Incremental frontier generation for improvement of exploration efficiency

Senarathne, P. G. Chaminda Namal

2015

Senarathne, P. G. Chaminda Namal. (2015). Incremental frontier generation for improvement of exploration efficiency. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/65285>

<https://doi.org/10.32657/10356/65285>

INCREMENTAL FRONTIER GENERATION FOR IMPROVEMENT OF EXPLORATION EFFICIENCY

P. G. CHAMINDA NAMAL SENARATHNE

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2015

Acknowledgements

First and foremost I would like to thank my advisor Prof. Wang Danwei for his unending support in completing this dissertation and his patience throughout the course of my study at NTU. I'm also grateful to him for giving me an opportunity to participate in TechX Robotics Challenge 2013 as part of NTU's E-Mobile team where I gained a lot of knowledge and skills in developing and deploying real world robotic systems.

I would also like to thank late Prof. Sardha Wijesoma, who was my supervisor during the initial years of my PhD, for providing me with an opportunity to read for a PhD and introducing me to field robotics.

I am grateful to NTU for providing me with the financial support and lab resources to carry out my research work and also to Mr. Chia, Mr. Neo and Mr. Tan for providing the logistical support to conduct various experiments. I also wish to thank my lab mates Diluka, Bharath, Kwang Wee, JunFeng and Akshay for all the valuable discussions and the advises given on conducting research. I must specially mention Diluka's help on settling in Singapore as well.

Furthermore, I would like to thank my parents who instilled in me the value of education, hard work and perseverance and without whom none of this is possible; my loving wife Hasini and her family for all the support and encouraging me to do my best; and my best friends Chathuranga, Gamaya, Buddhika and Thilok who even with their busy schedules kept in touch and helped ease the pressure of doing research.

Finally, I would like to thank Akshay for all the enjoyable tea time discussions and the camaraderie over the last five years; Buddhika, Gamini, Kumudu, Dulika, Kasun, Nawoda, Asiri and Sampath and the rest of the

Sri Lankan gang at NTU for their friendships; Chathura and Dinusha for hosting me over many meals during my bachelor days; and the amazing group of people Ehsan, Yosmar, Mou Wei, Rakesh, Lubing and Shenghai that I had the privilege of working with during the TechX 2013 competition which made my life in Singapore vividly memorable.

Abstract

Autonomously navigating a single or a multiple set of robots to map a previously unknown environment is known as autonomous exploration and is a fundamental operation in many robotic missions. Given the unknown nature of the environment, planning for the optimal exploratory path for a robot or a group of robots to completely cover the given environment is known to be NP-Hard. Therefore, all autonomous exploration strategies operate by successively selecting the next best sensing location as robot's intermediate target point in a greedy manner, until the environment is completely mapped. Frontier based exploration and its many variants, where the robot's next sensing location is selected from the boundary between the mapped-free and unknown space (i.e. frontier) from the occupancy grid representation of the environment, has emerged as the baseline exploration strategy due to its simplicity and scalability. This dissertation examines the efficient generation and management of frontier information to refresh frontier cell information at a high frequency, and how these frequently available frontier cells can be utilized to further improve the efficiency of exploration missions. Incremental algorithms to extract both safe and reachable area from the occupancy grid map by processing only the locally updated map data and in turn generate only the valid frontier cells in an efficient manner is introduced. The proposed algorithms are validated in real world experimental data and is shown to drastically reduce and bound the execution time throughout exploration missions with very high detection accuracy compared to state of the art incremental frontier extraction methods.

The resulting high frequency frontier cell refreshing is utilized to propose an improvement to the traditional frontier based exploration strategy as

the next contribution. A probabilistic decision step, based on the most recent frontier cell information is proposed to decide the desirability of the remaining motion of the robot to its current intermediate target and to re-plan when the current motion is undesirable. This reduces redundant exploratory motions of the robot and improves the total efficiency of exploration missions. Experiments carried out in both real and simulated, indoor and outdoor environments corroborate the efficiency of this method and also reveal that the re-planning capability is especially suitable for improving efficiency of indoor exploration missions.

This re-planning ability is then utilized to propose a new goal oriented navigation algorithm in unknown environments based on frontier based exploration. This is achieved by directing the exploration towards the goal. A new utility function based on the combined distance estimate to the goal from the robot's position through known and unknown cells is introduced for directing the exploration steps. The distance from frontier cells to the goal which is still in unknown area is estimated using a visibility graph generated through the unknown area thus negating the need to access the entire search space. Experimental results show that the proposed navigation algorithm is an alternative to popular dynamic planning algorithms and that it provides the same travel distance efficiency and a superior re-planning cost efficiency.

Finally, the efficiency of exploration missions in unbounded environments is discussed in terms of the compactness of generated maps. Balanced mapping in all possible directions to generate compact maps is captured as balancing the arrival time of the robot to available frontier cells in the robot's neighborhood. The proposed efficient frontier management algorithms are employed to efficiently update the waiting times of individual frontier cells for robot's arrival and a waiting time variance based utility function is derived to guide the robot in a compact exploratory motion. Simulated experiments conducted in multiple environments for both single and multiple robot systems validate the efficiency of the proposed compact exploration strategy in mapping unbounded environments.

Contents

List of Figures	xi
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Objective and Approach	4
1.3 Contributions	5
1.4 Thesis organization	6
2 Background	11
2.1 Notations and definitions	11
2.1.1 Occupancy grid map and related data structures	11
2.1.2 Traversability, Reachability and Frontier cells	15
2.1.3 Map update step	15
2.2 Frontier based Exploration	17
2.2.1 Introduction to frontier based exploration	17
2.2.2 Traditional frontier cell extraction algorithm	18
2.2.3 Calculation of utilities of frontier cells	20
2.2.4 Improving efficiency of frontier based exploration	25
2.2.5 Extending frontier based exploration for multi-robot systems	28
2.2.6 Exploration and SLAM	28
2.3 Efficient approaches for frontier cell extraction	29
2.3.1 Wavefront Frontier Detector (WFD)	30
2.3.2 Fast Frontier Detector (FFD)	30
2.3.3 Incremental WFD (WFD-INC)	31
2.3.4 Oriented bounding box based Frontier Detector (OFD)	32

CONTENTS

2.3.5	Frontier databases	32
2.3.6	Valid frontier extraction	34
3	Incremental Extraction of Safe and Reachable Map Information	35
3.1	Incremental identification of safe cells	36
3.2	Incremental detection of reachability	38
3.2.1	Checking the reachability of a grid cell	41
3.2.2	Checking for reachability across update steps	43
3.2.3	Maintenance of the proposed data structure	45
3.3	Experimental results	64
3.3.1	Evaluation of efficiency of incremental obstacle inflation	67
3.3.2	Evaluation of efficiency of reachability computation	68
3.3.3	The search depth of reachability graph	70
3.4	Summary	71
4	Incremental Safe and Reachable Frontier Detector	75
4.1	Incremental extraction of mapped-free-space boundary cells as contours	76
4.1.1	Filtering erroneous boundary cells from $\mathcal{B}_k^{new}$	77
4.1.2	Chaining new boundary cells to contours	79
4.2	Contour Database	85
4.2.1	MX-Quadtree	86
4.2.2	Linked MX-Quadtree (L-MX-Quadtree) to store contours	86
4.2.3	Updating L-MX-Quadtree	90
4.3	Safe and Reachable Frontier Detector (SRFD)	93
4.3.1	Retrieving frontier contours	94
4.4	Experimental results	95
4.4.1	Comparison of execution time for frontier generation operations	98
4.4.2	Evaluation of accuracy	102
4.5	Summary	107
5	Improving Frontier Based Exploration with Task Cancellation	109
5.1	Motivation and background	110
5.2	Formulation of exploration task cancellation step	111
5.2.1	Estimating neighborhood frontier cell selection probability	114

5.2.2	Selecting the neighborhood of intermediate target	114
5.3	Experimental results	115
5.3.1	Analyzing the effect of parameters on different environments .	116
5.3.2	The effect of task cancellation on exploration missions	120
5.3.3	Number of decision steps to complete missions	123
5.4	Summary	124
6	Directed Exploration for Goal Oriented Navigation in Unknown Environments	127
6.1	Motivation and background	128
6.2	Goal Directed Exploration : Problem formulation	130
6.3	Estimating shortest distances to goal through the unknown area . . .	133
6.3.1	Generating goal visibility graph through unknown area	133
6.3.2	Estimating the shortest distance from each frontier cell to goal	134
6.4	Detecting and avoiding dead-ends	138
6.5	Experimental evaluation	140
6.5.1	Completeness and efficiency of the generated paths	140
6.5.2	Quantitative comparison	142
6.5.3	Real world validation	143
6.6	Summary	144
7	Compact Exploration of Unbounded Environments	147
7.1	Motivation and background	148
7.2	Minimizing Waiting Time Variance based utility function	150
7.2.1	Formulation	150
7.2.2	Efficient computation of the cell utility	153
7.3	Compact exploration algorithms	157
7.3.1	Single robot compact exploration	157
7.3.2	Multi robot compact exploration	161
7.4	Experimental evaluation	162
7.4.1	Compact Ratio and Explored Ratio of the OBBs of map . . .	164
7.4.2	Compactness of the map	170
7.5	Summary	171

CONTENTS

8 Conclusion and Recommendations	173
8.1 Conclusion	173
8.2 Recommendations for future research	175
A Oriented Bounding Box based Frontier Detector (OFD)	177
A.1 Updating a frontier contour between two map update steps	177
A.2 Tracking cell status changes through an update window	178
A.3 Oriented bounding boxes (OBBs) for frontier-contour generation . . .	180
A.4 Implementation	181
A.4.1 Frontier Database	181
A.4.2 Algorithms	182
B Re-positioning of robots for efficient multi-robot exploration	185
B.1 Re-positioning robots for coordinated exploration	185
B.2 Discretization of map space	186
B.3 Calculation of the utility for a frontier block	188
B.3.1 Information gain	188
B.3.2 Influence factor	189
B.4 Strategy level decider	190
C Author's Publications	193
References	195
Glossary	207

List of Figures

1.1	Illustration of frontier cells extracted from an occupancy grid map . .	3
1.2	Illustration of phantom frontiers generated due to mapping errors . .	5
2.1	Illustration of inflating obstacles and identifying the safe cells	13
2.2	Relationship between map update step and sensor readings.	16
2.3	Frontier based exploration example - step 1	18
2.4	Frontier based exploration example - step 2	18
2.5	Frontier based exploration example - step 3	19
2.6	Frontier based exploration example - step 4	19
2.7	Illustration of major operations of the traditional frontier extraction algorithm	21
3.1	Illustration of addition and removal of obstacle grid cells in successive map update steps and their effect on the inflated cells	37
3.2	Propagation of wavefronts during obstacle cell inflation, deflation and repairing operations	39
3.3	Relationship between connected traversable areas and safe-patches . .	41
3.4	Illustration of a graph of safe-patches	41
3.5	Reachable connected component and its tree representation	44
3.6	Progression of depth of nodes of the spanning tree during an exploration	46
3.7	The order of update operations conducted to maintain the spanning tree data structure	46
3.8	Splitting of the safe-patch based on new obstacle information	47
3.9	Adding new nodes to the spanning tree based on new map information	48

LIST OF FIGURES

3.10	Updating the connectivity of the spanning tree considering newly added nodes	49
3.11	Updating safe-patches with new obstacle information	53
3.12	Comparison of safe-patch sizes generated due to (a) straight line motion and (b) curved motion.	63
3.13	Map of the IRL environment	65
3.14	Map of the ARRL environment	66
3.15	P3AT robot equipped with a Hokuyo LRF used during experiments .	66
3.16	Total number of cells processed for each deflation, repair and inflation tasks throughout the mission	68
3.17	Evaluation of efficiency of reachability computation for IRL environment	69
3.18	Evaluation of efficiency of reachability computation for ARRL environment	69
3.19	Percentage non-visited reachable grid cells for IRL	71
3.20	Percentage non-visited reachable grid cells for ARRL	71
3.21	Progression of the avg. no. of search steps for reachability check for IRL environment	72
3.22	Progression of the avg. no. of search steps for reachability check for ARRL environment	73
4.1	Extracting single grid cell thick boundary contours	77
4.2	Filtering erroneous boundary cells	78
4.3	Illustration of filtering of erroneous thin lines in the map	79
4.4	Mapped-free-space boundary contour chaining	81
4.5	Illustration of seed cells neighboring boundary cell clusters	83
4.6	Selection of the neighboring boundary cells to initialize contour chaining. S is the seed cell. The red arrows indicate the chaining of two selected cells	85
4.7	Illustration of an example MX-Quadtree	87
4.8	Linking neighboring leaves and introduction of a corner list to MX-Quadtree. C_1, C_2, C_3 and C_4 are the corners of the two contours. . . .	88
4.9	Linked MX-Quadtree (L-MX-Quadtree)	89

LIST OF FIGURES

4.10	Maps of data-sets used for experimental validation	97
4.11	Comparison of execution times for frontier generation between SRFD and TRA methods with different map dimensions for IRL environment.	99
4.12	Comparison of execution times for frontier generation between SRFD and TRA methods with different map dimensions for ARRL environ- ment.	99
4.13	Execution time comparison for IRL environment	100
4.14	Execution time comparison for ARRL environment	100
4.15	Execution time comparison for CMU-NSH, FR101 and Seattle-R en- vironments	101
4.16	Comparison of accuracy in IRL environment	103
4.17	Comparison of accuracy in ARRL environment	103
4.18	Comparison of accuracy in CMU-NSH environment	104
4.19	Comparison of accuracy in FR101 environment	104
4.20	Comparison of accuracy in Seattle-R environment	104
4.21	Qualitative comparison of accuracy of generated frontiers in IRL en- vironment	105
4.22	Qualitative comparison of accuracy of frontiers generated in ARRL environment	106
4.23	Efficiency comparison of exploration algorithms with and without SRFD	107
5.1	Illustration of the traditional frontier exploration sending robots deep in to dead-ends	111
5.2	Illustration of basic parameters used during exploration task cancel- lation	112
5.3	Illustration of augmenting the traditional exploration strategy with the task cancellation step	113
5.4	Selecting the dimensions of the neighborhood of the intermediate target	115
5.5	The narrow passage with dead-end used to analyze task cancellation parameters	117
5.6	The open space used to analyze task cancellation parameters	117
5.7	Travel distance comparisons for single task cancellation in narrow passage with dead-end	118

LIST OF FIGURES

5.8	Travel distance comparisons for single task cancellation in Room environment	119
5.9	Environments used for simulated experiments	120
5.10	Qualitative comparison of traversed paths during exploration of the indoor environment	121
5.11	Comparison of avg. travel distance to explore the indoor environment	122
5.12	Comparison of avg. travel distance to explore the outdoor environment	122
5.13	Qualitative comparison of travel distances in a real indoor environment using P3AT robot	123
5.14	Avg. number of decision steps required for full exploration in indoor environment	124
5.15	Avg. number of decision steps required for full exploration in outdoor environment	124
6.1	Illustration of orientation changes of map features due to best particle change	130
6.2	Illustration of several intermediate steps during a directed exploration towards goal g	131
6.3	Estimating the shortest distance from frontier points to goal	132
6.4	Generation of convex hulls for the mapped areas in three successive steps of a navigation mission	134
6.5	Generation of the visibility graphs from the computed convex hulls .	135
6.6	Adding temporary vertices to the goal visibility graph to estimate shortest distance from frontier cells	136
6.7	Computing shortest distance for frontier cells that are not visible to the convex-hull's boundary	137
6.8	Illustration of the effect of task cancellation on early detection and recovery from dead-end situations.	138
6.9	Environments used during simulated experiments	141
6.10	Comparison between DEGON and RP results in outdoor environment	142
6.11	Comparison between DEGON and RP results in maze environment .	143
6.12	Comparison between DEGON and RP results in office environment .	144

