAM, such as considering active local maps only to speed up the frontier detection procedure. One drawback of this method is a lack of compatibility with other types of SLAM systems since DFD is tightly coupled with SLAM cartographer. [13] is a spinning-off research of DFD since Sun's work is implemented on top of SLAM cartographer similar to DFD. This method introduces a dilation step prior to detecting the frontier region to discard unreachable frontier points.

In the context of parallel planning, there have been many types of research to parallelize A* search methods [14]. For instance, simple parallel A* search (SPA) [15] uses a single open list shared over whole threads. SPA is relatively simple to implement, while the shared memory causes extra timing overhead on the thread synchronization. The randomized methods [16], [17] are typical decentralizing strategies for alleviating the synchronization overhead, yet could suffer from a large amount of search overhead due to redundant expansion of the same node by different threads.[18] exploits GPU computation power to parallelize priority queues.However, relying on GPU power could be infeasible for low power light-weighted embedded systems. In contrast, our parallel search method is simple to work on grid maps and cost maps and shows scalable performance in terms of participated parallel processors.

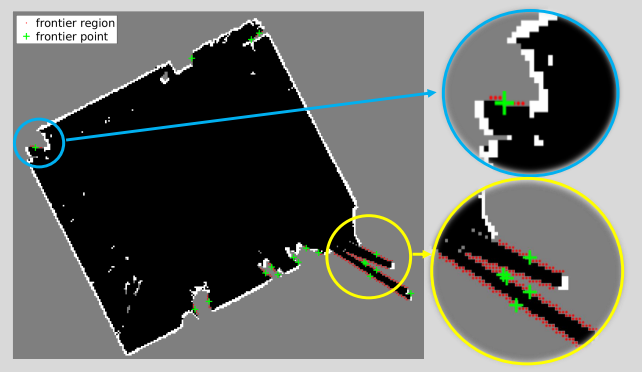


Fig. 1: The map image consists of black, gray and white pixels that represent free, unknown, and occupied map cells, respectively. The red points and green crosses are superimposed on top of the map to show frontier regions and their frontier points, respectively.

III. Preliminaries

A. Problem Formulation

A 2d grid map $\mathcal{M}$ consists of three different types of cell classes: (1) occupied cells $\mathcal{M}_o$ corresponding to obstacles, (2) unoccupied or free cells $\mathcal{M}_f$ corresponding to the space where a robot can freely navigate, and (3) unknown or unexplored cells $\mathcal{M}_u$. Therefore, the occupancy of a cell x, $\mathcal{M}(x)$, is labeled as

$$\mathcal{M}(x) = \begin{cases} \textit{OCCUPIED} & \text{if } \mathcal{M}(x) \in \mathcal{M}_o \\ \textit{UNKNOWN} & \text{if } \mathcal{M}(x) \in \mathcal{M}_u \\ \textit{FREE} & \text{if } \mathcal{M}(x) \in \mathcal{M}_f. \end{cases} \tag{1}$$

A set of unknown cells neighboring either free or occupied cells is defined as a boundary region $\mathcal{B}$. A set of unknown cells neighboring free cells is defined as a frontier region $\mathcal{F}$ which is a subset of $\mathcal{B}$; see Figure 1.

Information gain (IG) stands for the size of frontier region $|\mathcal{F}|$, where $|\cdot|$ refers to the cardinality of a set. c_i is the geometric centroid of i^{th} frontier region $\mathcal{F}_i$. Subsequently, i^{th} frontier point f_i is the closest point to $\mathrm{c_i}$ among all the points belonging to $\mathcal{F}_i$. That is,

$$f_i = \{x \mid \underset{x}{\operatorname{argmin}}\ D(\mathrm{c}_i, x) \text{ and } x \in \mathcal{F}_i\} \tag{2}$$

where $D()$ is the Euclidean distance operator. Lastly, f^* refers to the optimal frontier point that has the shortest path among all frontier candidate points; f^* is set to a goal point for global path planning.

