Expanding Wavefront Frontier Detection: An Approach for Efficiently Detecting Frontier Cells

Phillip Quin, Alen Alempijevic, Gavin Paul, Dikai Liu
Center for Autonomous Systems
University of Technology Sydney, Australia
phillip.d.quin@student.uts.edu.au

Abstract

Frontier detection is a key step in many robot exploration algorithms. The more quickly frontiers can be detected, the more efficiently and rapidly exploration can be completed. This paper proposes a new frontier detection algorithm called Expanding Wavefront Frontier Detection (EWFD), which uses the frontier cells from the previous timestep as a starting point for detecting the frontiers in the current timestep. As an alternative to simply comparing against the naive frontier detection approach of evaluating all cells in a map, a new benchmark algorithm for frontier detection is also presented, called Naive Active Area frontier detection, which operates in bounded constant time. EWFD and NaiveAA are evaluated in simulations and the results compared against existing state-of-the-art frontier detection algorithms, such as Wavefront Frontier Detection and Incremental-Wavefront Frontier Detection.

1 Introduction

Since the seminal work by Yamauchi on frontier-based exploration [Yamauchi, 1997], frontiers have been used extensively as part of robot exploration to great effect with single robots [Yamauchi *et al.*, 1998; Freda and Oriolo, 2005; Digor *et al.*, 2010; Wettach and Berns, 2010; Mobarhani *et al.*, 2011] and with teams of multiple robots [Yamauchi, 1998; Faigl and Kulich, 2013; Burgard *et al.*, 2000; Reid *et al.*, 2013]. In some approaches, frontiers are treated as path planning goals, and in other approaches, frontiers are used as a component of more complex strategies, for example in [Shade and Newman, 2011] where frontiers are used as sinks in a potential-field. Detecting frontiers is therefore a key operation, and it is important that it be performed as quickly as possible so that exploration can be more efficient.

Frontiers are defined as free-space cells in an occupancy grid that have at least one unknown cell as a neighbour [Yamauchi, 1997; Freda and Oriolo, 2005; Shade and Newman, 2011], though some work defines them as unknown cells that have at least one free-space neighbour [Keidar and Kaminka, 2012; 2014]. This paper assumes the first definition.

The naive method for detecting frontiers is to evaluate each cell in the map in turn by checking its neighbours. Another approach is Wavefront Frontier Detection (WFD) [Keidar and Kaminka, 2012; 2014], which consists of running a Breadth First Search (BFS) on the map, beginning at the robot's location, and rippling outwards through free-space until unknown space is encountered. This approach has the advantage of only evaluating a subset of the entire map at any time, though this subset becomes the entire map as exploration progresses.

Fast Frontier Detection (FFD) [Keidar and Kaminka, 2012; 2014] involves only evaluating the cells in each individual scan, particularly the edges of the scan range, along which any new frontier must necessarily lie. A bounding box is built around the scanned area, and cells that were frontiers in this bounding box before the scan are checked to see if they are no longer frontiers at the current time step. This approach is faster than WFD, but requires that frontier detection take place after every scan. These calculations may be wasteful for exploration strategies in which many scans are taken before the robot recalculates frontiers to decide where to move next. FFD is also less likely to correctly detect frontiers at the maximum range of the sensor. Divergence in laser points at extreme ranges means that Bresenham's line algorithm[1], used by FFD to determine the contour evaluated for potential frontiers, will cut across unknown space and not cover all cells that should be detected as frontiers. This makes implementation more complex and restricts the applicability of the strategy to specific sensor ranges and footprints.

Incremental WFD (WFD-INC), and the variant Incremental-Parallel WFD (WFD-IP) [Keidar and Kaminka, 2014] combine the bounding box concept from FFD to WFD, resulting in faster running time.

This paper presents two algorithms for detecting frontiers. The first novel approach is called Expanding Wavefront Frontier Detection (EWFD) which is suitable for scenarios where single or multiple scans might occur before the exploration

[1] A method for line tesselation.

strategy requires updated frontier information. The other novel algorithm, called Naive Active Area frontier detection (NaiveAA) is presented to be used as a more useful benchmark than the simple naive frontier detection algorithm. Both algorithms are analysed and their performances compared theoretically and in simulations with prior state of the art frontier detection methods.

The rest of this paper is structured as follows: the details of the NaiveAA algorithm will be presented in Section 2 and the details and analysis of EWFD presented in Section 3; experimental setup and results are shown and discussed in Section 4; conclusions are drawn in Section 5 and possible future work is suggested.

2 Naive Active Area Frontier Detection

The naive method involves evaluating all cells in the map to detect which are frontiers. If the number of cells in a map is small, and cell evaluations are fast enough, this might be sufficient for some applications. For example on a map which is 48×18.2 metres, if cells are 1 cm in width and length, then there would be 8,790,600 cells. If cell evaluations take 1ms, the Naive approach would take 2.44 hours, if cell evaluations take 1us, then the total would be 8.79 seconds, and if cell evaluations take 1ns, then the total time would be 0.01 seconds. On a 3D map which is $292 \times 167 \times 28$ meters, the number of cells grows to 1,365,392 million, and the time taken by the Naive approach would be, assuming cell evaluation times of 1ns, at least 22.76 minutes. This makes the Naive approach an unreasonable option for large or high resolution environments.

However, if the *active area* of each scan is considered, then only cells in those areas (or volumes in the 3D case) can have changed from unknown to freespace and become frontiers, and only cells bounding the active area can have lost their frontier status when their neighbours in the active area become freespace instead of unknown. The active area of a scan is defined as the set of cells in the map covered by the scan's field of view. This can be easily approximated, without underestimating, through the use of a minimal rectangular bounding box[2] that contains the scan's ray endpoints and the sensor origin.