LIST OF FIGURES

6.13 Comparison between DEGON and RP results in building-entrance environment	145
6.14 Illustration of stages of a directed exploration based navigation using a tracked robot (Jaguar) in a real environment.	145
7.1 Illustration of exploring an unbounded environment using a normal exploration strategy	149
7.2 Illustration of frontier cell updates for waiting time variance calculation	153
7.3 Illustration of intermediate target point selection for compact exploration at the start of the mission	159
7.4 Illustration of intermediate target point selection for compact exploration	160
7.5 Environments used for simulated experiments	163
7.6 Comparison of compactness ratio and explored ratio in bounding box for the single robot missions in the low obstacle density environment .	165
7.7 Comparison of compactness ratio and explored ratio in bounding box for the single robot missions in the high obstacle density environment	165
7.8 Comparison of compactness ratio and explored ratio in bounding box for the single robot missions in the environment with large barricaded region	166
7.9 Illustration of the generated map and the path of the robot using the compact exploration strategy in low obstacle environment and environment with the barricaded region	166
7.10 Comparison of compactness ratio and explored ratio in box for the utility discounting based multi robot missions in low obstacle density environment	167
7.11 Comparison of compactness ratio and explored ratio in bounding box for the utility discounting based multi robot missions in high obstacle density environment	168
7.12 Comparison of compactness ratio and explored ratio in bounding box for the utility discounting based multi robot missions in the environment with a barricaded region	168

LIST OF FIGURES

7.13	Qualitative comparison of the generated maps and the traversed paths of the robots using Normal and Compact exploration strategies . . .	169
7.14	Comparison of compactness ratio and explored ratio in bounding box for the utility discounting with re-positioning based multi robot missions in low obstacle density environment	169
7.15	Comparison of compactness ratio and explored ratio in bounding box for the utility discounting with re-positioning based multi robot missions in high obstacle density environment	170
7.16	Comparison of compactness ratio and explored ratio in bounding box for the utility discounting with re-positioning based multi robot missions in the environment with a barricaded region	170
7.17	Comparison of compactness of maps generated during exploration of the low obstacle density environment	171
7.18	Comparison of compactness of maps generated during exploration of the high obstacle density environment	171
7.19	Comparison of compactness of maps generated during exploration of the environment with barricaded region	172
A.1	Illustration of generated OBBs for cell sets	181
B.1	Information Gain of a Frontier Block	188
B.2	Influence on frontiers by a robot moving to a particular target point .	190

Chapter 1

Introduction

1.1 Motivation

Over the last several decades, research and development on mobile robotics has been expedited due to rapid improvements in computer and sensor technologies. These improvements have led robotics researchers to envision and develop advanced autonomy for mobile robotic systems that would ultimately replace humans or reduce the workload of humans operating in hazardous, uncomfortable environments. Making intelligent navigation decisions to completely sense a previously unknown environment and generate a map using its on-board sensors is one of the most important and interesting tasks expected of such an autonomous mobile robot. Planning and executing motion-and-sensing tasks for robots for map generation is popularly known as autonomous exploration. Autonomous exploration is fundamental in facilitating other useful autonomous behaviors of robots, such as autonomous navigation and autonomous tool manipulation, as many of these behaviors require complete or significant knowledge of the surrounding environment for optimal execution. Search and rescue [1, 2], planetary mapping[3] and underwater mapping [4] are some examples for missions requiring autonomous exploration capability.

For a robot to autonomously explore an unknown environment, it must sequentially sense the environment at new sensing locations until complete sensor coverage of the environment is achieved. At a given instance during an exploration mission, there are multiple candidate locations where the robot could perceive the environ-

1. INTRODUCTION

ment to expand the existing mapped area. Once the robot senses the environment through one of these candidate locations, a new set of candidate sensing locations appear in the map. This explosion of the candidate sensing space with each sensing action, coupled with the high uncertainties in the map information gain predictions for each of these sensing locations, makes finding an optimal sequence of sensing locations for complete exploration of unknown environments intractable[5]. Therefore, all the exploration strategies are reduced to finding the current best candidate sensing location as an intermediate target and moving to that target as the next task and repeating this process until complete mapping of the environment is achieved[6].

Many exploration approaches based on the above operating criteria have been examined in the literature and they mainly differ in the way the candidate sensing locations are generated and how these are evaluated to select the next intermediate target. Random generation[7, 8], extracting open-directions from topological maps[9, 10], extracting free-lines from poly-line maps[11] are some example strategies for generating candidate sensing locations. However, the most popular heuristic used to generate the set of candidate locations at each decision step, to find the next best sensing location (i.e. intermediate target) in 2D maps, is to extract cells in the boundary between the mapped free and unmapped cells from an Occupancy Grid Map representation of the environment[12]. These extracted boundary cells are referred to as *frontier* cells and the exploration strategy based on this heuristic is called Frontier Based Exploration[13]. Since its introduction, frontier based exploration has become a popular choice among researchers and practitioners in developing autonomous exploration strategies due to the relative ease of extracting frontier cells, planning safe paths and generating trajectories to selected intermediate targets using the occupancy grid maps. Fig. 1.1 illustrates an example of a 2D occupancy grid map of an environment and frontier cells extracted from the map.

Over the years, the various improvements examined and proposed for frontier based exploration has been centered on strategies selecting the best intermediate target that ultimately results in improved exploration efficiency in terms of the distance travelled and the number of sensing actions (i.e. cost of exploration). The proposed improvements include elaborate utility functions that quantify the desirability of frontier cells as candidate target points[14, 15, 16, 17, 18, 19, 20], clustering of frontier cells for better candidate target point representation[11, 21,

Figure 1.1: Illustration of frontier cells extracted from an occupancy grid map (read, blue and green colored). Note that not all boundary cells between free and unknown cells are extracted as frontier cells. Boundary cells that are sufficiently far away from obstacle cells (i.e. black colored cells) and only the ones that are reachable by the robot are extracted. Hence the boundary cells on the thin strip to the north east of robot are not extracted as frontier cells.

[22, 23, 24] and use of semantic information based intelligent boosting and lessening of utilities of candidate target points[25, 26]. A summary of these improvements are provided in Chap. 2.

Detailed evaluations of frontier based exploration efficiency on different conditions reveal that while the above examined approaches improve the efficiency of exploration missions significantly, they still suffer from the event based decision making paradigm[27, 28] where the decisions about the next intermediate target is made only after the robot reaches the current intermediate target. Since the updated map information is not used frequently, the robots are unable to re-evaluate their decisions on navigating to the selected intermediate target point which sometimes resulting in redundant motion thus degrading the performance of exploration missions[29]. In [28], it is reported that by increasing both the map update frequency and decision frequency, these redundant motions can be reduced to improve the total exploration efficiency. However, increasing the decision frequency in turn increases the computation required by the robot to generate frontier cell information and to execute the intermediate target selection step. This could quickly overwhelm a robot’s computational resources, especially for a robot with limited computing capability.

A compromise on decision frequency based on the available computing resources

1. INTRODUCTION

is suggested to overcome these issues[28]. However, with the processing of entire occupancy grid map at each map update step to extract frontiers, a bound on the computing demand cannot be provided as the map increases in size as the robot uncovers more area during exploration. Therefore, the optimal decision frequency is highly dependant on the dimensions of the mapped environment and the resolution of the map. Hence, with sufficiently large environments the decision frequency gets reduced, thus does not address the problem of redundant exploratory motions. Motivated by these issues, this thesis examines algorithms for efficient generation of frontier information, that support high frontier refreshing rates, and how these frequently available frontier information can be utilized to further improve frontier based exploration and related autonomous navigation missions.

1.2 Thesis Objective and Approach

As mentioned in the previous section, efficient generation and management of frontier information is vital for further improving the efficiency of frontier based exploration missions. Given that only a portion of the map is updated during a certain time window, the approach of processing of only the locally updated map information at each time window and incrementally updating a global database of frontier information with the processed data is considered for efficient frontier generation. However, it is also imperative that the proposed incremental algorithms maintain a very high level of accuracy for frontier identification, as the computational reduction should not come at the cost of accuracy. Therefore proposed incremental frontier extraction algorithms must ensure the non-extraction of phantom frontiers[30]. Phantom frontiers are boundary cells, between free and unknown area, that are unreachable. Fig. 1.2 illustrates the occurrence of phantom frontier cells in an occupancy grid map due to mapping errors.

In traditional frontier extraction algorithms where the entire occupancy grid is processed, phantom frontier cells are filtered out by computing their reachability by the robot. As this thesis is concerned about incremental algorithms for efficient frontier generation, computing the reachability of the grid cells in an incremental fashion is examined as the initial step of extracting only valid frontier cells. Since only local map data is processed at each step, development of an in-memory

Figure 1.2: Illustration of phantom frontiers generated due to mapping errors. The extracted frontier cells are marked in red. All these marked frontier cells are safe for the robot (i.e. avoids collisions with obstacles when placed on the cell) but unreachable by the robot from its current position due to mapping through small openings or due to errors in mapping. Therefore the reachability of the cells must be considered when extracting frontier cells.

database that maintains all the global boundary cells between free and unknown cells is considered as the next step, and these two steps are combined to extract only the reachable boundary cells as valid frontier cells. Subsequently the use of these frequently available frontier cell information due to their efficient computation is examined for improving frontier based exploration and related autonomous navigation missions.

1.3 Contributions

The main contributions of the thesis are summarized below

- A computationally efficient incremental strategy for frontier extraction and management with high frontier detection accuracy. The proposed strategy treats the incremental frontier extraction as solving the two sub-problems (a) incrementally updating a database of contours of boundary cells between free and unknown area and (b) incrementally identifying the reachability of the contours in the database, both using only locally updated map data. Finally, only the reachable boundary contours are extracted and reported as frontiers at each map update step.

1. INTRODUCTION

- Introduction of a new exploration task cancellation step to frontier based exploration algorithms using the frequently available frontier information. The task cancellation step is formalized as a probabilistic decision step where it decides whether further motion on the planned path to the next intermediate target is desirable or not based on the most recent frontier information. The introduction of this decision step reduces the total distance traveled during missions to improve exploration efficiency.
- A frontier exploration based, alternative autonomous navigation algorithm for moving a robot to a given destination through an unknown environment. The exploration task cancellation using the frequently available frontier information is employed to quickly detect and avoid dead-end situations resulting in identical travel distance performance and a superior re-planning efficiency compared to dynamic motion planning algorithms.
- A frontier based exploration strategy that generates compact maps during exploration of unbounded environments. Frequently available frontier information is utilized to update the waiting times of frontier cells efficiently, which is the time elapsed between the extraction of the frontier to current time step. A minimizing of waiting time variance based utility function is introduced to replace the traditional utility function to generate exploratory motions that results in compact maps for both single and multi-robot missions.

1.4 Thesis organization

Chapter 2 details the basic notations and definitions used throughout the thesis to define frontier based exploration and related navigation tasks. Contours of the boundary between free and unknown cells and frontier cells are carefully defined to aid their efficient generation and management as reported in the subsequent chapters. The second half of this chapter provides a detailed summary of the frontier based exploration strategy. The traditional frontier extraction algorithm which is used during the efficiency and accuracy comparison studies is described in detail for reference and to highlight the challenges

involved. This is followed by a summary of different use of frontier cell information to increase the efficiency of frontier based exploration strategies. The chapter concludes with a review of existing approaches to efficiently generate frontier cell information that are also used during comparison studies.

Chapter 3 describes the efficient identification of the safe and reachable cells using incremental updates on an evolving occupancy grid map. Incremental updating of the obstacle inflation information, using a brushfire algorithm restricted on robot's radius, is presented. Then the reachability computation of grid cells is formalized as a graph connectivity problem. The abstraction of the occupancy grid map as a graph of interconnected safe-patches is provided, where a safe-patch is defined as a connected set of safe grid cells generated during a single map update step. A collection of incremental update procedures to maintain the connectivity of the graph of safe-patches using spanning trees is formalized and the associated algorithms are provided. The resultant incremental safe and reachability extraction algorithms are compared with the traditional reachability computation procedure detailed in Chapter 2 using real world data gathered in two environments. The comparisons indicate that the proposed incremental approach provides a bound on the number of processed cells, hence the processing time, compared to the traditional approach.

Chapter 4 combines the algorithms presented in Chapter 3 to generate an incremental frontier extraction and management strategy referred to as Safe and Reachable Frontier Detector (SRFD). The chapter begins by formulating the incremental processing algorithms to extract the contours of the boundary between the free and unknown cells as a grid-cell-chain with two corners. Then an MX-Quadtree based data structure, Linked-MX-Quadtree (L-MX-Quadtree), is proposed as an in-memory database to efficiently store and update this contour information. A detailed analysis of the computational complexity of the update operations on the L-MX-Quadtree reveals that the update operations can be executed efficiently to support high frequency frontier cell refreshing. The resulting SRFD strategy is compared for efficiency and accuracy with the traditional and several incremental frontier extraction approaches using real

1. INTRODUCTION

world exploration data. The analysis of the comparisons reveal that the proposed approach is capable of efficiently generating frontier cells with superior computational efficiency and frontier detection accuracy compared to state of the art incremental methods in the literature.

Chapter 5 provides the first application of fast frontier refreshing rates provided by the efficient frontier extraction and management strategy, SRFD, developed in Chapters 3 and 4. A probabilistic decision step is formalized that provides the ability for the robot to decide on the remaining motion towards its current intermediate target using the frequently available frontier information. This decision step is used to augment the traditional frontier based exploration strategy to avoid robot executing redundant motions. Experimental evaluations conducted in both simulated and real environments reveal that augmenting the traditional frontier based exploration strategy to include an exploration task cancellation step improves the overall travel distance efficiency of robots.

Chapter 6 utilizes the exploration task cancellation step introduced in Chapter 5 to devise a navigation algorithm, based on frontier based exploration, to move a robot to a goal position through an unknown environment. The navigation algorithm is proposed as an alternative to traditional dynamic planning algorithms which require significant changes to support particle filter based mapping systems due to map orientation changes. Navigation to the goal is formulated as frontier based exploration directed towards the goal and the utility for each frontier cell is computed to be the distance estimate to the goal from the robot's current position through the frontier grid cell. The distances from frontier cells to the goal is estimated by computing a visibility graph through the unknown area. The proposed directed exploration algorithm is evaluated on four different simulated environments together with dynamic planning algorithms and is shown to perform comparably on distances travelled to reach goals while requiring on average less than a third of the re-planning steps used by dynamic planning algorithms.

Chapter 7 describes the development of a new utility function based on a new way of looking at the efficiency of exploration missions in unbounded environments, where the compactness of the generated map is considered for efficiency. The efficient frontier management algorithms developed in Chapters 3 and 4 are used to efficiently update the waiting time property of frontier cells where the waiting time of a frontier cell is the time elapsed since the cell being identified as a frontier. The new utility function is based on the idea of minimizing the waiting time variance of frontier cells thus sending robots to visit frontier cells in a balanced way resulting in a compact map. Oriented bounding box based compactness measure is proposed and the proposed utility function is evaluated for its ability to explore environments in a compact way in both single and multiple robot missions. The evaluations reveal that the proposed utility function guides the robot(s) to explore environments in a balanced way resulting in compact maps.