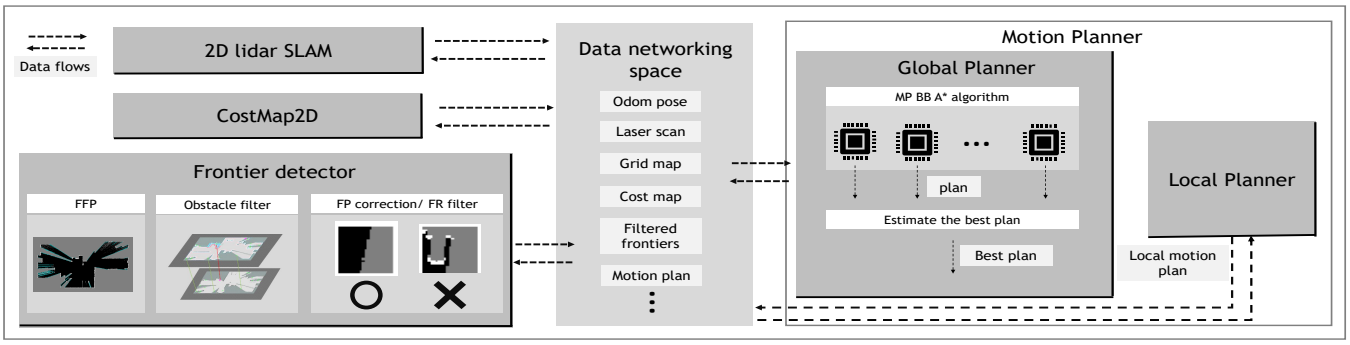


Fig. 2: Autoexplorer overview. The frontier detector finds a set of candidate frontier points $\{f_i\}$, which are subsequently fed into the global planner to identify the optimal frontier point f^*.

B. Autoexplorer System Overview

In this work, we assume that a mobile robot senses its surroundings with its range sensor during an exploration task, thereby constructing 2d occupancy grid maps with a SLAM module. The frontier detection algorithm takes a new map whenever it is updated by the map generator. Then, two filtering processes acting on the cost-map (obstacle filter) and grid-map (FR filter) eliminate spurious candidate frontier points. The resulting frontier points are efficiently handled by parallel global planner based on the A* search with branch and bound in order to select the optimal frontier point f^* and generates the global path plan using f^* as a goal point. Finally, the robot moves toward the goal under the guidance of both the global planner and a local planner using DWA [19] avoiding obstacles. Figure 2 illustrates the overall architecture of our exploration system.

IV. The Frontier Detection Algorithms

A. Frontier Region Detection using Fast Front Propagation

The fast marching level-set method [3] is a numerical technique to solve the boundary value problems in diverse fields of studies. The boundary value problem can be understood as an inside/outside classification problem or identifying the boundaries of particular regions, for instance, such as edges in a 2D image. Notably, Kim et al. [4] used this idea to find the boundary surface of swept volumes of complex polyhedrons. In this approach, the method propagates a front inward from arbitrarily far seeding points to arrive at the outer boundaries of the arrangement of open surfaces.

Our work was inspired by this work to locate frontier regions of given 2D maps. In addition, we further speed up the whole process by taking a multi-resolution approach for the map.

Algorithm 1 summarizes the pseudo-codes of our fast front propagation (FFP). The method consists of two consecutive procedures: 1) marching front (MARCHFRONT) and 2) extracting frontiers (EXTRACTFRONTIERS). Marching front scans over the entire map image to locate and store boundary cells. Extracting frontiers revisits the stored boundary cells to extract frontier regions.

At the beginning of the algorithm, a seed point is appended to a queue $\mathbf{Q}$ (Line 9). In the main loop, a cell $q \in \mathbf{Q}$ and its neighboring cells are inspected whether they are boundary cells (i.e., $q \in \mathcal{B}$) or not (Line 14-24). For example, if

Algorithm 1: Fast front propagation (FFP) algorithm

```
1  Function ExtractFrontierRegion ( input map M) :
2      Empty the scan list P
3      Initialize the lattice vector L as FAR
4      Empty the front queue Q
5      Initialize the frontier region list F
6      MarchFront()
7      ExtractFrontiers()
8  Function MarchFront():
9      Add a seed index s to Q
10     while Q is nonempty do
11         Extract a trial point index q from Q
12         L(q) ← KNOWN
13         for each neighboring point index n around q
               do
14             if L(n) is not KNOWN then
15                 if M(n) is UNKNOWN then
16                     if L(n) is FAR then
17                         Add n to Q
18                         L(n) ← TRIAL
19                     end
20                 end
21                 else
22                     Add q to P
23                 end
24             end
25         end
26     end
27 Function ExtractFrontiers():
28     for each point index p in P do
29         if For all neighboring indexes n around p,
30         P(n) are not OCCUPIED then
31             Add p to F
32         end
33     end
34     return F
```

any neighboring cell of q is not an *UNKNOWN* cell, then q is at the boundary region because q is an *UNKNOWN* cell. Subsequently, the boundary cells are stored in a linear list $\mathbf{P}$ (Line 23) during the main loop process since this cell potentially belongs to the frontier region. Moreover, $\mathbf{L}$ stores the information on whether the examined cell has been previously visited or not.