The naive method can therefore be restricted at any time t to evaluating only cells in the active area, A_t, of any scans taken since the last frontier detection step, and which includes the cells immediately adjacent to the scanned areas. As each cell is evaluated, if it is no longer a frontier, it can be removed from the set of frontiers, F, whereas if it is a new frontier, it can be added to F. This restricted version of the naive approach is called Naive Active Area frontier detection (NaiveAA) (Algorithm 1). Depending on whether the frontier cells need to be labelled as belonging to groups of connected cells, an additional step to the algorithm can be included using an algorithm like Kosaraju's series of Depth First Searches (KOSARAJU_DFS) to determine which frontiers cells are connected in $O(|F|)$ time.

Algorithm 1: Naive Active Area Frontier Detection

```
   Input: F, A_t
   Output: F, labelling
1  for c ∈ A_t do
2      if IS_FREESPACE(c) then
3          if IS_FRONTIER(c) then
4              F ← F ∪ c
5          else if WAS_FRONTIER(c) then
6              F ← F \ c
7  labelling ← KOSARAJU_DFS(F)
```

If there are $|M|$ cells in the map M, and α frontier detection operations take place, the naive algorithm requires $O(\alpha \times |M|)$ operations over the course of exploration. NaiveAA only requires $O(\alpha \times A_{max})$ cell evaluations, where A_{max} is an upper bound on the size of the active area of any scan. If a grid is used to store frontier cell states, then each insertion and deletion is $O(1)$, but if a structure like a kd-tree is used, then each insertion and deletion occurs in time $O(log|F|)$.

If radical changes in the map are expected, then the algorithm can be made more robust by removing the if-statement on line 2 of Algorithm 1 so that all cells are checked to see if they are marked as frontiers when they shouldn't be (for example, if they change from being a freespace cell to being an occupied cell).

3 Expanding-Wavefront Frontier Detection

The key principle of EWFD is that unless space previously known becomes unknown, once a part of the map is defined as inside the explored space, it never needs to be re-evaluated as a potential frontier.

3.1 EWFD Description

Let O_t represent the sequences of observations made between $t-1$ and t, where $t-1$ and t are times at which the frontier detection algorithm is executed. A_t refers then to the combined active area of all individual observations in O_t.

If at time 0, there is no prior known set of frontiers, then the first call to the EWFD algorithm involves a similar approach to WFD: a BFS is performed starting with the free-space the robot is located in (see Algorithm 2). Adjacent free-space cells are added to the queue. As cells adjacent to unknown space are visited, they are labelled as frontier cells. As part of the BFS, visited cells are marked as visited in the occupancy grid. These labels are not removed after the BFS.

Subsequent iterations involve first finding the set of frontiers $F_{t-1} \cap A_t$. These cells are marked as unvisited in the

[2] In the 3D case, this would be achieved using a bounding box defining a 3D volume.

occupancy grid and a BFS is performed starting with the set $F_{t-1} \cap A_t$ in the queue, adding neighbours of unvisited free-space cells to the queue. This enforces a search outwards, into newly discovered free-space, and prevents reevaluation of cells already determined not to be frontiers. As before, when cells adjacent to unknown space are visited, they are labelled as frontier cells. During the BFS, frontier cells in $F_{t-1} \cap A_t$ are evaluated to check whether they are still frontiers.

As with NaiveAA, if frontier grouping and labelling is required, then a series of DFSs is performed on F_t to determine which frontier cells should be grouped together.

Algorithm 2: Expanding Wavefront Frontier Detection

```
   Input: F, A_t, visited, robotStartingCell
   Output: F, labelling, visited
1  if F_{t-1} = ∅ then
2      queue ← robotStartingCell
3  else
4      queue ← F_{t-1} ∩ A_t
5  while queue ≠ ∅ do
6      c ← POP(queue)
7      visited ← visited ∪ c
8      if IS_FREESPACE(c) then
9          if IS_FRONTIER(c) then
10             F_t ← F_t ∪ c
11         else if WAS_FRONTIER(c) then
12             F_t ← F_t \ c
13         C_a ← GET_ADJACENT_CELLS(c) ∩ A_t
14         C_a ← C_a \ visited
15         PUSH(queue, C_a)
16 labelling ← KOSARAJU_DFS(F)
```

3.2 EWFD Soundness and Completeness

An assumption of EWFD is that entropy for each cell can only decrease over time. Given this assumption it can be proved that EWFD is both complete, and sound.

Lemma 3.1 *Suppose f is a frontier cell at t, and was not a frontier cell at $t-1$. Then EWFD will label f as a frontier cell.*

Proof At time 0, in the free-space region that the robot is in, $\mathcal{P}_0^{free}$, all free-space cells are connected; i.e. it is possible to start at one free-space cell, and using a sequence of moves to adjacent cells, end at any other free-space cell in the map. This is either because the only free-space cell is the sensor origin, or because the robot occupies physical space, and therefore that region, which describes the shape of a single physical object, must be connected.

Assuming that at $t-1$, all free-space cells in the robot's world map, $\mathcal{P}_{t-1}^{free}$, are connected. Let the observation at t be denoted O_t. Let $\mathcal{S}(O_t)$ represent the set of cells covered by O_t. The assumptions are that the sensor origin must sit in $\mathcal{P}_{t-1}^{free}$, and that all cells seen must be visible from the sensor origin. This results in each free-space cell in $\mathcal{S}(O_t)$ being connected to the sensor origin, and therefore each free-space cell in $\mathcal{S}(O_t)$ is connected (at least indirectly) to each other free-space cell in $\mathcal{S}(O_t)$. Since the sensor origin must also lie in known space, this means that each free-space cell in $\mathcal{S}(O_t)$ is connected to $\mathcal{P}_{t-1}^{free}$. Therefore all free-space cells in $\mathcal{P}_t^{free}$ are also connected.