Appendices briefly describe two earlier research works of the author which are later used for evaluating the proposed algorithms in this thesis. In Appendix A, Oriented bounding box based Frontier Detector (OFD)[31] is detailed which is also an incremental frontier detection algorithm. OFD is used as one of the algorithms used in Chapter 4 to compare the efficiency and accuracy of proposed SRFD strategy. Appendix B details a multi-robot exploration strategy targeted for exploring unstructured environments where semantic information cannot be used to improve the exploration efficiency[32]. This algorithm is used in Chapter 7 to illustrate the effect of the proposed utility function in multi-robot exploration algorithms on generating compact maps in unstructured environments.

Chapter 2

Background

2.1 Notations and definitions

The notations and definitions of the basic data structures used to describe the frontier based exploration algorithms are provided below for ease of reference throughout this thesis.

2.1.1 Occupancy grid map and related data structures

Frontier based exploration which is introduced in [13] employs an Occupancy Grid Map [12] representation of the environment. An occupancy grid map estimates the actual map of the environment at each step k by decomposing the map into individual grid cells and estimating the probability of occupancy of obstacles in each of these grid cells [14]. Considering M_k to be the map of the environment at step k , the set of grid cells of the occupancy grid map at the k^{th} update step is defined as $\mathcal{M}_k$.

Consider that the posterior of the map, given robot pose $x_{1:k}$ and sensor data $z_{1:k}$ up-to step k , is given by,

$$p(M_k \mid z_{1:k}, x_{1:k}) \tag{2.1}$$

Then for an occupancy grid map, the posterior of each grid cell $c \in \mathcal{M}_k$ is given by $p(c \mid z_{1:k}, x_{1:k})$ where the posterior of the map is estimated using the posteriors

2. BACKGROUND

of the grid cells as,

$$p(M_k \mid z_{1:k}, x_{1:k}) = \prod_{c \in \mathcal{M}_k} p(c \mid z_{1:k}, x_{1:k}) \quad (2.2)$$

These computed probabilities are thresholded to classify each grid cell to one of the three classes (a) FREE - cell is not part of an obstacle (b) OBS - cell is part of an obstacle and (c) UNKNOWN - cell is unmapped. While many different thresholding values can be used for this classification, the simple classification of any cell with an occupancy probability of less than 0.5 being FREE and probability greater than 0.5 being OBS and rest being UNKNOWN is used following the standard employed in [5].

Definition 1. Given a grid cell $c \in \mathcal{M}_k$, $\text{occ}(c)$ returns the class to which the cell belongs to, OBS for occupancy of an obstacle, FREE for a free cell and UNKNOWN for unmapped.

Definition 2. The set of obstacle cells after the k^{th} update step is defined as, $\mathcal{O}_k = \{c \in \mathcal{M}_k \text{ s.t. } \text{occ}(c) = \text{OBS} \}$

Definition 3. The set of free cells after k^{th} step is defined as $\mathcal{F}_k = \{c \in \mathcal{M}_k \text{ s.t. } \text{occ}(c) = \text{FREE} \}$

Frontier cell extraction from the set $\mathcal{M}_k$ requires deciding the safe set of cells. Unsafe cells that lead the robot to collisions with obstacles must be ignored from being considered for navigation. This requires classifying grid cells that should not be within the footprint of the robot.

In addition, the paths planned after a frontier cell is selected as the next intermediate target must also be safe for the robot to travel. However, the traditional algorithms used for path planning are point/grid cells based where the robot is considered to be represented by a single point or a grid cell. Since robot footprints are typically larger than a single cell area, direct planning on the cell set $\mathcal{M}_k$ cannot guarantee the output of safe paths.

Both of these problems are resolved by transforming the obstacle cells in $\mathcal{O}_k$ in such a way that only a single cell check is sufficient to decide safeness of a grid cell.

This transformation is done by computing the Minkowski sum between the obstacle cells and the robot's footprint cells [33]. In order to simplify this calculation, the footprint of the robot is defined to be a circle with a radius R_r in grid cell units, hence the cells within the defined circle contains all the possible footprints of the robot, corresponding to all different poses. The radius R_r is considered to be the radius of the footprint of the robot. Therefore, the resulting minkowski sum *inflates* all the obstacles to include grid cells that are within R_r distance to its boundary. Hence, the set of inflated obstacle cells is defined as follows. Fig. 2.1(a) illustrates a simple example of inflating an obstacle using robot's radius and Fig. 2.1(b) illustrates the resulting safe area by removing the inflated obstacle cells from the free space.

Figure 2.1: (a) The circular approximation of the robot's footprint is used to inflate the obstacle cells. (b) Subtracting the inflated cells from the free cells generate the set of safe cells $\mathcal{S}_k$

Definition 4. The set of inflated obstacle cells is defined as $\mathcal{J}_k = \{c \in \mathcal{M}_k \text{ s.t. } \exists o \in \mathcal{O}_k \text{ where } \text{eucl}(c, o) \leq R_r\}$ where $\text{eucl}(c, o)$ is the Euclidean distance between the two grid cells and R_r is the radius of the robot both given in grid cell units.

Definition 5. The set of safe grid cells, where the center of the robot could be placed without any of the robot's footprint cells overlap with obstacle cells is defined as, $\mathcal{S}_k = \{c \in \mathcal{F}_k - \mathcal{J}_k\}$

The neighboring grid cells for a given cell c is defined based on both 8-connectivity and 4-connectivity.

2. BACKGROUND

Definition 6. The set of 4-connected neighboring grid cells of cell c is defined as $\mathcal{N}_4(c) = \{c' \in \mathcal{M}_k \mid \text{eucl}(c, c') = 1\}$

Definition 7. The set of 8-connected neighboring grid cells of cell c is defined as $\mathcal{N}_8(c) = \{c' \in \mathcal{M}_k \mid \text{eucl}(c, c') \leq \sqrt{2}\}$

Definition 8. The set $\mathcal{R}_k(c) = \{c \in \{\mathcal{N}_8(c) \cap \mathcal{S}_k\}\}$ contains the neighboring cells of c , to where the robot could be moved from c without colliding with any obstacle.

Using the above definitions, a graph can be defined to represent the reachability relationships between “safe” grid cells. This graph representation will be later used to define frontier cells and to incrementally calculate the reachability information.

Definition 9. $G'_k = \{V'_k, E'_k\}$ is a graph where each vertex $v \in V'_k$ represents a grid cell in the set $\mathcal{S}_k$. Given, $c(u)$ being the grid cell associated with the vertex u , an edge $e = (u, v) \in E'_k$ if and only if $c(u) \in \mathcal{R}_k(c(v))$ and $c(v) \in \mathcal{R}_k(c(u))$.

Definition 10. The function $\mathcal{V}' : \mathcal{M}_k \mapsto V'_k$ provides the mapping between a grid cell and its associated node in the graph G'_k .

The above graph abstraction of the safe cell set can be used to define traversability of the cells by the robot as follows.

Definition 11. A set $C, C \subseteq \mathcal{S}_k$, of cells is *traversable* by the robot if $\forall u, v \in C$, $\exists$ a path p in G'_k connecting $\mathcal{V}'(u)$ and $\mathcal{V}'(v)$. It directly follows that the set of cells represented by each maximal connected sub-graph (i.e. connected component) in G'_k is traversable by the robot.

Definition 12. The set of connected components of G'_k is defined as $\mathcal{C}_k$.

Definition 13. The set of grid cells in $\mathcal{S}_k$ associated with a connected component $C \in \mathcal{C}_k$ is denoted by $A'(C)$ using the mapping $A' : \mathcal{C}_k \mapsto 2^{\mathcal{S}_k}$.

2.1.2 Traversability, Reachability and Frontier cells

Based on the above definitions, the mapped-free-space boundary contours and frontiers are defined below which will be used in chapters 3 and 4. It directly follows that the set of cells represented by each maximal connected sub-graph (i.e. connected component) in G'_k is traversable by the robot. The nodes corresponding to the grid cells of robot's footprint belong to only one of these connected components, denoted by C_k^* . Since the robot is capable of traversing the grid cells associated with the connected component, all the cells associated with C_k^* , $A'(C_k^*)$, are considered *reachable* by the robot. While the rest of the grid cells associated with other connected components are traversable, they are not reachable by the robot. This distinction between reachability and traversability is used to extract valid frontiers as follows.

Definition 14. The set of mapped-free-space-boundary cells (i.e. boundary cells between free and unknown cells) at step k is defined as $\mathcal{B}_k = \{c : c \in \mathcal{S}_k \text{ s.t. } \exists c' \in \mathcal{N}_4(c) \text{ where } \text{occ}(c') = \text{UNKNOWN}\}$.

Definition 15. The set of frontier cells present in the map is defined as $\Gamma_k = \{c : c \in \{\mathcal{B}_k \cap A'(C_k^*)\}\}$.

Therefore, according to the above definitions, frontier cells are mapped-free-space boundary cells that are reachable by the robot from its current location. This definition of mapped-free-space boundary cells and frontier cells are used throughout the thesis in formulating and developing the efficient incremental frontier extraction strategy. It is also important to note that 4-connectivity of neighboring cells is used in Def. 14 to define mapped-free-space boundary cells. This results in identification of single cell thick boundary contours and makes their extraction a simple cell chaining operation as described in Sec. 4.1.2.

2.1.3 Map update step

Since the main thesis of the work reported in this dissertation is about incremental updating of map information for efficient frontier cell extraction, it is important to have a clear understanding about the stages at which occupancy grid maps are

2. BACKGROUND

updated by robots and how and when these maps are processed by navigation algorithms. In general, continuous mapping modules used by robotic navigation systems continue to accumulate the received mapping sensor information at a higher frequency. A Laser Range Finder (LRF) for example usually operates at a frequency of around 20-30Hz. All of these sensor information are used to update the occupancy grid map. However, the rate at which the map and frontier information should be made available from the mapping module to the rest of the navigation system is highly dependant on the speed of operation of the robot (e.g. decision frequency) and the nature of the operating environment. Therefore, all of the mapping modules provide a way to alter the frequency of publishing the updated map information for other modules in the navigation system [34, 35]. Hence, the map update frequency experienced by the rest of navigation modules is the map publishing frequency. Therefore, throughout the rest of this article, *map update step* refers to the instance at which an updated map is made available to the rest of navigation system. This is visually illustrated in Fig. 2.2.

Figure 2.2: Relationship between map update step and sensor readings.

The proposed incremental algorithms operate on a batch of intermediate sensor readings between two map update steps. Therefore, all of the changes to the occupancy grid map between two map update steps are accumulated into intermediate data structures, which are defined next.

All the map grid cells whose occupancy class (i.e. $occ(c)$) is changed between two map update steps $k - 1$ and k are stored in a set denoted by U_k for all k . All the grid cells that were newly marked as obstacles are stored in the set $\mathcal{O}_k^{new}$. Mapping errors induced due to sensor noise and position estimation errors, results in certain grid cells being incorrectly marked as obstacles. And some of these grid cells are later cleared by more accurate sensor information resulting in disappearance of these obstacle cells in subsequent map update step. The set of such disappeared obstacle cells between two map update steps $k - 1$ and k are denoted by $\mathcal{O}_k^{del}$ for all k .

2.2 Frontier based Exploration

2.2.1 Introduction to frontier based exploration

Exploring an unknown environment requires a robot to travel to regions that are yet to be mapped. Therefore, for a robot to completely explore an unknown environment, it must sequentially sense the environment at new sensing locations until complete sensor coverage of the environment is achieved. As mentioned in the introductory chapter, generating the optimal plan for completely mapping the environment is NP-Hard and therefore all exploration algorithms are reduced to greedy mapping algorithms where only the next sensing action is selected and executed based on the available map data. The frontier based exploration strategy has become the baseline approach for developing exploration missions due to its simplicity and scalability and its most basic form is summarized in Alg. 1.

Algorithm 1 Frontier Based Exploration

```
1:  $k \leftarrow 1$ 
2: while  $|\Gamma_k| > 0$  do
3:    $m^* = \arg \max_{\gamma \in \Gamma_k} J(\gamma)$ 
4:   Move robot to new intermediate target point  $m^*$ 
5:   Update map
6:    $k \leftarrow k + 1$ 
7: end while
```

The exploration mission continues until there are no reachable safe frontier cells according to the check in line. 2. In line. 3 all the frontier cells are evaluated based on a defined utility function J and the best frontier cell, m^* , is selected as the next intermediate target of the robot and the robot is moved to this target in a planned path through an already explored region in the environment. Once the robot reaches the intermediate target point m^* , the occupancy grid map and the set of frontiers is updated and the process continues from the start until the termination condition is met. Considering the fact that the robot continues to unknown areas until there are valid reachable frontier cells, it can be trivially understood that the frontier based exploration strategy results in complete mapping of the the environment. Figs. 2.3-2.6 illustrate several stages of an exploration mission. The frontier cells extracted from the occupancy grid maps are shown in colored (red, green and blue) contours. The footprint of the robot is depicted by the blue box. At each step, the selected

2. BACKGROUND

intermediate target from the available frontier cells is indicated by the arrow and the planned path to reach this target is indicated in dotted line connecting the robot to the target.

Figure 2.3: Frontier based exploration example - step 1

Figure 2.4: Frontier based exploration example - step 2

2.2.2 Traditional frontier cell extraction algorithm

At each decision step of the exploration algorithm, the current set of frontier cells Γ_k is extracted. The traditional approach employed by current frontier based exploration strategies to extract frontier cells is illustrated in Alg. 2. This is later used as the baseline approach in comparing the performance of the incremental frontier extraction method proposed in Chap. 3 and 4.

Fig. 2.7 illustrates the outcome of some of the major steps involved in the traditional frontier extraction strategy. The occupancy grid map is first processed to extract the set of cells marked as obstacles, $\mathcal{O}_k$. Then the minkowski sum of the obstacles with the robot's footprint is computed to inflate the obstacle cells to ensure

Figure 2.5: Frontier based exploration example - step 3

Figure 2.6: Frontier based exploration example - step 4

that single cell checks can be performed to check for safety of cells and to perform traditional path planning algorithms. This operation on the grid can be executed using a flood-fill[36] operation restricted by the robot's radius R_r . The pink colored grid cell in Fig. 2.7(b) depict the resultant inflated cell set J_k after such operation. Next, the reachable set of grid cells from the free cells is extracted. This can be conducted either by a another flood-fill operation starting from the robot's position cell or by computing the shortest distance to all the safe cells from robot's position, considering the unknown cells are unreachable, and extracting cells with only finite distance values. The distance transform[37], Lee's algorithm[38] or Dijkstra's shortest path algorithm[39] can be used to compute the shortest paths. Finally,

2. BACKGROUND

Algorithm 2 Extract frontier cells at step k

- 1: Extract the set of obstacle cells $\mathcal{O}_k$
 - 2: Inflate the obstacle cells to generate $\mathcal{I}_k$
 - 3: Compute the reachable set of cells $A'(C_k^*)$
 - 4: Extract the frontier cells Γ_k by processing $A'(C_k^*)$
-

the frontier cells are extracted from the boundary cells that are also reachable, by processing the grid cell set $A'(C_k^*)$.

Considering that the extraction of obstacle cells require processing the entire occupancy grid map and extracting the frontier cells require processing the entire set $A'(C_k^*)$ of cells, which gets expanded as the robot explores more area, the computational complexity of the frontier extraction algorithm is dependant on the dimensions of the map and a bound cannot be provided on the number of processed cells even for a given map size. As mentioned earlier, this results in exploration algorithms not being able to use a guaranteed decision frequency to improve the efficiency of missions.

2.2.3 Calculation of utilities of frontier cells

As described in line. 3 of Alg. 1, the next intermediate target is selected by evaluating the frontier cells according their individual utility for sensing action. Therefore, different utility functions result in selecting different intermediate targets during exploration missions and this inevitably leads to different sensing motions for robots. Hence, this section reviews the different utility functions that are being used and proposed in the literature.