One can naively initialize the seed point to be an unexplored point, arbitrarily far from the current robot position (e.g. the origin of the entire map image), whereas a better idea estimates the nearest, unexplored point from the explored map using the extent of the map. Moreover, it is unnecessary to propagate through the entire grid map, if the size of the map is given. For example, Cartographer [6] builds a map with the currently explored region while Gmapping [20] uses a fixed map size (4K $\times$ 4K). When an input map is given, we pad the border of the map with UNKNOWN labels. For example, a $N \times N$ sized map expands to $(N + p) \times (N + p)$, where p is the size of the padding. Subsequently, FFP chooses the (0,0) coordinate of the expanded map as the seed point.

EXTRACTFRONTIERS revisits all the cells stored in $\mathbf{P}$ to check whether any of its neighboring cells are *FREE* (Line 29) or not. If all neighboring points are not *OCCUPIED*, then the function has found a frontier cell. Thus, this point needs to be collected (Line 30).

Once the frontier region $\mathcal{F}$ is identified, computing the frontier point f is performed as follows. We cluster frontier regions by computing a connected component for each frontier region, followed by computing their centroids and the corresponding frontier point.

B. Complexity Analysis

The marching front procedure handles three different data structures, two linear lists $\mathbf{L}$ and $\mathbf{P}$, and a queue $\mathbf{Q}$ whose sizes are bounded by the number of map cells. Moreover, the number of cells to process in extracting frontiers is linear to the size of $\mathbf{P}$ only. Therefore, our method has $O(n)$ complexity where n is the number of map cells. Note that the WFD runs a BFS inside another BFS loop, making the $O(n^2)$ complexity in the worst-case scenario.

C. Multi-resolutional Maps

Section IV-B discusses that the performance of our method is dependent on the number of map cells. Thus, reducing the map size to coarser levels would considerably speed up the whole process. We convert the SLAM maps to 2D map images and successively down-sample them to build a multi-resolution hierarchy [21] of map grids. [5] briefly mentions the possibility of using coarse level approximation without further analysis while we have carried out a number of experiments in Section VI-A to concretely provide the effect of down-sampled map-images in terms of both run-time and the number of frontier point detection.

D. Eliminating Spurious Frontier Points

Processing a map with the FFP algorithm may generate a high number of redundant frontier point candidates. These candidates often include unreachable cells such as a cell cluster surrounded by obstacles or false detection caused by incorrect laser sensor scanning. Motion planning to all of such cells would cause severe inefficiency in the exploration process.

To eliminate the spurious frontier candidates, we first employed the *obstacle filter*, where we reject points surrounded by obstacles. To do this, we compute the cost-map score [22] of obstacle-presence probability for a small patch around the frontier point candidate. Then, we reject the points whose cost is higher than a predetermined threshold. Secondly, we proceed to the *FR filtering*, which extracts a small patch around the candidate cell on the occupancy grid map, followed by computing the *boundary measure* ρ of the patch $\mathcal{P}$:

$$\rho = 1 - 2\left|\frac{|\mathcal{P}_u|}{|\mathcal{P}|} - \frac{1}{2}\right| \tag{3}$$

where $|\mathcal{P}|$ and $|\mathcal{P}_u|$ refer to the total number of cells in the patch and the number of unknown cells in the patch, respectively. A candidate point f_c with $\rho \approx 1$ is likely to be on the boundary of the unknown regions (the green point in Figure 3), which potentially yields a high probability of discovering abundant map information on visiting it.

It is possible that frontier point candidates computed from coarse-level maps are often found at inaccurate locations when the points are transformed to original map coordinate. Such location errors need to be corrected prior to computing their boundary measures. To this end, we successively push the frontier point's location outward in spiral direction until the point neighbors at least one FREE cell.

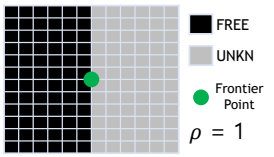


Fig. 3: An ideal frontier region patch where the number of FREE cells and UNKNOWN cells are the same, leading to $\rho = 1$.

V. Parallel Path Planning

It is not unusual to locate several frontier points at the end of the frontier detection process, especially at the beginning stage of exploration. One exploration strategy is visiting the closest frontier point f^* among the available frontier points. Such a strategy involves computing the path length to each of the individual frontier point. This path-length computation would be time consuming if the number of frontier points is high and parallel computation would reduce the overall time.

A. A* Search

A* algorithm [23] is an efficient graph-search method to find an optimal path from the starting to the goal node. Given a starting node, A* traverses the graph by choosing the node having the minimal cost function:

$$\mathsf{F}(n) = \mathsf{G}(n) + \mathsf{H}(n), \tag{4}$$

where $\mathsf{G}(n)$ computes the cost from the starting node n_s to the current node n, and $\mathsf{H}(n)$ is a heuristic function to estimate the cost from n to the goal node n_g.