Let us assume a BFS is performed starting from a free-space cell inside a known region of space. The BFS adds the adjacent cells to free-space cells it visits to the queue, and is not permitted to revisit cells it has already seen. Since all free-space cells inside a region are connected, the BFS is guaranteed to visit each free-space cell in that region. Since all free-space cells are visited, those that are adjacent to unknown space (i.e. frontier cells) will also be visited.

By definition, $\mathcal{P}_{t-1}^{free}$ is a subset of $\mathcal{P}_t^{free}$, with the frontier at $t-1$, F_{t-1} being included or enclosed by the frontier at t, F_t. Since new frontiers must be new free-space cells (based on the assumption that a cell cannot transition from known to unknown value), the region that needs to be evaluated for new frontiers is $\mathcal{P}_t^{free} \setminus \mathcal{P}_{t-1}^{free}$ (see Figure 1).

The region evaluated by the BFS in EWFD is $(\mathcal{P}_t^{free} \setminus \mathcal{P}_{t-1}^{free}) \cup (F_{t-1} \cap A_t)$. All cells, f that are a frontier cell at t, which were not a frontier cell at $t-1$ will therefore be evaluated and appropriately labelled as frontiers.

Lemma 3.2 *Suppose c is a cell that is not a free-space cell on the boundary between known and unknown space at t, then it will not be marked as a frontier at t.*

Proof If c is not a frontier cell at t, there are two possible cases:

Case 1. c was not a frontier cell at $t-1$. If c is free-space, then it either has been previously evaluated by EWFD and was not marked as a frontier, or it will be evaluated at t and will not be marked as a frontier.

Case 2. c was a frontier cell at $t-1$. If c was a frontier at $t-1$, then it will be in the queue of the BFS performed at t. It will therefore be removed as a frontier when it gets evaluated.

3.3 EWFD Algorithm Analysis

The various frontier detection algorithms can be best compared by counting the number of cell evaluations required to determine all frontier cells. Let the set of cells in the map be M, and let each frontier detection algorithm be iterated t times before exploration terminates. Let the number of cells covered by the sensor at each iteration be bounded from above by $\mathcal{S}_{max}$.

Several data structures are assumed to exist, as part of EWFD. There is a balanced k-d tree [Kanth and Singh, 1997; Procopiuc *et al.*, 2003] of frontier cells tuples, each tuple contains the cell coordinates. The robot keeps a map of the envi-

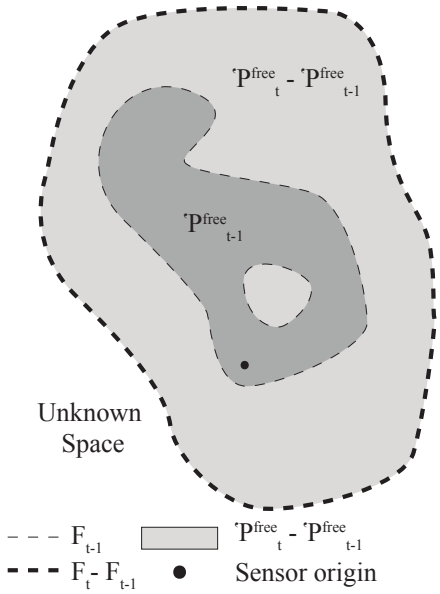


Figure 1: At time t, only cells in $\mathcal{P}_t^{free} - \mathcal{P}_{t-1}^{free}$ will be evaluated.

ronment as an occupancy grid. Each cell has a status, being either free-space, obstacle, or unknown. Each cell is labelled with its frontier status: either it is a frontier cell or it is not. It is also labelled as visited by the BFSs or not. If it is a frontier cell, the cell contains a pointer to the tuple in the frontier tree.

The rest of this section will step through the algorithm and determine its order of complexity in big-O notation.

1. *Finding Relevant Frontiers in F_{t-1}:* The first step of the EWFD algorithm involves finding the frontiers in the current frontier list, F_{t-1}, that are inside the bounding box defining the active area, A_t. Since the frontier cells, F_{t-1}, are sorted, finding the cells bound by these values in the balanced k-d tree will involve:

$$O(|F_{t-1}|^{(d-1)/d} + |F_{t-1} \cap A_t|), \tag{1}$$

where d is the dimensionality of the tuples.

2. *Evaluating Cells for Frontier Status:* The BFS used by EWFD starts with the set $F_{t-1} \cap A_t$ in its queue. It will evaluate all cells that have been newly marked as free-space since the last iteration of EWFD; i.e. cells in free-space at t, $\mathcal{P}_t^{free}$ minus those that were in free-space at $t-1$, $\mathcal{P}_{t-1}^{free}$. Given that all cells have a constant fixed number of adjacent cells, the BFS runs in order:

$$O(|\mathcal{P}_t^{free} \setminus \mathcal{P}_{t-1}^{free}| + |F_{t-1} \cap A_t|) \tag{2}$$

3. *Removing Frontiers:* At any iteration, the most frontier cells that need to be removed is the set $F_{t-1} \cap A_t$. These cells will need to have their status in the occupancy grid changed, then be removed from the k-d tree of tuples. In total, this will run in:

$$O(|F_{t-1} \cap A_t| \times log|F_{t-1}|) \tag{3}$$

4. *Inserting Frontiers:* The largest number of cells needing to be inserted at timestep t is $|F_t \setminus F_{t-1}|$, so that the complexity of inserting is:

$$O(|(F_t \setminus F_{t-1}) \cap (\mathcal{P}_t^{free} \setminus \mathcal{P}_{t-1}^{free})| \times log|F_{t-1}|) \tag{4}$$