2.2.3.1 Cost of reaching frontier cell

The most basic utility function used by exploration algorithms is the travel cost function. The travel cost in terms of the distance to the frontier cell or the time to reach the frontier cell from robot's current position is used.

$$J(\gamma) = -Cost(\gamma) \tag{2.3}$$

$Cost(\gamma)$ is the cost of reaching frontier cell γ from robot's current position.

Figure 2.7: Illustration of major operations of the traditional frontier extraction algorithm. (a) the occupancy grid map and the robot's footprint (b) inflating the obstacles (c) identifying cells that are reachable from robot's current position (d) the extracted frontiers are marked in red, blue and green colors.

Yamauchi's initial work on exploration [13] used this simple utility function. This utility function makes the robot select the closest frontier cell as its next intermediate target. While being simple, it suffers from needing more sensing actions to complete exploration missions as sensing at some of the closest frontier cells may add only small amount of new map information.

2.2.3.2 Gain by reaching frontier cell

Another approach is to set the gain in the map information by reaching the frontier cell as the utility value. Considering R_{max} to be the maximum sensor rang, this requires accessing for each frontier cell γ , the set of cells that falls inside the circle drawn at the frontier cell with the radius R_{max} , denoted by $\mathcal{N}_R(\gamma)$. Thrun et al. [14] discusses three ways of computing this map information gain value.

2. BACKGROUND

Entropy Each grid cell $c \in \mathcal{M}_k$ contains a probability of occupancy $p(c)$. This probability value can be used to compute its entropy according to eq. 2.4

$$H_p(c) = -p(c) \log(p(c)) - (1 - p(c)) \log(1 - p(c)) \quad (2.4)$$

According to eq. 2.4, the entropy for an explored grid cell, either occupied or free, is lower compared to the entropy computed for unexplored grid cells. Then the information gain, $I(\gamma)$, using the above equation can be computed as the summation of the entropies of all the grid cells that fall within R_{max} radius from the frontier cell in question as given below [15, 16].

$$J(\gamma) = I(\gamma) = - \sum_{c \in \mathcal{N}_R(\gamma)} H_p(c) \quad (2.5)$$

This directly translates to the requirement of robot preferring frontier grid cells with neighboring regions in the map with large number of unknown cells.

Expected information gain Entropy only measures the current information in the map and it does not specify the information that a robot would gain using its sensors by visiting this grid cell. The calculation of the expected information gain takes into account the probability of detection of occupancy by a robot's sensor [17, 18, 19]. The sensor can falsely detect unoccupied cells as occupied and occupied as unoccupied. These two false detections give rise to two new probabilities of detections, p^+ and p^- . These two new detection probabilities generate two occupancy update posteriors, and two corresponding entropy equations $H_{p'}^+$ and $H_{p'}^-$ for detecting occupied and unoccupied respectively for each grid cell. p' corresponds to the respective posterior generated. Therefore, the expected entropy for a grid cell c after the sensing operation is given by,

$$E[H_{p'}(c)] = p^+ H_{p'}^+(c) + p^- H_{p'}^-(c) \quad (2.6)$$

Thus the expected information gain for a cell c after carrying out the sensing operation becomes the difference between the previous entropy and the entropy after detection operation.

$$E[\text{Information Gain}(c)] = H_p(c) - E[H_{p'}(c)] \quad (2.7)$$

Therefore, the total information gain (and the utility) for the robot is computed as,

$$J(\gamma) = I(\gamma) = \sum_{c \in N_R(\gamma)} (H_p(c) - E[H_{p'}(c)]) \quad (2.8)$$

Evaluations given in [40] establish that the expected information gain (i.e. mutual information) based utility is superior to the entropy based method in representing next best places to conduct the exploratory sensing operation.

Binary gain This is a crude thresholding operation where all grid cells whose occupancy values are changed from the initial 0.5 probability are marked as explored and the rest are marked as unexplored. Grid cells marked as unexplored get gain value of 1 while the values for grid cells marked as explored are set to 0. Then the information gain can be computed by simply summing up the number of grid cells with the gain value 1. Simmons et al. [20] computes the information gain according to this method where the number of unknown grid cells within the robot’s sensor footprint is counted. In [11], a ray-sweeping method is employed to better estimate the robot’s usable footprint at the given frontier cell location using the currently available map. Then similar to [20] the number of unknown grid cells within the identified sensor footprint is reported as the information gain.

The gain calculation method proposed in [41] is an extension of the ray-sweeping method proposed in [11]. This method uses two different grid maps to represent the environment. One map is generated by integrating the raw range sensor data while the other map is generated by limiting the maximum range the sensor data through range thresholding. With range sensors with limited angular resolution, the sensor rays tend to spread over with considerable arc spaces at long range values. This reduces the accuracy of the map at large range values. Therefore, thresholding range values after a certain value,

2. BACKGROUND

provides accuracy control over grid maps and provides a conservative representation of the environment. The first map, generated from raw range data, can be then used to provide close to realistic estimates of the information gain when the ray-sweeping method is used.

While the entropy and expected information gain based methods are more theoretically intensive, evaluations conducted in [42] and [20] indicate that there is no significant difference to the robot’s exploratory behavior by choosing the unknown cell counting method over the previous two approaches. Since the latter approach is less computationally intensive, it has been used extensively in computing the information gain. The evaluations also reveal that using the information gain as the utility reduces the number of sensing locations the robot must navigate to completely explore the environment. However, this utility function does not consider the travel cost and this leads to robot moving to sensing locations that are far away from its current position and resulting in many back and forth motions during the latter stages of the mission. This ultimately results in a larger travel distance for the robot which is inefficient.

2.2.3.3 Trade-off between information gain and cost

Combining information gain and travel cost into a single utility function provides the best of the both worlds in terms of reducing the number of sensing locations and reducing the total travel distance during an exploration mission [42]. The two components are combined linearly to generate the utility function as shown below.

$$J(\gamma) = \alpha I(\gamma) - \beta Cost(\gamma) \quad (2.9)$$

The two variables α and β function as weight parameters and can be used to change the importance of one component over the other and is employed by many exploration strategies as the base of its utility function [5, 14, 43, 44]. Increasing α relative to β allows robots to seek frontier cells that provide more information gain while the opposite allows the robots to seek closer frontier cells as their next intermediate targets. The utility function $A(\gamma) \exp(-\lambda L(\gamma))$ proposed in [11] also provides the same trade-off between information gain and travel cost and has also

been popularly used [42, 45, 46] replacing the utility function in eq. 2.9. $A(\gamma)$ is the information gain and $L(\gamma)$ is the travel cost. Reducing the value of λ decreases the effect of the cost, hence leads robots to select more informative frontier cells while increasing the λ value increases the effect of the cost on the utility and makes the robot select nearby frontier cells.

2.2.4 Improving efficiency of frontier based exploration

While changing the parameters of the basic utility function provides some form of control over the behavior of the robot, the number of frontier cells used and how they are used during evaluation steps for selection of intermediate targets also affects the efficiency of exploration missions and is examined in the literature to further improve the efficiency of exploration missions.

Different frontier based exploration strategies use extracted frontier cell information in different ways. A large collection of exploration strategies use the frontier cell information as it is and evaluate cells individually [5, 20, 43, 44, 47, 48, 49, 50, 51]. For these strategies the efficiency is dictated solely by the selection of the utility function, as the information about the spatial arrangement of the frontier cells are not considered during evaluation. In addition, evaluating each and every frontier cell in fact increases the computational burden of the robot. Considering the standard utility functions reported in Sec. 2.2.3, computing the utility value for each frontier cell requires estimating the information gain parameter and this involves processing all the neighboring cells $\mathcal{N}_R(\gamma)$ for each frontier cell γ . As the robot continues to map the environment during an exploration mission, it discovers/generates more frontier cells (especially in open environments) resulting in an increase in computational requirement. The following subsections provide a brief overview of some of the approaches examined in the literature in reducing this computational requirement and other approaches of candidate point representations for improving exploration efficiency.

2.2.4.1 Frontier cell clustering based improvements

Yamauchi's initial work [13] grouped nearby frontier cells and selected only the groups that satisfy a certain size threshold as *frontier regions* to be considered

2. BACKGROUND

for exploration. In addition to reducing the number of candidate target points evaluated during the decision step, the introduction of a cluster size threshold filters out isolated frontier cells which occur due to mapping errors resulting in further improvement in the computational efficiency of intermediate target selection step. Such isolated frontier cells generally occur near obstacles due to specular reflection of the sensor rays or due to other noise in the sensor. Therefore, moving to these isolated frontier cells do not add new map information and could potentially lead the robot to a stuck situation. Filtering out these isolated frontier cells thus improves the efficiency of exploration missions by reducing the total number of sensing locations and the travel distance to completely map the environment.

Exploration strategy proposed in [21] follows the same approach and clusters adjacent frontier cells together for evaluation as a single entity. While the strategy proposed in [11] does not specifically use an occupancy grid map representation of the environment, its use of the free edges is similar to extraction of frontiers from occupancy grid maps. A free edge can be considered a connected contour of frontier cells (i.e. frontier contour). In [11] the number of candidate target points are sampled from each such free edge according to its length and is similar to clustering frontier cells with size thresholding and provides the same improvement to exploration missions. Additionally the length information of frontier contours can be directly used to replace the information gain components of the utility functions further improving the computational complexity of the candidate target point evaluation step.

Faigl et al. [22] proposed the concept of Representatives of Free Edges (RPE) where the frontier contours are represented by a set of representative cells. The representatives are means of n_r number of clusters for a specific frontier contour i denoted by $\Gamma_{k,i}$. n_r is computed using R_{max} as $n_r = 1 + \lfloor \frac{|\Gamma_{k,i}|}{1.8R_{max}} + 0.5 \rfloor$. These computed mean points can be used as candidate target points during evaluation. In [23], a polar histogram of the frontier cells centered on the robot's current position is generated. This histogram is used to generate a set of clusters in the polar histogram where the size of each cluster indicates a measure of the desirability of the corresponding "direction" for exploration. The centroids of these clusters can then be used as candidate target points. K-Means based frontier clustering[52], Quadtree based clustering[24] and frontier blocks[32] are alternative clustering techniques that

are proposed in the literature to improve the exploration efficiency, especially in multi-robot missions.

2.2.4.2 Semantic information based improvements

While clustering based methods reduce the number of candidate target points used for evaluation and reduces the travel cost by preventing robots from selecting isolated frontier cells, the use of semantic labels extracted from already mapped features in the occupancy grid map is capable of selecting intermediate target points more intelligently, to further improve the exploration efficiency of missions.

An early attempt of this approach is described by Simmons et al. [20] where it is reported that during exploring indoor environments, sometimes robots come out of rooms before completely exploring it due to the remaining information gain of the frontier cells inside the room being lesser compared to gains of frontier cells outside it. This behavior in turn requires robots to reach the same room multiple times to completely explore it, thus increasing the travel distance and reducing the efficiency of the mission. They provided a heuristic solution where the utility of the frontier cells that falls within R_{max} are boosted, by dividing it by a *hysteresis value* (a constant between 0 and 1), forcing the robots to completely map a room before coming out. However, further experiments conducted by the author of the thesis using relatively smaller sensor ranges compared to room dimensions, indicated that boosting the utility using the hysteresis ratio does not always make the robot fully explore the room before it moves out reducing the exploration efficiency.

Extracting semantic labels from the available occupancy grid map information has been examined in the literature to support exploration missions[53, 54, 55, 56]. The knowledge of semantic labels of the map features allows the partitioning of the available candidate target points so that it allows robot to intelligently boost or reduce the utility of candidate target points in the same partition. This ensures that robots do not make multiple trips to the same region during exploration, thus improving the efficiency. These generated partitions are also used in multi-robot exploration missions to better coordinate robots for efficient exploration[25, 26].

2. BACKGROUND

2.2.4.3 Beyond greedy decision making

The greedy decision making approach used by many of the frontier based exploration strategies discard any spatial correlation among candidate target points during the decision making step. This sometimes results in robots visiting the available candidate target points in a sub-optimal motion sequence, thus resulting in larger travel distances to complete exploration missions. Kulich et al. [46] proposes an improvement to the frontier based exploration strategy where the decisions are made based on a candidate tour of frontier cells rather than on a single candidate target point. The evaluation of tours introduces the spatial correlations among the frontier cells, consisting the tours, thus results in a better (i.e. shorter) motion sequence for the robot during exploration missions. A similar approach of evaluating tours of frontier cells rather than single candidate target points is examined in [57]. Again the spatial correlations among the different frontier cells are captured by the tours, hence more efficient exploratory motions can be planned that ultimately results in efficient exploration missions.

2.2.5 Extending frontier based exploration for multi-robot systems

The single robot frontier based exploration strategy has been extended to support multiple robot systems. In all of these strategies, the frontier cells or their representations, according to the different methods summarized in Sec. 2.2.4.1 and 2.2.4.2, are assigned to different robots using utility functions that are identical or derived from the ones reviewed earlier. These multi robot frontier based exploration strategies, apart from being operated as centralized or decentralized strategies, differ mainly in the way the robots are spatially coordinated [5, 20, 24, 26, 44, 58], how the connectivity among robots are maintained or recovered [47, 59, 60, 61] and on how the starting poses' of robot's are estimated [21].

2.2.6 Exploration and SLAM

At each step of the exploration mission, the robot's pose is required to update the occupancy grid map information, according to eq. 2.2. While high accuracy GPS

2.3 Efficient approaches for frontier cell extraction

(Global Positioning System) and IMU (Inertial Measurement Unit) sensors provide absolute positioning information for robots, Simultaneous Localization and Mapping (SLAM) approaches [62] are proposed to concurrently estimate the robot’s pose and environment map when such absolute positioning information is unavailable. A large body of research work that spans over two decades have addressed the SLAM problem in great detail. This has resulted in the availability of several accurate real-time positioning systems for mobile robots [63, 64, 65]. Therefore, the work presented in this thesis assumes the availability of a positioning system for the robot at all times during an exploration mission, and is concerned only with the actions that lead to further exploration of the environment. However, it must be noted that the works in [16, 19, 51, 66, 67] relaxes this assumption to include actions in the exploration mission that reduces the robot pose and map estimation uncertainty. As exploration actions can be easily combined with SLAM estimate improving actions using behavior based action selection [68], in order to limit the scope of the presentation of this thesis, integration of SLAM estimate improving actions to exploration missions is not further examined in this dissertation.

2.3 Efficient approaches for frontier cell extraction

As mentioned in the introduction of this dissertation, efficient algorithms on generating and managing frontier information allows frontier based exploration algorithms to operate at a high decision frequency to improve the exploration efficiency. In addition, existing works on incremental generation and management of dynamic navigational data [69, 70, 71, 72] illustrate the need for the development of efficient algorithms for robotic systems. Such efficient algorithms free up valuable computing resources which can then be used in deploying other complex navigation algorithms, even on robots with low computing capability, to improve efficiency of missions. This section concludes the chapter with a detailed review of research work in the literature on efficient algorithms for frontier cell detection, which are targeted towards increasing decision frequencies and reducing the computational burdens of robots. These methods are also compared with the proposed incremental frontier detection

2. BACKGROUND

method which is described in the following chapters. The approach which is common to most of these proposed efficient frontier detection strategies is to bound the processing of the occupancy grid map information to only the updated regions in the map at each map update step and incrementally manage all the frontier cell information across map update steps.