B. Parallel A* Search with Branch and Bound

An A* algorithm should satisfy the following theorems to satisfy the *completeness* [24].

Theorem 1 (Consistent heuristic): A heuristic function $\mathsf{H}()$ is consistent if $\mathsf{H}(n) <= \mathsf{C}(n, n') + \mathsf{H}(n')$ where $\mathsf{C}(n, n')$ is the cost of moving from the node n to n' .

Corollary 1 (Monotonic cost): In consistent A* search, $\mathsf{F}(n)$ is a monotonic increasing function. –i.e.) $\forall n, n' \in \mathcal{M}$, $\mathsf{F}(n') \geq \mathsf{F}(n)$.

Now we assume the following to preserve the underlying consistency property over the entire map, and the grid map consistency is guaranteed.

Assumption 1: A path is calculated on the occupancy grid map, where each cell on the map is classified into three distinct labels (FREE, OCCUPIED, UNKNOWN)

Theorem 2 (Grid map consistency): A* search with Euclidean heuristic on the occupancy grid map is consistent.

Proof: If theorem 2 is false, then $\exists n, n'$, $\mathsf{H}(n) > \mathsf{C}(n, n') + \mathsf{H}(n')$ must be true. Let $\mathsf{H}(n) := \mathsf{D}(n, n_g)$, which refers to Euclidean distance between the current node and the goal node. Rewriting the equation, we need to see if $\mathsf{D}(n, n_g) > \mathsf{C}(n, n') + \mathsf{D}(n', n_g)$ is valid. Accordingly, $\mathsf{D}(n, n_g) > \mathsf{D}(n, n') + \mathsf{D}(n', n_g)$ must hold because $\mathsf{C}(n, n') >= \mathsf{D}(n, n')$. However, $\mathsf{D}(n, n_g) - \mathsf{D}(n', n_g)$ is equivalent to $\mathsf{D}(n, n')$, making $\mathsf{D}(n, n') > \mathsf{D}(n, n')$, which is an invalid statement. ■

Our parallel A* search algorithm using branch and bound combined with Corollary 1 and Theorem 2 works as follows:

1) **Branch**: we launch many CPU threads T_i's to perform A* search concurrently by computing $\mathsf{F_i}$ in Eq. 4.
2) **Bound**: whenever the thread T_i encounters $\epsilon < \mathsf{F_i}$ during A* search, T_i is terminated. Here, ϵ is an upper bound of all $\mathsf{F_i}$ values and is initialized to a large value.
3) If T_i successfully finishes the search and $\mathsf{F_i} < \epsilon$, ϵ is updated to $\mathsf{F_i}$ ($\epsilon := \mathsf{F_i}$).

Algorithm 2 summarizes the pseudo-code of our parallel A* search algorithm. Similar to the standard A* search, our algorithm initializes two lists of the open and the closed nodes and keep track of them (Line 4-5). Then, each thread starts performing A* search based on the bound condition (Line 6). The function exploits the bound condition to terminate the thread (Line 9). Lastly, the algorithm updates the bounding parameter ϵ if the thread has found a better path (Line 15-17). The algorithm finally selects the optimal frontier point f^* which has the shortest path length. Subsequently, the trajectory to f^* is used as the global plan to guide the robot.

Algorithm 2: Parallel A* with Branch and Bound

```
1  Function MULTI-PROCESSING(MP) BRANCH AND BOUND(BB) A* SEARCH ( a set of frontier points {f} ) :
2      ϵ := ∞
3      launch T_i and do in parallel
4      Empty the priority buffer q_o
5      Empty the buffer q_c
6      while f_j ∈ {f} is not processed do
7          Dequeue open cells n from q_o
8          Update cell n's cost F(n)
9          if min(F(n)) > ϵ then
10             Abandon f_j and relanunch the thread T_i
11         end
12         Enqueue new discovered cells into q_o
13         Enqueue n into q_c
14     end
15     if F(f_j) < ϵ then
16         ϵ ← F(f_j)
17     end
18 end
19 return the optimum f* s.t. F(f*) := ϵ
20
```

VI. Experiments and Results

In this work, we tested our system on two different types of simulators: 1) Tesse-Unity simulator [25] and 2) AWS Gazebo simulator [26]. The former offers realistic rendering of scenes while the latter provides a more practical robotic model with rich sensory data. We observed that our system successfully completed all environments available in both simulators. These results are qualitatively reported in Figure 4.