5. *Labelling Frontiers:* Using the occupancy grid to represent edges of a graph, and using the tree of frontier cells as the list of vertices, a series of DFSs can be used to determine the connected sets that frontier cells belong to. This runs in order:

$$O(|F_t|) \tag{5}$$

$|A_t|$ is bound from above by the maximum sensor observation area, A_{max}, times the number of observations, o_t that have taken place since the previous call of the frontier detection algorithm. Similar bounds apply to $|F_{t-1} \cap A_t|$, $|\mathcal{P}_t^{free} \setminus \mathcal{P}_{t-1}^{free}|$, and $|(F_t \setminus F_{t-1}) \cap (\mathcal{P}_t^{free} \setminus \mathcal{P}_{t-1}^{free})|$. Combining equations (1), (2), (3), (4), and (5), the total complexity can therefore be reduced to:

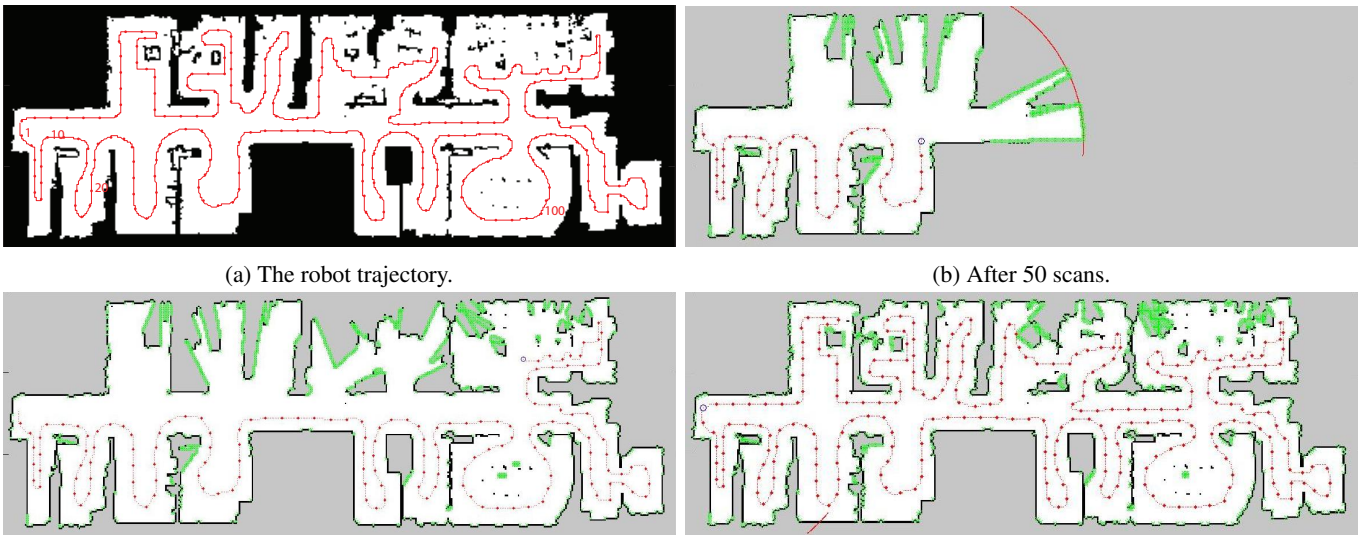
$$O(|F_{t-1}|^{(d-1)/d} + o_t \times log|F_{t-1}| + |F_t|) \tag{6}$$

As the observation overlap increases, the minimum bound is given by the case where observations overlap completely:

$$\begin{aligned} &\Omega(|F_{t-1}|^{(d-1)/d} + log|F_{t-1}| + |F_t|) \\ &= \Omega(|F_t|) \end{aligned} \tag{7}$$

Over all t steps of exploration, the sum of $|\mathcal{P}_t^{free} \setminus \mathcal{P}_{t-1}^{free}|$ becomes simply $|M_{free}|$. This is therefore an intuitive lower bound not captured by big-O notation.

If observations don't overlap, then when EWFD is performed at t, it is similar to performing o calls of EWFD, one after each individual observation, save that the overhead of finding previous frontiers and labelling frontier groups, is lower. As observations overlap, it is similar to performing EWFD for a single new observation. The best and worst case for EWFD therefore depends on the behaviour of frontiers. The best case is where frontiers remain a constant size, the worst is when the set F_t increases by $o_t \times A_{max}$ per timestep, so that over all exploration the lower and upper bounds of EWFD are $\Omega(t)$ and $O((\sum_{i=1}^{t} o_i)^2)$.



(a) The robot trajectory.

(b) After 50 scans.

(c) After 150 scans.

(d) After 252 scans.

Figure 2: The Freiburg environment (Trajectory 1), 414×149 cells. a) Lines show the robot trajectory, and the dots show the poses where scans were taken and frontiers evaluated. b, c, d) The robot is shown with a small blue circle, frontiers are marked with green crosses, and a red arc shows the robot's FOV.

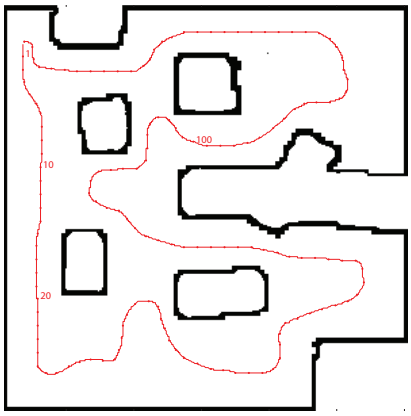


Figure 3: A cave environment (Trajectory 1), 305×305 cells. The red line shows the robot trajectory, and dots show the poses where scans were taken, and frontiers evaluated.

4 Experimental Results

Simulations were run on an Intel Core 2 Duo 3.00GHz desktop computer in Matlab. Simulations involved a point robot being moved through a map and running frontier detection using each algorithm at various intervals. Several environments were used, and the runs for each were repeated 5 times to obtain an average and standard deviation for the running time.