2.3.1 Wavefront Frontier Detector (WFD)

Wavefront Frontier Detector (WFD) [29] is based on the idea of processing only the mapped area instead of the entire map negating the need to unnecessarily process unknown map cells to detect frontiers. At each map update step a Breadth First Search (BFS) is initialized from the robot's current position cell and is progressed outwards until all the mapped cells are processed to extract frontier cells. The method utilizes the heuristic that frontier cells appear in contours to combine them in to groups of cells. Therefore when a frontier cell is detected during WFD's initial BFS, in order to extract the rest of the cells chained to the contour a second BFS is initialized and executed. As the robot continues to explore the environment the number of mapped cells in the grid map gets increased. Hence at each increasing map update step the number of grid cells processed by WFD gets increased. In addition, with the increasing number of frontier contours present especially during initial stages of an exploration mission, the WFD method regenerates all the contours at each map update step thus requiring additional unnecessary computation. These result in the WFD method's execution time to extract frontier contours to increase monotonically along the mission time and ultimately degenerate it to the traditional approach of processing the entire map, as high percentage of the grid map gets marked as mapped by the robot.

2.3.2 Fast Frontier Detector (FFD)

The Fast Frontier Detector [29] method processes only the current range readings rather than map information. At each range sensor update, the output range values are sorted according to their polar coordinates with respect to sensor's current pose. Then contours of points are extracted from these sorted range points and are mapped

2.3 Efficient approaches for frontier cell extraction

to grid cells. Then the cells in these contours are checked for being frontier cells. The detected frontier cells are grouped together into frontier contours using the previous contour information. However some of these detected frontier contours may not be new frontier contours due to the robot not moving during a sensor update step and the fast refreshing rate of the sensor. Therefore, FFD method explicitly keeps track of the number of sensor visits (i.e. sensor covers) of each map cell to identify any re-detection of an already available frontier contour as a new frontier contour. Using the sensor visit counts, any frontier contour whose cells are visited more than once are filtered out from being re-detected as new frontier contours. FFD's use of only the latest sensor data for frontier detection requires a separate database to store frontiers detected during previous steps. This database is incrementally updated to maintain the frontier information globally. More information about this database is provided in Sec. 2.3.5. The current sensor data does not indicate the elimination of previously detected frontier cells due to them being mapped by the sensor. In FFD, this is done by computing a rectangular *Active Area* based on the borders of the current sensor range values. The use of Active Region is similar to the proposed SRFD approach in Chap 4 where only the area bounded by the borders of the newly updated cells are processed. Each grid cell inside the Active Area is checked to detect for any cells that are no longer frontier cells and they are deleted from the database accordingly. Similarly any newly detected frontier contour is inserted to the database including merging of multiple frontier contours. While FFD is considerably efficient compared to WFD, its non-use of the probabilistic occupancy grid map could restrict its use only to robotic systems fitted with highly accurate range sensors.

2.3.3 Incremental WFD (WFD-INC)

WFD-INC [73] is a modification to the WFD method by restricting the BFS for frontiers only to the Active Region in the map (i.e. locally updated region). Since WFD-INC processes the map locally, a frontier database is required to maintain the global frontier information across map update steps. The method uses the same data-structure used by FFD as the frontier database. New frontier contours are extracted from the Active Region and are inserted to the frontier database.

2. BACKGROUND

Active Area is also checked for possible elimination of old frontier cells and then the corresponding contours are removed from the frontier database. Since WFD-INC algorithm is proposed and tested on particle based SLAM systems, it requires clearing of previous data from the database and reinitialization of all the data by processing of the complete map when the best particle representing the map changes. WFD-IP is a simple improvement upon WFD-INC where for each particle a separate instance of WFD-INC is executed. This alleviates the need to clear old data and reinitialize the whole detection process saving additional computation time.

2.3.4 Oriented bounding box based Frontier Detector (OFD)

The OFD method proposed in [31] is similar to WFD-INC where only the active region is processed to update frontier information. Instead of generating the active region based on the current range sensor boundaries, OFD further restricts the processing only to the actual cells that become frontiers or lose frontier cell status. This is done by tracking the changes to updated map cells between two map update steps through each sensor update. Four rules are used in OFD to ensure that all the changes that associate with potential frontier cells and potential elimination of frontier cells are tracked between two map update steps. These tracked cells are clustered and their Oriented Bounding Boxes (OBB) [74] are used to detect the changes to frontier cells to further reduce the execution time. The detected frontier cell changes need to be transformed to the map coordinates from the OBB coordinate frames. This required back and forth coordinate transforms on a grid results in shifting of detected frontier cells, results in a reduction of the accuracy of detection.

2.3.5 Frontier databases

The frontier database proposed for FFD and also used by WFD-INC and WFD-IP methods is primarily based on two data-structures. A 2D grid which is an extension of the 2D occupancy grid map is used to store two additional parameters for each grid cell. For each grid cell an integer value that labels the frontier contour the grid cell belongs is stored as one parameter. If the grid cell is not a frontier cell, the label

2.3 Efficient approaches for frontier cell extraction

is empty. A counter that stores the number of times the sensor has mapped the cell is also stored for each grid cell as the second parameter. This is used to identify re-detection of frontiers in FFD.

The extracted frontier contours are stored using a hash table data structure with a unique label used to index into each frontier contour. The frontier cells associated with each index element is stored using a list. During extraction of frontiers, the algorithm only needs to iterate through the frontier index. For each index the associated frontier contours can be directly retrieved using the stored lists. Removal of each frontier cell requires extracting the correct label from the 2D grid and then indexing to the correct list in the hash table and searching for the correct cell in the list. Since individual cells cannot be indexed in this database, this searching operation can lead to inefficient deletion operations especially if the frontier contours stored in the database are long. In addition, deletion of each cell triggers splitting of a contour list, removing the associated frontier index and inserting the two resultant lists to the index with two new index labels. This requires all the labels of the cells of the two resultant lists, in the 2D grid to be updated. This also becomes inefficient with long frontier contours. Similarly, inserting new frontier contours require the labels in the 2D grid to be updated. In addition, if the inserted contour overlaps with existing contours, they are merged together and inserted to the hash table as new elements. This also requires relabeling of grid cells in the 2D grid and could make the updating procedure inefficient if the frontier contours are long.

The database used by the OFD method is largely similar to the one described earlier with minor differences. This also stores a 2D label grid similar to the previous database. However, instead of storing the entire contour as an element in the hash table, only a single frontier cell is stored for each frontier index. Therefore unlike the database used by FFD, no search operations are required during deletion of canceled frontier cells. The rest of the operations in frontier cell deletion are similar to FFD's database. The frontier contour insertion operation is similar to the FFD's operation with a minor difference of not storing the newly extracted frontier cell as a list in the hash table. In order to retrieve a frontier contour, an operation similar to flood-fill must be executed on the 2D label grid with the seed cell for each contour extracted from the hash table. Hence, this database sacrifices the frontier retrieving efficiency to have more efficient updating operations.

2. BACKGROUND

2.3.6 Valid frontier extraction

FFD and many variants of WFD methods do not specifically mention a step of calculating the safe regions in the environment. In addition, none of these methods contain reachability detection integrated to the frontier extraction process. Therefore, they could output map boundaries that are unreachable by the robot, requiring additional steps to filter these out, as evaluation of unreachable frontier cells degrade the efficiency of the decision steps of exploration missions. The incremental frontier extraction strategy formulated in this thesis includes both incremental management of safe-space and incremental management of reachable space, which results in a single integrated set up of detecting only safe and reachable frontiers eliminating the need for separate frontier filtering step. In addition, the extracted frontier cells are reported as contours, further avoiding the need for additional grouping/clustering steps.

Chapter 3

Incremental Extraction of Safe and Reachable Map Information

Frontier based exploration missions evaluate only valid frontier cells (i.e. both safe and reachable cells) during their decision steps. Invalid frontier cells are discarded from being evaluated as they cannot be reached from robot's current location during an exploration mission. Therefore an incremental frontier generation procedure requires incremental generation and management of safe and reachable cells from the map, at each map update step to filter out phantom frontier cells. This chapter details a data structure and a set of associated incremental update algorithms and the corresponding work-flow that extracts the safe and reachable cells from the occupancy grid map at each update step in an efficient manner.

Sec. 3.1 details incrementally computing the set of safe cells from the occupancy grid map at each map update step. This is achieved by incrementally calculating the Minkowski sum between obstacle cell set and robot's footprint cell set using a modified dynamic brushfire algorithm. The generated inflated obstacle cell information is used to compute the safe cell set $\mathcal{S}_k$ incrementally. In Sec. 3.2, incremental detection of the reachability of the grid cells from incrementally extracted $\mathcal{S}_k$ is presented. The notion of *safe-patch* is defined and a graph data structure that stores the connectivity among these safe-patches is used to formalize the operation to detect reachable cells in the map (Sec. 3.2.1). The maintenance of the proposed graph data structure through a forest of incrementally managed spanning trees is formalized in Sec. 3.2.3

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

and the associated update procedures are detailed as algorithms. The ability of the proposed approach to efficiently generate the safe and reachable grid cells through complete exploration missions with a bound on the number of processing operations is validated using real world experimental data and the results are summarized in Sec. 3.3.

3.1 Incremental identification of safe cells

Extraction of the mapped free space boundary requires the identification of safe cells in the occupancy grid map. Considering the definition of safe cell set $\mathcal{S}_k = \mathcal{F}_k - \mathcal{J}_k$, determining the inflated obstacle cells $\mathcal{J}_k$ in the map incrementally is sufficient as the set $\mathcal{F}_k$ denoting free grid cells is already available through the updated map information. Since obstacle grid cells can appear and disappear from the map due to map uncertainties, the inflated obstacle detection algorithm must consider both inflation and deflation of obstacle cells. Fig. 3.1 illustrates an example situation where grid cells are correctly/incorrectly being classified as obstacles or cleared from being obstacles and resultant changes to the inflated obstacle cell information. Addition of new obstacle cells require an inflation operation while deletion of obstacle cells require a deflation operation.

Algorithm 3 Inflate-Obstacle-Cells($\mathcal{O}_k^{new}$)

```

1:  $\mathcal{J}_k \leftarrow \mathcal{J}_{k-1} \cup \mathcal{O}_k^{new}$ 
2:  $Q \leftarrow \mathcal{O}_k^{new}$ 
3: for all  $o \in \mathcal{O}_k^{new}$  do
4:    $I[o] \leftarrow o$ 
5:    $D[o] \leftarrow 0$ 
6: end for
7: while  $\sim \text{EMPTY}(Q)$  do
8:    $c \leftarrow \text{GET}(Q)$ 
9:   for all  $n \in \mathcal{N}_4(c)$  do
10:    if  $\text{eucl}(n, I[c]) < D[n]$  and  $\text{eucl}(n, I[c]) < R_R$  then
11:       $D[n] \leftarrow \text{eucl}(n, I[c])$ 
12:       $I[n] \leftarrow I[c]$ 
13:       $\mathcal{J}_k \leftarrow \mathcal{J}_k \cup \{n\}$ 
14:       $\text{Insert}(Q, n)$ 
15:    end if
16:  end for
17: end while

```

A modified version of Dynamic Brushfire [70] algorithm is used to process the sets

3.1 Incremental identification of safe cells

Figure 3.1: Incremental management of obstacle inflation. (a) Occupancy grid map and (b) the map with inflated obstacles superimposed at step k . At next update step $k+1$, (c) some new obstacle cells are added (e.g. area marked by green box) and some old obstacle cells are deleted (e.g. area marked by yellow box). (d) The incremental changes to the obstacle inflation are indicated. Blue box is the footprint of the robot.

$\mathcal{O}_k^{new}$ and $\mathcal{O}_k^{del}$ (refer Sec. 2.1.3) to incrementally generate the inflated obstacle cell set from $\mathcal{J}_{k-1}$ to $\mathcal{J}_k$. The update algorithm uses the mapping between an inflated obstacle cell c in $\mathcal{J}_k$ and its associated closest obstacle cell o in $\mathcal{O}_k$, denoted by $I[c] = o$. It also stores the current distance from the inflated obstacle cell to its closest obstacle cell in D . Algorithm 3 uses the new obstacle information $\mathcal{O}_k^{new}$ to inflate cells according to definition 4 and its operation is illustrated in fig 3.2(a) and fig. 3.2(b). Algorithm 4 removes all the inflated cells whose distances were calculated from cells in $\mathcal{O}_k^{del}$ which are no longer classified as obstacle cells using a breadth first search. The progression of the wavefronts of deletion is illustrated in fig. 3.2(c). However this operation could delete inflated cells which still satisfy definition 4 with a different obstacle cell. In line 18 of Alg. 4, deflated cells that have at least one neighboring inflated cell whose associated obstacle cell is different are stored in the set L as depicted in fig. 3.2(d). The set L is later used by algorithm

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

5 to repair incorrectly removed inflated cells.

Algorithm 4 Deflate-Obstacles($\mathcal{O}_k^{del}$)

```

1:  $\mathcal{J}_k \leftarrow \mathcal{J}_k - \mathcal{O}_k^{del}$ 
2:  $Q \leftarrow \mathcal{O}_k^{del}$ 
3:  $I^{old} \leftarrow I$ 
4:  $L \leftarrow \phi$ 
5: for all  $o \in \mathcal{O}_k^{del}$  do
6:    $I[o] \leftarrow \phi$ 
7:    $D[o] \leftarrow \infty$ 
8: end for
9: while  $\sim \text{EMPTY}(Q)$  do
10:   $c \leftarrow \text{GET}(Q)$ 
11:  for all  $n \in \mathcal{N}_4(c)$  s.t.  $n \in \mathcal{J}_k$  do
12:    if  $I[n] = I^{old}[c]$  then
13:       $D[n] \leftarrow \infty$ 
14:       $I[n] \leftarrow \phi$ 
15:       $\mathcal{J}_k \leftarrow \mathcal{J}_k - \{n\}$ 
16:       $\text{Insert}(Q, n)$ 
17:    else
18:       $L \leftarrow L \cup \{c\}$ 
19:    end if
20:  end for
21: end while
22: return  $L$ 

```

In line 2 of algorithm 5, the neighboring inflated cell with the smallest distance to its associated obstacle cell is selected for cell c which is deflated in alg. 4. If the distance is less than the inflation radius it is added to a priority queue Q to reinitialize the inflation step to add cells that were deflated which are still valid inflated cells. The progression of the wavefront to repair still valid inflated cells is illustrated in fig. 3.2(e) and the result is shown in fig. 3.2(f).

3.2 Incremental detection of reachability

This section details the development of an incremental algorithm to maintain the reachability information of the map for the robot which is later used to extract only reachable boundary cells as frontiers according to Definition 15. The formulation is based on generating a graph of traversable “safe-patches” and maintaining their interconnectedness through a collection of dynamic spanning trees to retrieve the reachability information.

Figure 3.2: (a) Progression of the two wavefronts (dark blue circles) to identify inflated obstacles. The two red dots A,B are new obstacle cells. The arrows indicate the direction of the wavefront propagation. (b) The wavefront progression terminates when it reaches the inflation radius threshold R_R . (c) Propagation of wavefronts for deflation operation (light blue circle) after obstacle cell B is deleted. (d) The green line indicates the set of cells stored in L in algorithm 4 that represent neighboring cells of deflated cells with different source obstacle cells. (e) Propagation of new wavefronts (in orange color) to rediscover cells that are valid inflated cells. (f) At the end of the second inflation step, only the inflated cells based on remaining obstacle cells are present.