A. Performance of the Frontier Detection Algorithm

To quantitatively measure the performance of our frontier detection algorithm (FFP), we collected a sequence of map images while driving an agent in the 2nd Tesse-scene. The size of the covered area gradually increases over time as the robot moves around. We measured the performances of WFD vs. FFP in terms of both run-time and the number of detected frontier points. Figure 5(a) shows that FFP is generally faster than WFD while the number of detected points of the two methods are almost identical, as shown in Figure 5(b). The speed-up is much more prominent when the down-sampling procedure is applied. FFP with a double down-sampled map image is about 14 times faster than the original FFP and 87 times faster than WFD.

This run-time experiment highlights the benefits of down-sampled map images, but it also raises another question: to what extent does the down-sampling procedure affect the number of detected frontier points? To answer this question, additional experiments were undertaken and reported in Figure 5(c). This figure shows how the down-sampled map images affects the number of detected frontier points. The figure implies that thresholding the cluster size of a frontier region with IG > 6 is almost equivalent to the performance of applying single down-sampling - see Section III-A for the definition of IG. That is, if we consider only the reasonable size of frontier region clusters, the coarse level analysis still finds all the necessary frontier points. Here, IG > 6 refers to a cluster with a cell size greater than 6, meaning a region

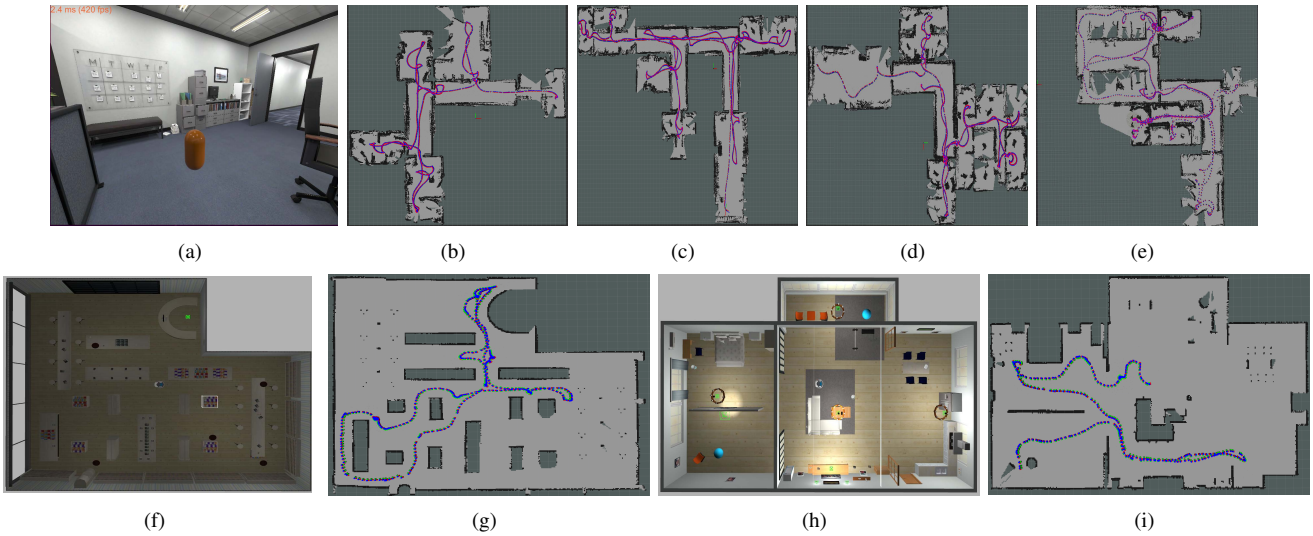


Fig. 4: (a) An example snap shot taken from the Tesse-envs. (b)∼(e) are the exploration results of four different Tesse-envs. The red dots represent agent's trajectories. (f) and (h) are AWS-book-store and AWS-small-house environments. (g) and (i) are the corresponding exploration results, respectively. The blue dots represent agents' trajectories.

with 0.3m since the map resolution was set as 0.05m. Note that 0.3m is the size of an ordinary home cleaning robot such as Turtlebot, and IG = 12 is about the size of a warehouse robot –e.g., Fetch robot. Thus, with single down-sampled map images, our algorithm can find all frontier regions that a Turtlebot-sized robot.

B. Ablation Study on the Parallel Path Planning

It is significant to analyze how the ideas of filtering and parallel A* developed in IV-D and V impacts the final efficiency of our method. To this end, beginning from the vanilla version[1] of our frontier detector, we measured the performance of our system by progressively adding improvements: 1) filtering, 2) parallel A* without branch and bound (MP naive), and 3) parallel A* with branch and bound (MPBB). The experiment result shown in Figure 6(a) highlights the significance of each improvement technique's contribution to the proposed system. For example, we observed that the MPBB approach was approximately 4.2 times faster than MP naive. When our system equipped with filtering and branch-and-bound utilizes 20 threads, we have observed about 32 times speed up compared to the vanilla version of the system. Figure 6(b) shows the scalability of our parallel search algorithm in terms of the number of participating threads.