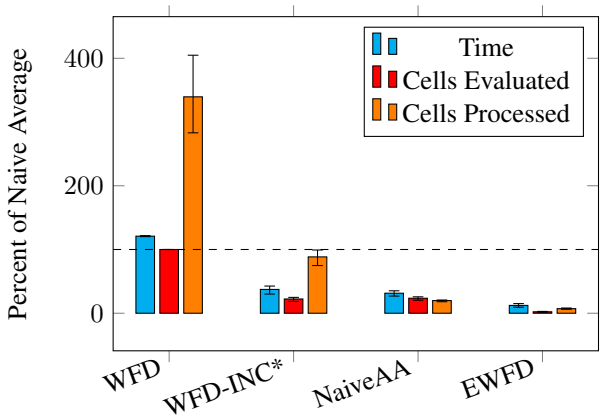
4.1 Experiment Design

The algorithms compared were the Naive approach, WFD, WFD-INC*, NaiveAA, and EWFD. The asterisk over WFD-INC is to denote the fact that in this paper, the algorithm was implemented without pivot lists, and instead added and removed frontiers from the same type of set structure as the other algorithms, and Koseraju's connectivity algorithm was run to determine frontier grouping, as with the other frontier detection methods. This was done so that all the algorithms would be equally applicable in 3D.

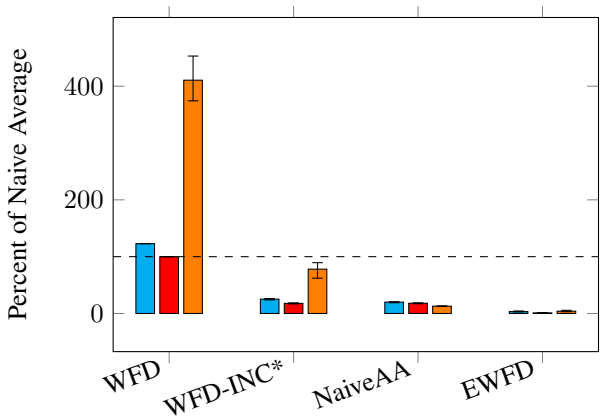
FFD was not implemented because of some ambiguity in the descriptions in the [Keidar and Kaminka, 2014] paper that made some portions of the algorithm's intended behaviour unclear. For example, Bresenham's line algorithm is used to create the contour between endpoints of the sensor scan, which will then be evaluated, but this means that the endpoints start in freespace and then potentially move through unknown space. Therefore some cells that should be evaluated as frontiers will not be. This problem seems to remain whether the definition of frontiers is as freespace cells with at least one unknown neighbour, or whether it is as unknown cells with at least one freespace neighbour. Since the implementation attempted by the authors might not result in a fair representation of the algorithm, FFD, was omitted from the list of algorithms to use as comparisons to NaiveAA and EWFD.

It should be noted that after adding new frontiers to the frontier set, WFD-INC and FFD both involve a step in which all cells in the active area of the scan are checked, to see whether they were frontiers but are not now, and should be removed. This is similar to performing NaiveAA. It is therefore expected that NaiveAA would perform similarly to WFD-INC and FFD, but involve less computational overhead.

EWFD did not use kd-trees to store frontiers, and instead



(a) The Freiburg trajectories.



(b) The Cave trajectories.

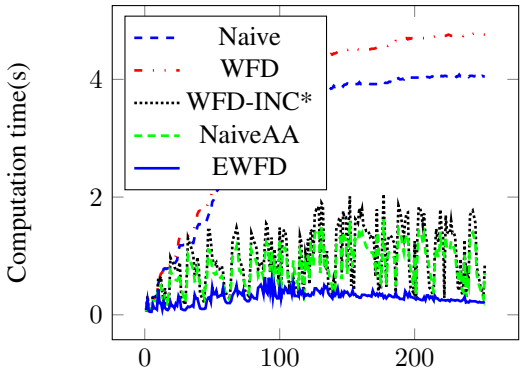
Figure 4: Comparison of each algorithm in the Freiburg and cave environments summarised for all trajectories. Bars shows the average, while the ticks show the minimum and maximum values from the data sets.

used a matrix implementation in Matlab[3], in the same fashion as the other algorithm implementations stored their frontiers. This meant that retrieval times for frontiers in the active area are slower than what might be otherwise expected.

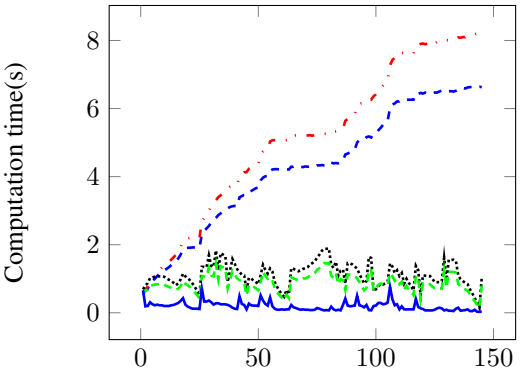
The set of frontiers for all algorithms was compared for correctness to the set found by the Naive approach.

The 2D environments used were the Frieburg labs map (see Figure 2), and a cave environment (see Figure 3) from the Radish repository [Howard and Roy, 2003]. Several trajectories were created by driving the robot through the environment. Frontiers were evaluated at regular intervals using each algorithm. The Freiburg environment trajectory 1 involved 252 scans and frontier detection steps and the cave environment trajectory 1 involved 145 scans and frontier detection steps.

[3]A k-d tree would have been very inefficient in Matlab, or have to be run through a C++ linked library, which would make the results ambiguous when compared to the other algorithms implemented solely in Matlab.