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

Algorithm 5 Repair-Obstacle-Inflation(L)

```

1: for all  $c \in L$  do
2:    $n^* \leftarrow \arg \min_{n \in \{\mathcal{N}_8(c) \cap \mathcal{J}_k\}} D[n]$ 
3:   if  $I[n^*] \neq \phi$  and  $\text{eucl}(c, I[n^*]) \leq R_R$  then
4:      $D[c] \leftarrow \text{eucl}(c, I[n^*])$ 
5:      $I[c] \leftarrow I[n^*]$ 
6:      $\mathcal{J}_k \leftarrow \mathcal{J}_k \cup \{c\}$ 
7:      $\text{Insert}(Q, c)$ 
8:   end if
9: end for
10: while  $\sim \text{EMPTY}(Q)$  do
11:    $c \leftarrow \text{GET}(Q)$ 
12:   for all  $n \in \mathcal{N}_4(c)$  do
13:     if  $\text{eucl}(n, I[c]) < D[n]$  and  $\text{eucl}(n, I[c]) < R_R$  then
14:        $D[n] \leftarrow \text{eucl}(n, I[c])$ 
15:        $I[n] \leftarrow I[c]$ 
16:        $\mathcal{J}_k \leftarrow \mathcal{J}_k \cup \{n\}$ 
17:        $\text{Insert}(Q, n)$ 
18:     end if
19:   end for
20: end while

```

At each map update step k , a set U_k of cells are updated. Hence it can be deduced that at any given update step k an existing $A'(C_k^i)$ of connected component $C_k^i \in \mathcal{C}_k$ of G'_k , is generated by a discrete sequence of updates up to step k . Therefore, the associated connected component i at step k , C_k^i , can be partitioned as $P_k^i = \{\mathcal{X}_1^i, \mathcal{X}_2^i, \dots, \mathcal{X}_k^i\}$ according to the update step values of grid cells, i.e. $ts(c)$, with $\forall u \in \mathcal{X}_j^i, ts(c(u)) = j$ for $j = 1 \dots k$. For example, $C_4^1 = \{\mathcal{X}_1^1 \cup \mathcal{X}_2^1 \cup \mathcal{X}_3^1 \cup \mathcal{X}_4^1\}$ in Fig. 3.3.

Given a $\mathcal{X}_j^i \in P_k^i$, for some $C_k^i \in \mathcal{C}_k$, each l^{th} connected component of $\mathcal{X}_j^i$, $\mathcal{Y}_j^{i,l} \subseteq \mathcal{X}_j^i$, $l = 1 \dots |\mathcal{Q}_j^i|$, represents a single patch of traversable safe grid cells generated at step j . Therefore, each $\mathcal{X}_j^i$ can be partitioned according to its connected components, i.e. $\mathcal{X}_j^i = \bigcup_{Q_j^i} \mathcal{Y}_j^{i,l}$, where Q_j^i is the set of connected components of $\mathcal{X}_j^i$. E.g. $\mathcal{X}_4^1 = \{\mathcal{Y}_4^{1,1} \cup \mathcal{Y}_4^{1,2}\}$ and $\mathcal{X}_3^1 = \{\mathcal{Y}_3^{1,1}\}$ in Fig. 3.3. The set of grid cells represented by each connected component $\mathcal{Y}_j^{i,l}$ is denoted by $A(\mathcal{Y}_j^{i,l})$ and is referred to as a “safe-patch”. The collection of all $\mathcal{Y}_j^{i,l}$ is denoted by Q_k and is defined as $Q_k = \bigcup Q_j^i; j = 1 \dots k, i = 1 \dots |\mathcal{C}_k|$. Each element in Q_k represents the connected sub-graph in G'_k of the associated safe-patch.

Using the above definitions, a graph $G_k = (V_k, E_k)$ is defined as an abstraction of G'_k using safe-patches. Each node $v \in V_k$ represents a safe-patch $A(\mathcal{Y})$, where $\mathcal{Y} \in Q_k$. The function $A, A : Q_k \mapsto 2^{S_k}$ provides a mapping between a node in G_k to

Figure 3.3: Generation of safe-patches up to update step 4. Two connected components $\mathcal{C}_4^1, \mathcal{C}_4^2$ are identified. $\mathcal{C}_4^1$ is reachable and $\mathcal{C}_4^2$ is not. The dotted line indicates the robot's traversed path. Except for $\mathcal{X}_4^1$, all the other partition members of connected components have only one safe-patch. The maroon colored blobs represent obstacles and brown colored blobs surrounding these obstacles represent the inflated obstacle cells. The shaded area is the mapped free space.

Figure 3.4: The graph that maintains the connectivity among different safe-patches. Green nodes - visited nodes. White nodes - non-visited nodes.

its associated set of connected safe cells. The function $Y : V_k \mapsto Q_k$ maps each node in graph G_k to an element in Q_k which represents a safe-patch. An edge $e = (u, v)$ is defined to be in $E_k \iff \exists$ a $r \in Y(u)$ and $s \in Y(v)$ s.t $(r, s) \in E'_k$. The function $\mathcal{V} : \mathcal{S}_k \mapsto V_k$ provides the mapping between a safe grid cell and its associated node in the graph G_k . Given a node $\varphi \in V_k$, $\mathcal{N}^\varphi$ is defined as the set of neighboring nodes of φ . $\mathcal{N}^\varphi = \{\psi \in V_k : (\varphi, \psi) \in E_k\}$. Fig. 3.3 illustrates an example generation of safe-patches and Fig. 3.4 illustrates the graph representing the connectivity of these safe-patches.

3.2.1 Checking the reachability of a grid cell

Using the above definitions, the following basic results provide an operation to check the reachability of safe cells in the occupancy grid map using the graph G_k .

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

Lemma 1. *Given grid cells m and n , $\exists$ a traversable path for robot from m to n at step k , if $\mathcal{V}(m)$ and $\mathcal{V}(n)$ belong to the same connected component in G_k .*

Proof. Let $\mathcal{V}(m), \mathcal{V}(n) \in V_k$ belong to the same connected component C^i of G_k . Two cases need to be considered.

Case I : There exists an edge e between $\mathcal{V}(m)$ and $\mathcal{V}(n)$. $e = (\mathcal{V}(m), \mathcal{V}(n)) \in E_k$.
 From the definition of E_k , $\exists r \in Y(\mathcal{V}(m))$ and $s \in Y(\mathcal{V}(n))$ s.t. $(r, s) \in E'_k$.
 And also $\forall p \in Y(\mathcal{V}(m)) \exists$ a path from p to r in G'_k and $\forall q \in Y(\mathcal{V}(n)) \exists$ a path from q to s in G_k , as both $Y(\mathcal{V}(m)), Y(\mathcal{V}(n)) \in Q_k$ and are connected sub-graphs in G'_k . Since $(r, s) \in E'_k$, it follows that $\forall p \in Y(\mathcal{V}(m))$ and $\forall q \in Y(\mathcal{V}(n)) \exists$ a path from p to q in G'_k . Hence, $\exists$ a path from $\mathcal{V}'(m)$ to $\mathcal{V}'(n)$, hence a traversable path for robot from grid cell m to n .

Case II : There does not exist an edge e between $\mathcal{V}(m)$ and $\mathcal{V}(n)$

Since $\mathcal{V}(m)$ and $\mathcal{V}(n)$ belong to the same connected component, $\exists$ a path $p = (e_1, e_2, \dots, e_n)$ from $\mathcal{V}(m)$ to $\mathcal{V}(n)$ in G_k where $e_1 = (\mathcal{V}(m), u)$ for some $u \in V_k \setminus \mathcal{V}(m)$ and $e_n = (v, \mathcal{V}(n))$ for some $v \in V_k \setminus \mathcal{V}(n)$.

Consider e_1 ; Using the proof of Case I, $\exists$ a path from $\mathcal{V}'(m)$ to $\forall p \in Y(u)$ in G'_k . Considering $e_2 = (u, r)$, $\exists$ a path from p to $q \forall p \in Y(u)$ and $\forall q \in Y(r)$ in G'_k . Hence, $\exists$ a path from $\mathcal{V}'(m)$ to $q \forall q \in Y(r)$ in G'_k .

By using the above procedure until e_n , it is deduced that $\exists$ a path from $\mathcal{V}'(m)$ to $q \forall q \in Y(\mathcal{V}(n))$ in G'_k . Since $\mathcal{V}'(n) \in Y(\mathcal{V}(n))$, $\exists$ a path from $\mathcal{V}'(m)$ to $\mathcal{V}'(n)$ in G'_k . Therefore a traversable path exists for robot from grid cell m to n .

□

Theorem 1. *Given the robot's position grid cell $r \in S_k$ and a grid cell $c \in S_k$, cell c is reachable by the robot if $\mathcal{V}(r)$ and $\mathcal{V}(c)$ belong to the same connected component in G_k .*

The proof of Theorem 1 follows directly from Lemma 1.

3.2.2 Checking for reachability across update steps

Now that it is established that the reachability of a grid cell from the robot's position can be checked using the graph G_k and its connected components, this section introduces the data structure used to represent the connected components in G_k that change with map update step k and the associated update algorithms used to maintain this dynamic data structure.

First, it is important to understand how the map and its abstracting graph G_k evolves with each update step k so that this information can be used to maintain the graph connectivity information incrementally. At each step k , updates to the occupancy grid map would add a new set of safe cells $\Delta\mathcal{S}_k$. The safe patches (i.e. $A(\varphi)$ where $\varphi \in Q_k$) generated from these safe cells can be represented by a set of new nodes, denoted by $\Upsilon'_k \subset V_k$. At the same time, discovery of some new obstacle cells during step k could split a safe patch $A(\psi)$ present at step $k-1$, $\psi \in V_{k-1}$, to create multiple disjoint safe patches and add their representative nodes to V_k as illustrated in Fig. 3.9(a). Let, $\hat{\Upsilon}_k$ denote the set of such new nodes added due to splitting of old safe patches. Hence, after update step k , the graph G_{k-1} is converted to G_k by deleting nodes that correspond to safe patches that were split and adding newly generated nodes $\{\Upsilon'_k \cup \hat{\Upsilon}_k\}$ together with updates to the associated edges.

Connectivity queries in graphs are efficiently resolved using spanning trees. The connectivity between any two nodes is evaluated by checking for the equality of the root nodes of their associated spanning trees. The dynamic nature of the graph G_k requires management of dynamic spanning trees. However, the existing approaches do not support full dynamic graph connectivity management and are restricted to a fixed graph size [75] or only to increasing graphs without deletions [76]. Therefore, a new data-structure to manage the connectivity among nodes in G_k is introduced in the following sections.

3.2.2.1 Representing the reachable connected component

The *reachable connected component* in the graph G_k is defined to be the connected component that contains the node that represents the safe patch the robot currently resides. Since the robot's exploratory motion is continuous this motion can be abstracted to robot visiting a sequence of connected safe patches, thus visiting their

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

associated nodes. A node $\psi \in V_k$ is defined to be a visited node if $A(\psi) \cap \mathcal{P}_k \neq \phi$, where $\mathcal{P}_k \subset \mathcal{S}_k$ is the set of grid cells representing robot's trajectory so far.

Since visited safe patches are connected it follows that the visited nodes are connected too and belong to the reachable connected component. This *visited* information can be used to store these nodes in a simple tree where every node has the same root node ν . The set of nodes which are not visited by the robot, yet belong to the same reachable connected component can be attached to the visited nodes to generate a spanning tree for the reachable connected component as illustrated in Fig. 3.5. Then the check for connectivity between any two visited nodes in the reachable connected component is bounded to a single operation of extracting the parent node ν .

Figure 3.5: (a) A set of connected safe-patches and their associated nodes. Green colored nodes represent the visited nodes. White colored nodes are not visited yet, but are in the reachable connected component, hence connected to visited nodes. (b) The spanning tree representing the reachable connected component.

The reachability detection of safe-patches represented by non-visited nodes require multiple search steps to access the root node of the associated spanning tree. Hence, for nodes representing reachable non-visited safe-patches, the maximum shortest-distance to its closest visited-node provides a bound on the reachability detection search steps. Considering v_r to be the constant speed of the robot, R_{max} to

be the maximum sensor range and f_m to be the map update frequency, the distance the robot travels between two map update steps can be calculated as $\delta d = v_r / f_m$. Considering the common situation of $\delta d < R_{max}$, the maximum number of consecutive update steps, s_u , the map goes through without the robot moving to a different safe-patch is given by $s_u = \lfloor R_{max} / \delta d \rfloor = \lfloor f_m R_{max} / v_r \rfloor$. It directly follows that the upper bound on the shortest-distance from a non-visited reachable node to a visited node in G_k is s_u as illustrated in Fig. 3.6(a). Hence, the maximum number of search steps to retrieve the root ν of the reachable connected component is bounded to $s_u + 1$.

3.2.2.2 Representing non-reachable connected components

Non-visited nodes that do not belong to the reachable connected component can be extracted to separate disjoint connected components and are represented using spanning trees. The safe patches represented by such nodes occur when sensor maps areas that are visible through smaller openings, that the robot is unable to travel through. Such a situation is illustrated by the two nodes represented by the tree $\mathcal{T}_{k-1}^2$ in Fig. 3.8. The progression of connected components representing unreachable safe-patches is bounded based on the robot's inability to move beyond the smaller opening in the environment as depicted in Fig. 3.6(b). Therefore the depth of such a connected component is also bounded by s_u , as calculated in the previous section. During any update step, if any of these non reachable connected components gets connected to the reachable connected component, their representative spanning trees can be merged to the spanning tree representing the reachable connected component.

3.2.3 Maintenance of the proposed data structure

Considering that at step $k - 1$, the connected components of graph G_{k-1} are represented using a set of spanning trees $\mathcal{T}_{k-1}$ according to section 3.2.2. A collection of update algorithms are provided below that modifies the set $\mathcal{T}_{k-1}$ of spanning trees to generate a new set $\mathcal{T}_k$ of spanning trees that represent the connected components of graph G_k . The order of invocation of these update steps and their dependencies are illustrated in the flowchart in Fig. 3.7 and are summarized below.

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

Figure 3.6: (a) The robot (rectangle) moves inside the visited safe-patch (white) in the arrow's direction and for each δd distance, a new reachable non-visited safe-patch (shaded) is generated. Nodes representing these non-visited safe-patches are sequentially added while increasing their depth values until the robot ventures out the initial safe-patch after the $\lfloor R_{max}/\delta d \rfloor$ step. (b) During the two previous map updates (the robot positions are shown in dotted rectangles), two safe-patches are generated (shaded) beyond the small opening. The associated graph nodes are also indicated.

Figure 3.7: The order of update operations conducted to maintain the spanning tree data structure

First, the safe-patches that lose full traversability due to new obstacle information are identified and are split to individual safe-patches and their corresponding nodes $\hat{\Upsilon}_k$ are generated. The nodes corresponding to split safe-patches, Υ_k^s , are removed from $\mathcal{T}_{k-1}$. An example of splitting a safe-patch to multiple disjoint safe-patches is illustrated in Fig. 3.8 and Fig. 3.9(a). The nodes in $\hat{\Upsilon}_k$ are inserted to the resultant spanning trees to generate the collection of trees $\dot{\mathcal{T}}_k$. Note that the operations of removing split nodes and adding nodes in $\hat{\Upsilon}_k$ are local operations and hence might render the trees in $\dot{\mathcal{T}}_k$ non-spanning trees. Next, safe patches due to new safe cells $\Delta\mathcal{S}_k$ and their associated nodes Υ'_k are generated, Fig. 3.9(b). They are inserted to $\dot{\mathcal{T}}_k$ to generate the collection of trees $\ddot{\mathcal{T}}_k$ as illustrated in Fig. 3.10(a).