C. Real World Experiments

1) Large-scale benchmark: We tested our system on *Deutsche museum* bag-file [6] to see the performance of our method on pose graph optimization type SLAM. This dataset explores a very large scale space that increases up to 18M cells, including many potential frontier regions. We observed that our system successfully completed the dataset, and so did the dense frontier detector (DFD).

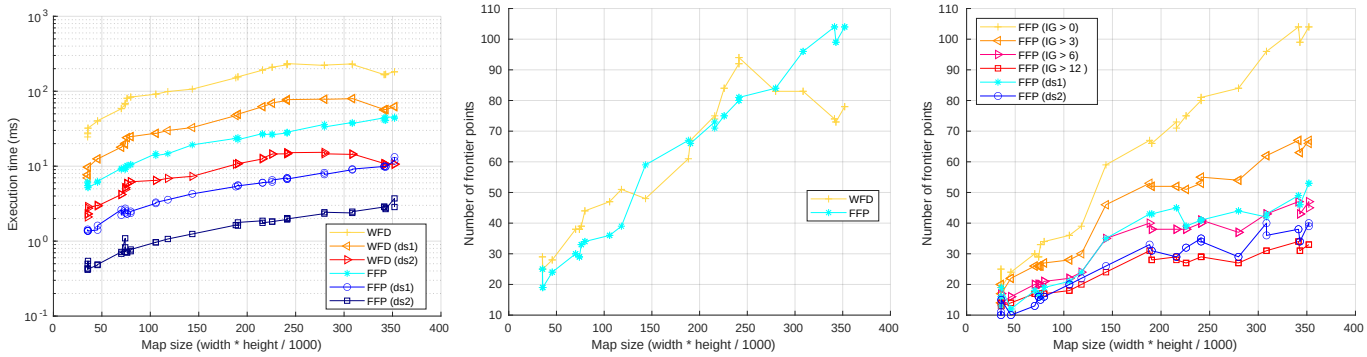
[1]Here, the vanilla system refers to a serialized global planner operated with the whole FFP outputs without any filtering process

Unfortunately, however, it was not straightforward to compare the performance of our method against the DFD algorithm for the following two reasons. First, DFD computes local frontier regions on every laser scan message, while global frontier contours are computed only when pose graph optimization is performed. Conversely, our method requires subscribing to the global map messages generated from the SLAM system to find global frontier points. This underlying difference hinders us from making the timing comparison between the two methods. Second, DFD does not offer a selection procedure to find frontier points from the detected frontier contours, while the selection function is one of the essential steps to determine the final frontier points. To address this issue, we attempted to apply the same connected component labeling algorithm used in our approach to cluster the points found in the DFD method. However, unfortunately, the clustering procedure on DFD ends up with many fragmented small components that are supposed to be grouped together while our method successfully finds meaningful clusters.

To the best of our knowledge, this fragmentation problem is associated with a sub-map-based frontier detection strategy. That is, these methods narrow down the scanning area to active sub-maps only, guaranteeing fast running time at the cost of imperfect alignment when the frontier regions are transformed and merged into the global coordinate frame.

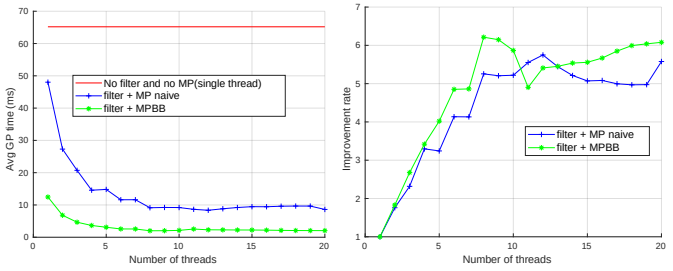
Therefore, we found that a quantitative analysis of the two methods is an infeasible task. Instead, we show Figure 8 to present that the performance of our method is qualitatively comparable to DFD in terms of finding frontier regions.

2) Experiments with a real robot: Lastly, we tested our system on the Fetch robot's onboard CPU (Intel i5 Haswell) and memory (16GB). In this experiment, the Fetch robot was operated in fully autonomous mode relying on our SW framework shown in Figure 2. We deployed our Fetch robot on the entire floor of Ewha-SK Telecom Center. Figure 9



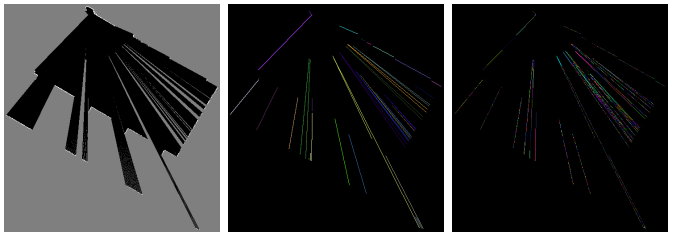
(a) (b) (c)