(a) Freiburg trajectory 1.



(b) Cave trajectory 1.

Figure 5: Computation time for each algorithm at each iteration.

4.2 Results

The results for all 6 trajectories over the both environments were fairly similar, so only a few are shown here in detail. The others are listed in the Appendix. Tables 1 and 2 show the results from two set of experiments. The metrics presented are total computation time for detecting frontiers, the time for labelling them, and the average time per iteration. The number of cells evaluated is the number of times a cell was checked to see whether it was a frontier or not. The number of cells processed is an attempt to capture the amount of computational overhead required in dealing with more complex operations than a simple iteration through a list or grid. The number of cells processed is the number of times a cell was "handled" by the algorithm. In the case of WFD, WFD-INC*, EWFD, "Cells Processed" is the number of times the algorithm had to decide whether or not a cell should go in the queue, and in the case of the Naive, or NaiveAA algorithms, it is the number of times the algorithm needed to decide if a cell should be evaluated to check if it was a frontier cell. The result for each metric for each algorithm is the average of all runs, and the $\pm$ column records the standard deviation.

Figure 4 shows the performance of WFD, WFD-INC*, NaiveAA, and EWFD relative to the Naive approach accord-

Table 1: Results in Freiburg map, trajectory 1.

	Naive	±	WFD	±	WFD-INC	±	NaiveAA	±	EWFD	±
Detecting (s)	726.71	3.65	896.20	4.53	181.17	0.83	156.49	0.74	20.42	0.13
Labelling (s)	53.53	0.27	53.53	0.27	53.53	0.27	53.53	0.27	53.53	0.27
Total (s)	780.25	3.91	949.73	4.80	234.70	1.09	210.03	1.01	73.96	0.40
Avg. Time/Iteration	3.10	0.02	3.77	0.02	0.93	0.00	0.83	0.00	0.29	0.00
Cells Evaluated	6,991,517	-	6,991,517	-	1,292,630	-	1,428,509	-	108,247	-
Cells Processed	15,544,872	-	62,923,662	-	11,633,670	-	2,805,111	-	974,223	-

Table 2: Results in Cave map, trajectory 1.

	Naive	±	WFD	±	WFD-INC	±	NaiveAA	±	EWFD	±
Detecting (s)	605.60	4.35	746.30	1.88	145.20	0.36	115.43	0.50	15.18	0.07
Labelling (s)	12.04	0.04	12.04	0.04	12.04	0.04	12.04	0.04	12.04	0.04
Total (s)	617.65	4.40	758.34	1.93	157.24	0.40	127.47	0.55	27.22	0.12
Avg. Time/Iteration	4.26	0.03	5.23	0.01	1.08	0.00	0.88	0.00	0.19	0.00
Cells Evaluated	5,864,472	-	5,864,472	-	1,046,799	-	1,062,647	-	84,016	-
Cells Processed	13,488,625	-	52,780,248	-	8,374,392	-	1,772,547	-	756,144	-

ing to the three key metrics. Each graph is a summary of the data collected for all trajectories in that environment, where the bar shows the average, and the ticks show the maximum and minimum values over the set of three trajectories.

In all trajectories, all algorithms resulted in 100% accuracy relative to the set of frontiers detected by the Naive strategy.

In both environments, a surprising result of these experiments was that WFD was found to be slower than the Naive approach, and WFD-INC* was found to be slower than the NaiveAA approach. This is because of the overhead of running a BFS over the cells, and maintaining a queue.

As expected, NaiveAA was faster than Naive, allowing it to be a good benchmark for other algorithms, like WFD-INC* and EWFD, that are bounded by the active area of sensor scans.

EWFD performed fastest of all algorithms, requiring less than 10% of the time required by the Naive approach in the Freiburg map, and less than 5% in the cave environment. EWFD required approximately 33% the time required by WFD-INC* in the Freiburg environment, and 20% of the time required by WFD-INC* in the cave environment.

These results held true not only for the total time over all exploration, but also for the computation time at each iteration (see Figure 5). Naive and WFD initially take little time, and gradually take more and more time as the amount of known freespace in the map increases. WFD-INC*, NaiveAA, and EWFD computation times did not measurably increase as a function of time, varying mostly based on the size of the active area of each scan.

5 Conclusions and Future Work

This paper presented and demonstrated two algorithms for frontier detection. The first, Naive Active Area frontier detection (NaiveAA) sets a basic benchmark for frontier detection algorithms that are bound to the scanned region at each timestep. Since there is very little computational overhead in its implementation, it runs relatively quickly, and provides a more useful comparison for other frontier detection approaches than the naive approach.

The second approach was Expanding Wavefront Frontier Detection (EWFD), which was shown to be on average, faster than the other approaches, including NaiveAA and WFD-INC*. This was a result of performing fewer cell evaluations, and minimising the overhead in choosing which cells to evaluate.

The advantage of EWFD is that it is faster on average than other state-of-the-art approaches. Unlike WFD-INC, EWFD is directly useable in 3D environments. EWFD is able to be used either after every scan, or after many scans have been taken. In order to achieve this speed however, assumptions are made that make it unsuitable in certain circumstances: that known cells cannot become unknown, or change state. If either of these situations occurs, it is possible that EWFD will not correctly identify the resulting frontier cells.

NaiveAA does not suffer from these problems however, and has the advantages of WFD-INC* and EWFD in having running time bounded by the size of the sensor scan.

It would be possible after each scan to detect whether a cell changed state from known to unknown, or from occupied to freespace, in which case the safest and fastest approach would be to use NaiveAA, and use EWFD otherwise.

As mentioned in the experiment design, EWFD could be expected to run faster if a kd-tree was implemented and allowed for faster range searches of frontiers within the active area.