3.2 Incremental detection of reachability

Again, since the trees in $\hat{\mathcal{T}}_k$ might be non-spanning trees the set of generated trees $\hat{\mathcal{T}}_k$ may not be spanning-trees. The remaining new nodes which are not used for spanning tree augmentation, Υ_k^I , due to these nodes not having any neighboring nodes, are represented in isolated spanning trees $\hat{\mathcal{T}}_k$. The collection of trees in $\hat{\mathcal{T}}_k$ need to be checked for possible connectivity among different trees and any two connected trees are merged to generate spanning trees of connected components of G_k . An example of such a tree merging operation is illustrated in Fig. 3.10(b). Finally the union of the set of trees generated from the merging operation and $\hat{\mathcal{T}}_k$ results in the collection of spanning trees $\mathcal{T}_k$ that represent the connected components in G_k . The individual update steps are described in detail in the following sub-sections.

Figure 3.8: The figure on the left illustrates nodes in G_{k-1} with new obstacle/inflated obstacle information added during step k . The green circles are visited nodes and white circles are non-visited nodes in V_{k-1} . The dark blue node corresponds to the safe-patch that gets split due to new obstacle information. The enlarged section on the right illustrate the map before adding obstacle information from update step k . During step k , new obstacle information (dotted) is added to the map that changes the connectivity information.

The update algorithms make use of the following definitions.

Definition 16. The set of boundary cells of inflated obstacle cells generated for step k , $\Delta\mathcal{J}_k^B$, is defined as $\{c : c \in \mathcal{S}_k \text{ s.t. } ts(c) = k \text{ and } \exists (\mathcal{V}'(c), \mathcal{V}'(i)) \in E'_k \text{ where } i \in$

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

Figure 3.9: (a) Due to new obstacle information, the dark blue colored node in fig. 3.8 need to be split. Two new nodes, a and b , replace this node which correspond to disconnected safe-patches. Node a is connected to its neighboring node greedily. Node b remains isolated to generate the tree $\dot{\mathcal{T}}_k^3$. The set of new nodes generated by splitting nodes is denoted by $\hat{\Upsilon}_k$. The set of spanning trees resulting from the splitting of nodes and adding of nodes a and b is denoted by $\dot{\mathcal{T}}_k$. (b) A new node, $c \in \Upsilon'_k$ is added based on newly added free cells $\Delta\mathcal{S}_k$ to occupancy grid map. The dashed lines attached to c indicate adjacency information that is used to connect c to spanning trees.

$\mathcal{T}_k\}$.

Definition 17. The set of inflated obstacle boundary cells generated at step k that belong to the same $A(\psi)$ where $\psi \in V_{k-1}$ is defined as $\xi_k^\psi = \{c : c \in \Delta\mathcal{J}_k^B \text{ s.t. } \mathcal{V}(c) = \psi \in V_k\}$

Definition 18. The set Ξ_k is defined as the partition of $\Delta\mathcal{J}_k^B$ using all the possible ξ_k^ψ .

$$\Delta\mathcal{J}_k^B = \bigcup_{\xi_k^\psi \in \Xi_k} \xi_k^\psi$$

Definition 19. Given a node φ representing a safe patch, $\Lambda(\varphi)$ denotes all the nodes $n \in \mathcal{N}^\varphi$ s.t. $\text{parent}[\varphi] \neq n$ and $n \notin \text{child-nodes}[\varphi]$ in the spanning $\mathcal{T}_k^i$ where $\varphi \in \text{nodes}[\mathcal{T}_k^i]$.

Figure 3.10: (a) Nodes in Υ'_k are connected to nodes in trees in $\check{\mathcal{T}}_k$ to generate trees $\check{\mathcal{T}}_k$. Here, only node c gets connected to $\check{\mathcal{T}}_k^3$, illustrated by the dark black edge. (b) Finally, possible connectivity among trees in $\check{\mathcal{T}}_k$ are searched and the set of spanning trees for the connected components in graph G_k , $\mathcal{T}_k$ is generated. In this example, node c is connected to tree $\check{\mathcal{T}}_k^1$.

3.2.3.1 Updating safe-patch information

Identifying all the set of grid cells belonging to a particular safe-patch, given any cell in the safe-patch, and associated them with a new node is the fundamental operation in reachability management algorithms. Given a cell p and its current node mapping $\mathcal{V}(p) = \psi$, Alg. 6 executes a flood-fill operation to identify all the cells that are connected to p with the same node mapping to ψ . Node mappings of the identified cells are changed to a new node φ .

Line 1 initializes the flood-fill operation with the seed cell. The loop in lines 5-15 conduct flood-fill for safe connected similar cells where similarity is given by the mapping to the same node (check $\mathcal{V}(q) = \psi$ in line 8). In line 9, the mapping for each identified cell is changed to φ . In line 12, $\mathcal{N}^\varphi$ is populated with all the neighboring nodes to the new node φ . Along with the above mentioned modifications to the flood-fill operation, the following basic theorem can be stated.

Theorem 2. *Given a seed cell $p \in \mathcal{S}_k$ and its associated node $\mathcal{V}(p) = \psi$, execution of the function UPDATEV identifies a connected set of grid cells $A(\varphi)$, that share*

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

Algorithm 6 UPDATEV(p, ψ)

```

1:  $F \leftarrow \phi$ 
2: ENQUEUE( $F, p$ )
3:  $\varphi \leftarrow \text{GENERATE-NEW-NODE}()$  and  $\mathcal{N}^\varphi \leftarrow \phi$ 
4:  $\mathcal{V}(p) \leftarrow \varphi$ 
5: while  $\sim \text{EMPTY}(F)$  do
6:    $p \leftarrow \text{DEQUEUE}(F)$ 
7:   for all  $q \in \mathcal{N}_8(p)$  do
8:     if  $q \in \mathcal{S}_k$  AND  $\mathcal{V}(q) = \psi$  then
9:        $\mathcal{V}(q) \leftarrow \varphi$ 
10:      ENQUEUE( $F, q$ )
11:     else if  $\mathcal{V}(q) \neq \text{NIL}$  AND  $\mathcal{V}(q) \neq \varphi$  then
12:        $\mathcal{N}^\varphi \leftarrow \mathcal{N}^\varphi \cup \{\mathcal{V}(q)\}$ 
13:     end if
14:   end for
15: end while
16: return  $\{\varphi, \mathcal{N}^\varphi\}$ 

```

the same time-stamp (i.e. a safe-patch), and generates its abstracting node $\varphi \in V_k$.

Alg. 6 is used in Lemma 3 (Sec. 3.2.3.3) to derive an operation to identify safe-patches $A(\varphi_1), A(\varphi_2), \dots$ generated after splitting an existing safe-patch $A(\psi)$ due to changes to inflated obstacle data. Lemma 4 (Sec. 3.2.3.5) utilizes Alg. 6 to derive an operation to extract new safe-patches from new safe cells $\Delta\mathcal{S}_k$. Therefore, $A(\varphi) \neq A(\psi)$ for all ψ and φ .

3.2.3.2 Updating information about visited nodes

Identifying whether a newly generated node is visited or not is another fundamental operation and is described next. A node in V_k can be marked visited according to two scenarios. In the first scenario, for any newly generated $\varphi \in \{\Upsilon'_k \cup \hat{\Upsilon}_k\}$, if the robot's current position grid cell c_k^* is in $A(\varphi)$, then φ is marked visited by directly adding it to the spanning tree representing the reachable connected components. This is done by setting the parent of φ to the root node ν of the spanning tree.

The second scenario involves splitting a visited safe-patch due to obstacle information. The following trivial results provide a rule to mark such newly generated nodes as visited.

Lemma 2. *Let $\mathcal{P}_k$ be the trajectory of the robot so far. Let $A(\psi)$ be a safe-patch with $\psi \in V_{k-1}$ being the representing node and is a visited node. Consider, splitting*

$A(\psi)$ to multiple safe-patches represented by $\Omega = \{\varphi_1, \varphi_2, \dots\}$ due to new obstacle information. Then,

(a) Splitting generates only one visited node φ_i .

(b) $\{\mathcal{P}_k \cap A(\psi)\} = \{\mathcal{P}_k \cap A(\varphi_i)\}$.

Proof.

a) Consider, for the purpose of contradiction, that splitting generates more than one safe-patch that is marked visited. Then $\exists \varphi_1, \varphi_2 \in V_k$ s.t. $A(\varphi_1), A(\varphi_2)$ are two safe-patches generated by splitting $A(\psi)$. Since, $A(\varphi_1)$ and $A(\varphi_2)$ are not connected

$$\nexists \text{ a path from } \forall c_1 \in A(\varphi_1) \text{ to } \forall c_2 \in A(\varphi_2)$$

Since, both φ_1, φ_2 are assumed visited

$$\{\mathcal{P}_k \cap A(\varphi_1)\} \subseteq A(\varphi_1) \text{ and } \{\mathcal{P}_k \cap A(\varphi_2)\} \subseteq A(\varphi_2)$$

Hence, $\nexists$ a path from $\forall c_1 \in \{\mathcal{P}_k \cap A(\varphi_1)\}$ to $\forall c_2 \in \{\mathcal{P}_k \cap A(\varphi_2)\}$, this leads to $\mathcal{P}_k$ is not connected, which is a contradiction.

b) Let $\mathcal{J}^\psi \in \mathcal{J}_k$ is the set of new inflated obstacle grid cell information that triggered the splitting of $A(\psi)$. Then,

$$A(\psi) = \{\cup_k A(\varphi_k)\} \cup \mathcal{J}^\psi$$

Hence,

$$\begin{aligned} \mathcal{P}_k \cap A(\psi) &= \mathcal{P}_k \cap \{\cup_k A(\varphi_k)\} \cup \mathcal{J}^\psi \\ &= \{\cup_k \{\mathcal{P}_k \cap A(\varphi_k)\}\} \cup \{\mathcal{P}_k \cap \mathcal{J}^\psi\} \end{aligned}$$

Since $\mathcal{J}^\psi \cap \mathcal{S}_k = \phi$, $\mathcal{P}_k \cap \mathcal{J}^\psi = \phi$

From a)

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

$\exists! \varphi_i \in \Phi$ s.t. φ_i is visited

Hence, $\mathcal{P}_k \cap A(\varphi_i) \neq \emptyset$ and $\mathcal{P}_k \cap A(\varphi_j) = \emptyset \forall j \neq i$

Therefore,

$$\mathcal{P}_k \cap A(\psi) = \mathcal{P}_k \cap A(\varphi_i)$$

□

Theorem 3. *Let $p \in \{A(\psi) \cap \mathcal{P}_k\}$ be a grid cell associated with visited node ψ . Consider that $A(\psi)$ gets split to disjoint safe-patches represented by $\Omega = \{\varphi_1, \varphi_2, \dots\}$ due to obstacle information. Then, if $p \in A(\varphi_j)$ for some $\varphi_j \in \Omega$, then $A(\varphi_j)$ is the visited safe-patch generated by splitting $A(\psi)$.*

Proof. From Lemma 2, $A(\psi) \cap \mathcal{P}_k = A(\varphi_i) \cap \mathcal{P}_k$ and φ_i is the only node that is marked visited after splitting ψ . Since, for any $p \in \{A(\psi) \cap \mathcal{P}_k\} \Rightarrow p \in \{A(\varphi_i) \cap \mathcal{P}_k\}$ and $\{A(\varphi_i) \cap \mathcal{P}_k\} \subseteq A(\varphi_i)$, it implies that $p \in A(\varphi_i)$. If, $p \in A(\varphi_j)$, then it implies that $A(\varphi_j) \equiv A(\varphi_i)$, hence φ_j is the node representing the visited safe-patch generated by splitting $A(\psi)$. □

Due to the results of Lemma 2 and Theorem 3, associating a single cell $p \in \{A(\psi) \cap \mathcal{P}_k\}$ with each node ψ is sufficient to identify the node φ that becomes the only visited node when ψ is split.

3.2.3.3 Updating safe-patches with new obstacle information to generate $\hat{\Upsilon}_k$

Changes to inflated obstacle information, $\Delta\mathcal{J}_k$, may modify existing safe-patches by shrinking their sizes or by making them non-connected. If a safe-patch becomes non-connected, the safe-patch is considered to be split to multiple new safe-patches. Shrinking a safe-patch can also be considered as splitting it in to one new safe-patch and one non-safe-patch. Hence, all three of these operations require the node representing old safe-patch to be removed from the spanning tree and to generate/extract the new safe-patches and adding the corresponding nodes to the spanning tree. Given a node $\psi \in V_{k-1}$ and its associated safe-patch $A(\psi)$, the only way that $A(\psi)$

gets split to multiple disjoint safe-patches during update step k is if some grid cells in $\Delta\mathcal{J}_k$ breaks the connectedness of $A(\psi)$. Such a situation is illustrated in Fig. 3.8 where the safe-patch represented by the dark blue node gets partitioned according to new obstacle information. Hence, it follows that each disjoint safe-patch generated from such a splitting has a set of boundary cells neighboring some cells in $\Delta\mathcal{J}_k$ as illustrated in Fig. 3.11. Suppose $\xi_k^{\psi,\varphi}$ denotes the aforementioned set of inflated obstacle boundary cells for a safe-patch $A(\varphi)$, $\varphi \in V_k$ generated after splitting $A(\psi)$ at step k . Based on Theorem 2, Lemma. 3 presents the operations required to identify a safe-patch resulted from splitting an old safe-patch.

Figure 3.11: Using new inflated obstacle boundary cells (blue cells), $\{\xi_k^{\psi,\varphi_1} \cup \xi_k^{\psi,\varphi_2}\} \subset \mathcal{J}_k^B$ to identify the splitting of an old safe-patch $A(\psi)$ to two new safe-patches $A(\varphi_1)$ and $A(\varphi_2)$ (colored areas). The outwards arrows from boundary cells indicate the direction of the flood-fill operations executed. Notice that only the boundary cells that are classified as free are marked as inflated obstacle boundary cells.

Lemma 3. *If function UPDATEV is executed with $c \in \xi_k^{\psi,\varphi}$ and $\mathcal{V}(c)$ as inputs and generates the safe-patch $A(\rho)$, $\rho \in V_k$, then $A(\rho) \equiv A(\varphi)$. i.e. $A(\rho)$ is a safe-patch resulted from splitting safe-patch $A(\psi)$.*

Proof. It follows directly from Lemma 2 that executing UPDATEV, the generated $A(\varphi)$ is a safe-patch with all the cells sharing the same time-stamp. Since $c \in A(\varphi)$ and $c \in A(\psi)$, it follows that $A(\varphi) \subseteq A(\psi)$ hence $A(\varphi)$ is the connected grid cell-set i that resulted from splitting $A(\psi)$. $\square$

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

Corollary 1. *Executing UPDATEV for a cell $c \in \xi_k^{\psi, \varphi_i}$*

1. *for each φ_i , generates the set of nodes $\hat{\Upsilon}_k^u$ that abstracts the safe-patches resulted from splitting the safe-patch $A(\psi)$.*
2. *for each φ_i , and for all nodes ψ , such that $\exists \xi_k^\psi \in \Xi_k$, generates the set $\hat{\Upsilon}_k$*

Corollary 1 follows directly from Lemma. 3 to formalize the operation to generate the set $\hat{\Upsilon}_k$. The Alg. 7 uses these results to generate $\hat{\Upsilon}_k$ and update the collection of spanning trees from $\mathcal{T}_{k-1}$ to $\hat{\mathcal{T}}_k$.