Fig. 5: (a) Our FFP method is about six times faster than WFD, which can be further accelerated even by down-sampling map sizes. Here, FFP(ds1) refers to applying FFP on a single down-sampled map-image. Note that FFP on a double down-sampled map-image was about 14 times faster than FFP on the original map size and 87 times faster than WFD on the original map size. Yaxis of this picture is the log-scaled execution time. While FFP is faster than WFD, the two methods scored a similar performance in terms of the number of detected frontier points as shown in (b). (c) shows the effect of down-sampling map-images on the number of detected frontiers. The experiments were performed in AWS-small-house-world.



(a) Average global planning time (b) Improvement rate

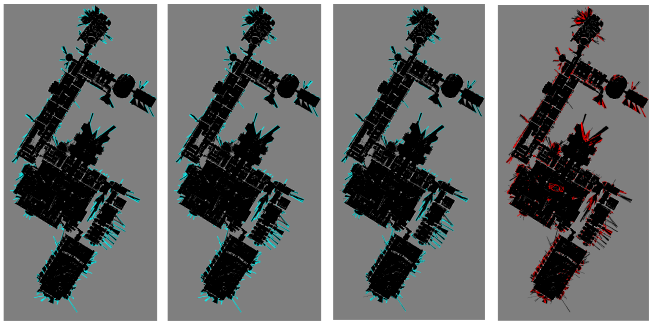
Fig. 6: (a) Our parallel planning approach is about 4.2 times faster than the parallel A* search without branch and bound. Note that our approach was about 32 times faster than the vanilla system (red line) when 20 threads were operating. (b) shows the relative run-time improvement w.r.t the single-threaded planning. All experiments were carried out in the same grid-map generated from AWS-small-house-world.



(a) (b) (c)

Fig. 7: (a) The input map image and (b) the corresponding frontier regions found by FFP followed by the clustering. (c) DFD locates a similar amount of frontier cells. However, many of them are not connected in the global coordinate frame.

shows the trajectories of the robot and the covered area. The robot took 396s for the exploration time in order to cover the entire floor when the full system architecture with filtering and



(a) FFP (b) FFP (ds1) (c) FFP (ds2) (d) DFD

Fig. 8: The detected frontier regions on a large scale map-image of the *Deutsche museum*: (a) FFP result in green lines. (b) FFP with single down-sampling (c) FFP with double down-sampling, and (d) DFD in red lines. Note that the FFP results on down-sampled map images are qualitatively comparable to other cases.

parallel A* with four CPU threads was employed. We also carried out an identical experiment with the vanilla navigation system. In this case, the robot had spent 671s to explore the entire floor.

VII. Conclusion

We have introduced a novel frontier point detection based autonomous exploration system in unknown environments. Our FFP is more efficient than WFD in the worst-case analysis. Moreover, we have shown that FFP could be further accelerated by down-sampling map sizes while maintaining sufficient accuracy in detecting essential frontier regions. Besides, our method is capable of eliminating spurious point candidates, preventing tours to unreachable goals. Lastly, our global planning module is efficiently parallelized by the branching and bound style A* algorithm. Both synthetic and real-world experiments support that our method is applicable