Further experiments should be run to determine the behaviour of each algorithm in different scenarios. For example, if scans take place closer together or further apart and if scans involve more or less new information. The effects of the increased rate of frontier detection on exploration should be measured to determine not just how much faster exploration can be performed as a result of faster computation times, but

also how much more efficiently (i.e. resulting in less robot motion).

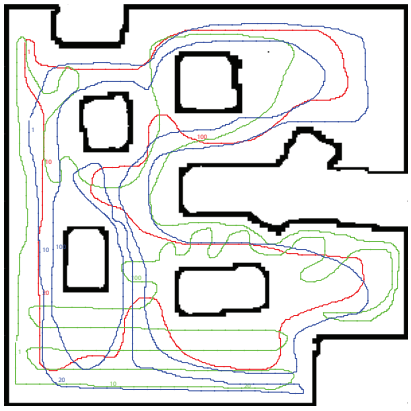
Acknowledgments

This work is supported by the Australian Research Council (ARC) Linkage Grant (LP100200750), the NSW Roads and Maritime Services, and the Centre for Autonomous Systems (CAS) at the University of Technology, Sydney.

Appendix

This section contains results from the additional trajectories run over the two environments. Figure 6 shows the two additional trajectories used to test the algorithms in each environment. Tables 3, 5, 4, and 6 show the results, and these are part of the data set summarised in Figure 4. The time taken at each iteration of frontier detection by each algorithm is shown in Figure 7 for each environment and trajectory.

(a) Freiburg trajectory 1 (red), 2 (green), and 3 (blue).

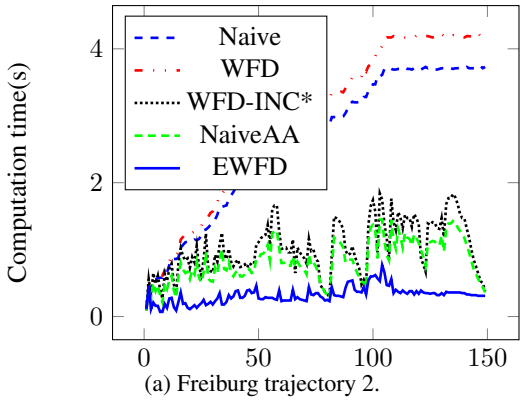


(b) Cave trajectory 1 (red), 2 (green), and 3 (blue).

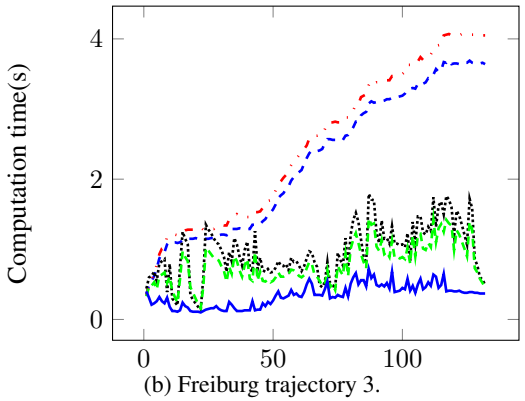
Figure 6: All 6 trajectories. Lines show the robot trajectory, and the dots show the poses where scans were taken and frontiers evaluated.

References

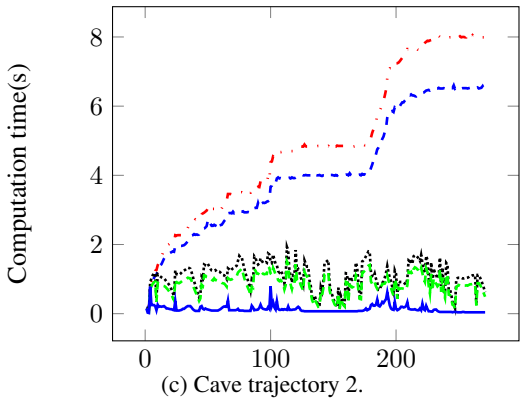
[Burgard *et al.*, 2000] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 476–481, 2000.



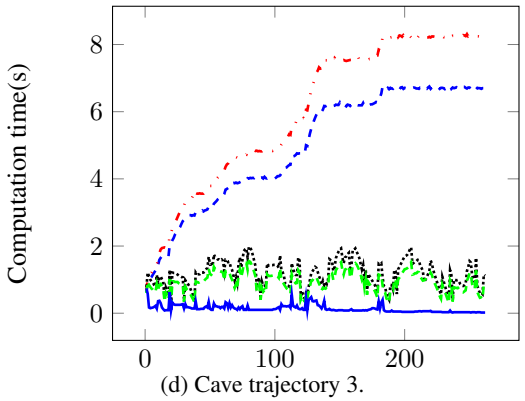
(a) Freiburg trajectory 2.



(b) Freiburg trajectory 3.



(c) Cave trajectory 2.



(d) Cave trajectory 3.

Figure 7: Computation time for each algorithm at each iteration.

Table 3: Results in Freiburg map, trajectory 2.

	Naive	±	WFD	±	WFD-INC	±	NaiveAA	±	EWFD	±
Detecting (s)	355.74	1.60	436.41	1.09	120.48	0.26	92.35	0.31	15.39	0.07
Labelling (s)	32.15	0.09	32.15	0.09	32.15	0.09	32.15	0.09	32.15	0.09
Total (s)	387.90	1.69	468.57	1.18	152.63	0.35	124.51	0.40	47.54	0.16
Avg. Time/Iteration	1.54	0.01	1.86	0.00	0.61	0.00	0.49	0.00	0.19	0.00
Cells Evaluated	3,377,384	-	3,377,384	-	801,459	-	825,402	-	81,057	-
Cells Processed	9,191,214	-	30,396,456	-	9,118,150	-	1,905,019	-	648,456	-

Table 4: Results in Freiburg map, trajectory 3.