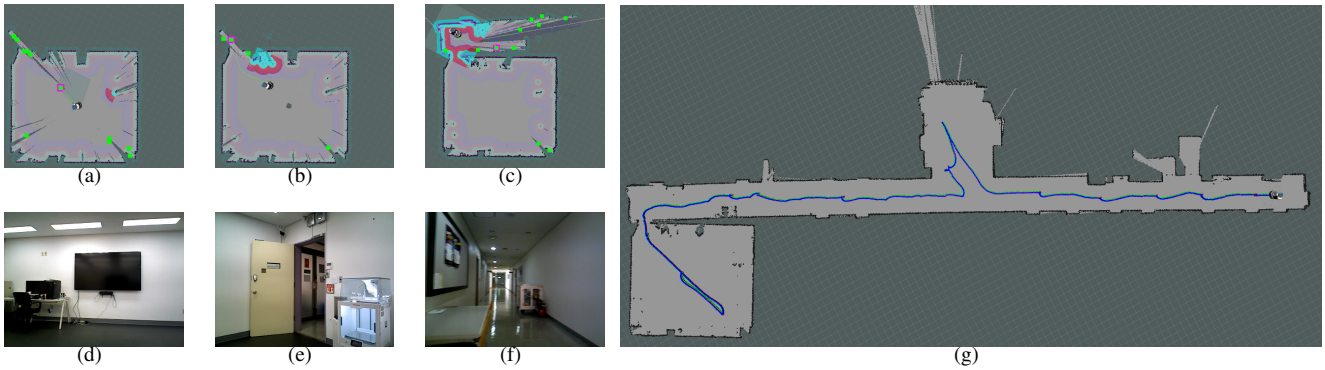


Fig. 9: Autonomous navigation in the real-world environment: (a) the Fetch robot approaches the first frontier point (b) the robot arrives at the first target, then locates the second frontier point in turn (c) the Fetch arrives at the third goal, discovering new areas subsequently. (d), (e), and (f) are the views from the Fetch robot's perspective at (a), (b), and (c), respectively. (g) is the covered area of Ewha-SK Telecom center where blue dots represents the robot's exploration trajectory

to the real-world mobile robot map building problem. In the future, we plan to tackle large-scale dynamic space covering problems using our system. To attain this goal, we need to make the system robust against moving obstacles, such as human objects.

REFERENCES

[1] L. Matthies, M. Maimone, A. Johnson, Y. Cheng, R. Willson, C. Villalpando, S. Goldberg, A. Huertas, A. Stein, and A. Angelova, "Computer vision on mars," *International Journal of Computer Vision*, vol. 75, no. 1, pp. 67 – 92, October 2007.

[2] S. Lee and S. Lee, "Embedded visual slam: Applications for low-cost consumer robots," *IEEE Robotics Automation Magazine*, vol. 20, no. 4, pp. 83–95, 2013.

[3] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996. [Online]. Available: https://www.pnas.org/content/93/4/1591

[4] Y. J. Kim, G. Varadhan, M. C. Lin, and D. Manocha, "Fast swept volume approximation of complex polyhedral models," in *SM03 Proceedings of the eighth ACM symposium on Solid modeling and Applications*, 2003, pp. 11–22.

[5] M. Keidar and G. A. Kaminka, "Efficient frontier detection for robot exploration," *International Journal of Robotics Research*, vol. 33, no. 2, pp. 215–236, 2014.

[6] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278.

[7] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*. IEEE, 1997, pp. 146–151.

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

[9] P. Quin, 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. 2014*, 2014.

[10] P. Senarathne, D. Wang, Z. Wang, and Q. Chen, "Efficient frontier detection and management for robot exploration," in *2013 IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems*, 2013, pp. 114–119.

[11] 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)*, 2017, pp. 1396–1402.

[12] J. Oršulić, D. Miklić, and Z. Kovačić, "Efficient dense frontier detection for 2-d graph slam based on occupancy grid submaps," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3569–3576, 2019.

[13] Z. Sun, B. Wu, C.-Z. Xu, S. E. Sarma, J. Yang, and H. Kong, "Frontier detection and reachability analysis for efficient 2d graph-slam based active exploration," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 2051–2058.

[14] A. Fukunaga, A. Botea, Y. Jinnai, and A. Kishimoto, "A survey of parallel A," *CoRR*, vol. abs/1708.05296, 2017. [Online]. Available: http://arxiv.org/abs/1708.05296

[15] K. B. Irani and Y. Shih, "Parallel a* and ao* algorithms: An optimality criterion and performance evaluation," in *ICPP*, 1986.

[16] V. Kumar, K. Ramesh, and V. N. Rao, "Parallel best-first search of state-space graphs: A summary of results," in *AAAI*, 1988, pp. 122–127.

[17] R. Karp and Y. Zhang, "A randomized parallel branch-and-bound procedure," ser. STOC '88. New York, NY, USA: Association for Computing Machinery, 1988, p. 290–300. [Online]. Available: https://doi.org/10.1145/62212.62240

[18] Y. Zhou and J. Zeng, "Massively parallel a* search on a gpu," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. AAAI Press, 2015, p. 1248–1254.

[19] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[20] 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.

[21] R. Szeliski, *Computer Vision - Algorithms and Applications, Second Edition*, ser. Texts in Computer Science. Springer, 2022.

[22] D. V. Lu, D. Hershberger, and W. D. Smart, "Layered costmaps for context-sensitive navigation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 709–715.

[23] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[24] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[25] D. Yadav, R. Jain, H. Agrawal, P. Chattopadhyay, T. Singh, A. Jain, S. Singh, S. Lee, and D. Batra, "Evalai: Towards better evaluation systems for AI agents," *CoRR*, vol. abs/1902.03570, 2019. [Online]. Available: http://arxiv.org/abs/1902.03570

[26] AWS-Robotics-Team, "aws-robomaker-bookstore-world and aws-robomaker-small-house," https://github.com/aws-robotics/aws-robomaker-bookstore-world, 2018.