	Naive	±	WFD	±	WFD-INC	±	NaiveAA	±	EWFD	±
Detecting (s)	270.15	0.69	330.95	0.80	97.51	0.30	75.09	0.27	14.72	0.08
Labelling (s)	30.88	0.05	30.88	0.05	30.88	0.05	30.88	0.05	30.88	0.05
Total (s)	301.03	0.75	361.83	0.85	128.39	0.36	105.97	0.32	45.60	0.13
Avg. Time/Iteration	1.19	0.00	1.44	0.00	0.51	0.00	0.42	0.00	0.18	0.00
Cells Evaluated	2,560,652	-	2,560,652	-	637,010	-	659,824	-	74,103	-
Cells Processed	8,142,552	-	23,045,868	-	7,422,285	-	1,689,195	-	666,927	-

Table 5: Results in Cave map, trajectory 2.

	Naive	±	WFD	±	WFD-INC	±	NaiveAA	±	EWFD	±
Detecting (s)	1098.79	4.82	1353.08	6.62	275.75	1.31	212.42	0.83	15.54	0.06
Labelling (s)	16.29	8.15	16.29	8.15	16.29	8.15	16.29	8.15	16.29	8.15
Total (s)	1115.08	12.96	1369.37	14.77	292.04	9.46	228.71	8.97	31.83	8.21
Avg. Time/Iteration	4.42	0.05	5.43	0.06	1.16	0.04	0.91	0.04	0.13	0.03
Cells Evaluated	10,484,511	-	10,484,511	-	1,957,872	-	1,968,978	-	87,945	-
Cells Processed	25,209,775	-	94,360,599	-	20,817,419	-	3,196,571	-	791,505	-

Table 6: Results in Cave map, trajectory 3.

	Naive	±	WFD	±	WFD-INC	±	NaiveAA	±	EWFD	±
Detecting (s)	1281.08	1.40	1581.30	2.94	291.23	0.52	225.29	0.46	15.65	0.07
Labelling (s)	20.20	0.08	20.20	0.08	20.20	0.08	20.20	0.08	20.20	0.08
Total (s)	1301.28	1.48	1601.50	3.02	311.43	0.60	245.48	0.54	35.85	0.16
Avg. Time/Iteration	5.16	0.01	6.36	0.01	1.24	0.00	0.97	0.00	0.14	0.00
Cells Evaluated	12,270,470	-	12,270,470	-	2,074,337	-	2,096,590	-	86,917	-
Cells Processed	24,372,550	-	110,434,230	-	21,796,080	-	3,127,047	-	782,253	-

[Digor *et al.*, 2010] E. Digor, A. Birk, and A. Nüchter. Exploration strategies for a robot with a continously rotating 3d scanner. In *Proceedings of the Second international conference on Simulation, modeling, and programming for autonomous robots*, SIMPAR'10, pages 374–386, Berlin, Heidelberg, 2010. Springer-Verlag.

[Faigl and Kulich, 2013] J. Faigl and M. Kulich. On determination of goal candidates in frontier-based multi-robot exploration. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 210–215, Sept 2013.

[Freda and Oriolo, 2005] L. Freda and G. Oriolo. Frontier-based probabilistic strategies for sensor-based exploration. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 3881–3887, 2005.

[Howard and Roy, 2003] Andrew Howard and Nick Roy. Radish: The robotics data set repository, 2003.

[Kanth and Singh, 1997] K. V. Ravi Kanth and Ambuj K. Singh. Optimal dynamic range searching in non-replicating index structures. In *In Proc. International Conference on Database Theory*, pages 257–276, 1997.

[Keidar and Kaminka, 2012] Matan Keidar and Gal A. Kaminka. Robot exploration with fast frontier detection: Theory and experiments. In *Autonomous Agents and Multiagent Systems - Volume 1, Proceedings of the 11th International Conference on*, pages 113–120, 2012.

[Keidar and Kaminka, 2014] Matan Keidar and Gal A. Kaminka. Efficient frontier detection for robot exploration. *The International Journal of Robotics Research*, 33(2):215–236, 2014.

[Mobarhani *et al.*, 2011] Amir Mobarhani, Shaghayegh Nazari, Amir H. Tamjidi, and H.D. Taghirad. Histogram based frontier exploration. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1128–1133, Sept 2011.

[Procopiuc *et al.*, 2003] Octavian Procopiuc, PankajK. Agarwal, Lars Arge, and JeffreyScott Vitter. Bkd-tree:

A dynamic scalable kd-tree. In *Advances in Spatial and Temporal Databases*, volume 2750, pages 46–65. Springer Berlin Heidelberg, 2003.

[Reid *et al.*, 2013] R. Reid, A. Cann, C. Meiklejohn, L. Poli, A. Boeing, and T. Braunl. Cooperative multi-robot navigation, exploration, mapping and object detection with ros. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 1083–1088, June 2013.

[Shade and Newman, 2011] Robbie Shade and Paul Newman. Choosing where to go: Complete 3D exploration with stereo. In *Robotics and Automation (ICRA), Proc. IEEE International Conference on*, pages 2806–2811, 2011.

[Wettach and Berns, 2010] Jens Wettach and Karsten Berns. Dynamic frontier based exploration with a mobile indoor robot. In *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pages 1–8, June 2010.

[Yamauchi *et al.*, 1998] B. Yamauchi, A. Schultz, and W. Adams. Mobile robot exploration and map-building with continuous localization. In *Robotics and Automation, Proc. IEEE International Conference on*, volume 4, pages 3715–3720, 1998.

[Yamauchi, 1997] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Robotics and Automation (CIRA), Proc. Symp. IEEE Int Computational Intelligence in*, pages 146 – 151, 1997.

[Yamauchi, 1998] B. Yamauchi. Frontier-based exploration using multiple robots, 1998.