The input to the algorithm is the partition of the inflated obstacle boundary cells, $\Xi_k = \bigcup_{\psi} \xi_k^\psi$ (Definition. 18). However, ξ_k^{ψ, φ_i} used in Lemma 3 to formalize the generation of a new safe-patch after splitting, for each ψ , is not available as it is this algorithm that extracts each resultant safe-patch i for each ψ . In line 4, the for-loop iterates through each cell $p \in \xi_k^\psi$. However, inside the loop, UPDATEV is executed only when $\mathcal{V}(p) = \psi$ according to check in line 5. According to Lemma 3, a single execution of UPDATEV with $p \in \xi_k^\psi$ extracts a safe-patch, say φ_i . Then for each subsequent iteration of the loop for cells $q \in \{\xi_k^{\psi, \varphi_i} \setminus \{p\}\}$, $\mathcal{V}(q) \neq \psi$ as $\mathcal{V}(q)$ has been changed earlier, thus UPDATEV is not executed. This proves that UPDATEV is executed only for a single cell from ξ_k^{ψ, φ_i} for each safe-patch φ_i , as required by Corollary 1. The for-loop from line 1, executes the for-loop from line 4 for each ξ_k^ψ . Hence, combining the operation of these two loops, it can be seen that UPDATEV is executed for a single cell from each ξ_k^{ψ, φ_i} for all i and for all modified nodes ψ , hence according to Corollary 1, the algorithm generates the set $\hat{\Upsilon}_k$.

The algorithm uses the variable *RPC* (Robot Position Cell) to keep track of the visitedness of each node based on Theorem 3. This is used in line 28 to check if the newly generated node φ is the visited node after splitting a visited node ψ . If the node φ becomes a visited node based on the check in line 28, the RPC variable associated with node φ is updated with the robot's current position cell c_k^* (line 30).

3.2.3.4 Removing split nodes Υ_k^s and adding nodes in $\hat{\Upsilon}_k$ to $\mathcal{T}_{k-1}$

Each removal of a node in Υ_k^s and adding its substitute nodes in $\hat{\Upsilon}_k$ to the spanning trees requires remapping of parent-child links in the spanning trees. The following

Algorithm 7 Update-Modified-Nodes(Ξ_k)

```

1: for all  $\xi \in \Xi_k$  do
2:    $\psi \leftarrow \mathcal{V}(\xi[1])$ 
3:    $r^\psi \leftarrow \text{GETROOT}(\psi)$ 
4:   for  $c \in \xi$  do
5:     if  $\mathcal{V}(c) \neq \psi$  OR  $c \notin \mathcal{S}_k$  then
6:       continue
7:     end if
8:      $(\varphi, \mathcal{N}^\varphi) \leftarrow \text{UPDATEV}(c, \psi)$ 
9:     for all  $n \in \mathcal{N}^\varphi$  do
10:      if  $\text{visited}[n] = \text{TRUE}$  AND  $\text{parent}[\varphi] = \text{NIL}$  then
11:        make  $n$  the parent of  $\varphi$ 
12:      else if  $\text{parent}[n] = \psi$  then
13:        make  $\varphi$  parent of  $n$ 
14:      else
15:        make  $\varphi$  and  $n$  adjacent nodes
16:        if  $\text{parent}[\psi] = n$  then
17:           $e = (n, \varphi)$ 
18:        end if
19:      end if
20:    end for
21:    if  $\text{parent}[\varphi] = \text{NIL}$  then
22:      if  $e \neq \text{NIL}$  then
23:        make  $e[1]$  parent of  $e[2] = \varphi$ 
24:      else
25:        connect  $\varphi$  to a node  $n' \in \Lambda(\varphi)$  s.t. edge  $(n', \varphi)$  is not in a cycle
26:      end if
27:    end if
28:    if  $(\text{visited}[\psi] = \text{TRUE}$  AND  $\mathcal{V}(\text{RPC}[\psi]) = \varphi)$  OR  $\mathcal{V}(c_k^*) = \varphi$  then
29:       $\text{visited}[\varphi] \leftarrow \text{TRUE}$ 
30:       $\text{RPC}[\varphi] \leftarrow c_k^*$ 
31:      if  $\text{parent}[\varphi] = n$  AND  $\text{visited}[n] = \text{FALSE}$  then
32:        make  $\varphi$  the parent of  $n$ 
33:      end if
34:      make  $\nu$  the parent of  $\varphi$ 
35:    else
36:       $r^\varphi \leftarrow \text{GETROOT}(\varphi)$ 
37:      if  $r^\varphi \neq r^\psi$  then
38:         $\mathbb{N}_k \leftarrow \mathbb{N}_k \cup \{r^\varphi\}$ 
39:      end if
40:    end if
41:  end for
42:  for each old child node  $n$  of  $\psi$  do
43:    if  $\text{parent}[n] = \psi$  then
44:       $\mathbb{N}_k \leftarrow \mathbb{N}_k \cup \{n\}$ 
45:    end if
46:  end for
47:   $\text{REMOVEDNODE}(\psi)$ 
48: end for

```

set of rules ensure the retention of information of connected components even after the safe-patch splitting operations. Let $\psi \in \Upsilon_k^s$ and φ be any resulting node after splitting safe-patch $A(\psi)$, and r^ψ and r^φ are the two root nodes of the spanning trees associated with ψ and φ respectively. Then,

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

1. If $\exists$ some $n \in \mathcal{N}^\varphi$ s.t. n is visited $\Rightarrow \text{parent}[\varphi] = n$
2. If $\nexists n \in \mathcal{N}^\varphi$ s.t. n is visited, and for some $n \in \mathcal{N}^\varphi$ $\text{parent}[\psi] = n$ and $\text{parent}[\varphi] = \text{NIL} \Rightarrow \text{parent}[\varphi] = n$
3. If $\nexists n \in \mathcal{N}^\varphi$ s.t. $(n \text{ is visited or } \text{parent}[\psi] = n) \Rightarrow \text{parent}[\varphi] = n'$ for any $n' \in \mathcal{N}^\varphi$ that maintains tree properties.
4. If $\nexists n \in \mathcal{N}^\varphi$ s.t. n is visited, and for some $n \in \mathcal{N}^\varphi$ $\text{parent}[n] = \psi \Rightarrow \text{parent}[n] = \varphi$
5. If for some n $\text{parent}[n] = \psi$ and $\nexists \varphi$ s.t. $n \in \mathcal{N}^\varphi \Rightarrow \text{add } n \text{ to } \aleph_k$
6. If $r^\varphi \neq \nu$ and $r^\varphi \neq r^\psi \Rightarrow \text{add } r^\varphi \text{ to } \aleph_k$.

Operations defined using these rules are implemented in the Alg. 7. Rules 1, 2 and 3 are mutually exclusive to each other and ensure that the new node φ is connected to some tree if a connection is available. Rule 1 connects the new node φ to the spanning tree representing reachable connected component if one of its neighboring nodes n is a visited node (lines 10-11). If none of new node φ 's neighboring nodes are visited and if one of the neighboring nodes n was also the parent of the split node ψ (lines 16-17), then this node n is made the parent of φ using Rule 2 to connect φ to the same spanning tree that ψ was a member (lines 22-23). If such a neighboring node cannot be found, using Rule 3 it is connected to any neighboring node that satisfies the tree properties (lines 24-25). Rule 4 is used to redirect parent links of child nodes of the split node ψ to the new node φ in lines 12-13. If such re-directions cannot be found, the sub-trees originating from these child nodes get disconnected from existing spanning trees. Hence these child nodes are stored in the set $\aleph_k$ using Rule 5 to ensure the connectivity information of the aforementioned sub-trees are not lost (lines 42-46). If a parent link of a child node of ψ is redirected to φ using rule 4 and φ is connected to a different tree (i.e. $r^\psi \neq r^\varphi$, check in line 37) based on Rule 3 or not connected to a tree at all, possible connectivity information between nodes in the tree containing φ and the tree that contained node ψ could be lost due to the local nature of these operations. Hence, the root of the tree containing φ , r^φ , is stored in $\aleph_k$ using Rule 6 in line 38.

At this point it can be noted that removing split nodes in Υ^s from $\mathcal{T}_{k-1}$ and inserting their substituting nodes to generate $\dot{\mathcal{T}}_k$ does not guarantee that the set of trees in $\dot{\mathcal{T}}_k$ represent spanning trees of the connected components of G_k . The Alg. 11 described in Sec. 3.2.3.8 later utilizes this knowledge to repair any disconnected spanning trees by enumerating the trees stored in $\aleph_k$ to check for possible global connectivity among different trees to generate $\mathcal{T}_k$.

3.2.3.5 Adding new nodes to G_k

The set of newly added safe cells $\Delta\mathcal{S}_k = \{\mathcal{S}_k \cap U_k\}$ need to be abstracted into the graph G_k by generating safe-patches. Let $\mathbf{S}$ be the partitioning of $\Delta\mathcal{S}_k$ to safe-patches. Then similar to Lemma 3, Lemma. 4 presents the operation to extract a safe-patch from $\Delta\mathcal{S}_k$. Similarly Corollary. 2 formalizes the operation to generate the set Υ'_k .

Lemma 4. *Given a cell $c \in S^i$, where $S^i \in \mathbf{S}$, executing UPDATEV on c extracts the safe-patch S^i from $\Delta\mathcal{S}_k$ and generates its associated node.*

Proof. Let $A(\varphi)$ be the safe-patch generated according to Lemma 2 where φ is the abstracting node. Then, $c \in A(\varphi)$ and $ts(c) = k$. Also $\forall c' \in A(\varphi), ts(c') = k$. Since, $\forall p \in \Delta\mathcal{S}_k, ts(p) = k$ and $c \in S^i$, it is deduced that $A(\varphi) \subseteq \Delta\mathcal{S}_k$ and $A(\varphi) = S^i$ where S^i is a disjoint connected grid cell-set (i.e. safe-patch). $\square$

Corollary 2. *Executing UPDATEV for one cell $c \in S^i$ for each $S^i \in \mathbf{S}$ generates the set Υ'_k*

The Alg. 8 that generates the set Υ'_k is developed based on the above results. The input to the algorithm is the set of newly generated safe cells $\Delta\mathcal{S}_k$. The for-loop in line 2 iterates for each newly added safe cell c . UPDATEV is executed in line 4 only if a node in G_k is not associated with the cell c , which indicates that cell c has not yet been extracted to a safe-patch. According to Lemma 4, a single execution of UPDATEV with $c \in S^i$ extracts the set S^i and generates its associated node in V_k . Hence, all the subsequent calls to the check in line 3 for cells $c' \in \{S^i \setminus \{c\}\}$ fails, thus do not execute the UPDATEV, as expected from the result of Lemma 4.

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

Algorithm 8 Insert-New-Nodes($\Delta\mathcal{S}_k$)

```

1:  $\Upsilon'_k \leftarrow \phi$ 
2: for all  $c \in \Delta\mathcal{S}_k$  do
3:   if  $\mathcal{V}(c) = \text{NIL}$  then
4:      $(\varphi, \mathcal{N}^\varphi) \leftarrow \text{UPDATEV}(c, \text{NIL})$ 
5:      $e \leftarrow \phi$ 
6:      $\Upsilon' \leftarrow \Upsilon' \cup \{\varphi\}$ 
7:     for all  $n \in \mathcal{N}^\varphi$  do
8:       if  $e = \phi$  then
9:          $e \leftarrow (n, \varphi)$ 
10:      else
11:        make  $n$  and  $\varphi$  adjacent nodes
12:      end if
13:    end for
14:    if  $\mathcal{V}(c_k^*) = \varphi$  then
15:       $\text{visited}[\varphi] \leftarrow \text{TRUE}$ 
16:       $\text{RPC}[\varphi] \leftarrow c_k^*$ 
17:      make  $\nu$  the parent of  $\varphi$ 
18:      make  $e[1]$  and  $\varphi$  adjacent nodes
19:    else
20:       $\Phi_k \leftarrow \Phi_k \cup \{e\}$ 
21:    end if
22:  end if
23: end for

```

Hence, it can be seen that `UPDATEV` is executed once for each $S^i \in \mathbf{S}$. Therefore, according to Corollary 2 the algorithm generates the set Υ'_k .

In line 9, the edge connecting φ to one of the neighboring nodes of φ is selected greedily (i.e. the first neighboring node that is evaluated). All the other neighboring nodes are made adjacent nodes of new node φ . If the newly generated node is a visited node (check in line 14), it is marked visited, its *RPC* variable is updated with c_k^* and is added to the reachable connected component using the operations in lines 15-17. If φ is visited, it no longer requires the edge e to connect it to a different node as it is already connected to ν . Therefore, the node at the other end of the edge e is added as an adjacent node of φ in line 18. If φ is not visited, the selected edge e is added to the set Φ_k in line 20. This set stores one edge for each newly generated node $\varphi \in \Upsilon'_k$ that can connect φ to one of the nodes in $\{V_{k-1} \cup \hat{\Upsilon}_k \setminus \Upsilon_k^s\}$. The motivation behind the population of this set is described in the next section.

3.2.3.6 Attaching newly created nodes to existing spanning trees

Consider the set $\Upsilon_k = \Upsilon'_k \setminus \Upsilon_k^V$ (given Υ_k^V to be the visited nodes in Υ'_k) of newly generated nodes that are not marked as visited by the robot. The following theorem

3.2 Incremental detection of reachability

states how a new node generated from $\Delta\mathcal{S}_k$ can be connected to a spanning tree in $\dot{\mathcal{T}}_k$.

Theorem 4. *Consider a node $\varphi \in \Upsilon_k$ such that φ belongs to some connected component C_k^i in G_k and let $\dot{\mathcal{T}}_k^i \subseteq \dot{\mathcal{T}}_k$ be the collection of trees that contain the node set $C_k^i \cap \{V_{k-1} \cup \hat{\Upsilon}_k \setminus \Upsilon_k^s\}$. Then, $\exists$ a node $n \in \mathcal{N}^\varphi$ s.t. $n \in \text{nodes}[\dot{\mathcal{T}}_k^i]$.*

Proof. Consider, for the purpose of contradiction, that $\nexists$ a node $n \in \mathcal{N}^\varphi$ s.t. $n \in \text{nodes}[\dot{\mathcal{T}}_k^i]$. Therefore, $\forall n \in \mathcal{N}^\varphi, n \notin \text{nodes}[\dot{\mathcal{T}}_k^i]$. Since φ belongs to C_k^i , all the neighboring nodes of φ also belong to C_k^i . Since for all $n \in \mathcal{N}^\varphi, n \notin \text{nodes}[\dot{\mathcal{T}}_k^i] \Rightarrow n \in \Upsilon_k$. Therefore, for any $c_1 \in A(n)$ and $c_2 \in A(\varphi)$, $ts(c_1) = ts(c_2) = k$, since both $n, \varphi \in \Upsilon_k$ for any $n \in \mathcal{N}^\varphi$. Then according to Theorem 2, n and φ cannot be two distinct nodes, which leads to a contradiction. $\square$

Hence, according to the above theorem, each newly generated non-visited node $\varphi \in \Upsilon_k$ that gets added to an existing connected component from G_{k-1} can be attached to a node in an existing spanning in $\dot{\mathcal{T}}_k$. For such nodes a connecting edge can be found during the generation of these nodes. The set Φ_k populated in Alg. 8 contains a unique edge for each such node in Υ_k that can be connected to some existing connected component.

The Alg. 9 uses the above result to iterate through each edge in Φ_k to attach the new nodes to existing spanning trees in order to generate the collection of trees $\ddot{\mathcal{T}}_k$. Consider a tree $\dot{\mathcal{T}}_k^i \in \dot{\mathcal{T}}_k$. The augmented tree after adding new nodes according to Alg. 9 is denoted as $\ddot{\mathcal{T}}_k^i \in \ddot{\mathcal{T}}_k$.

Algorithm 9 Attach-New-Nodes-To-Spanning-Trees(Φ_k)

```

1: for  $i \leftarrow 1 \dots |\Phi|$  do
2:    $(n, \varphi) \leftarrow \Phi[i]$ 
3:   make  $n$  parent of  $\varphi$ 
4: end for

```

3.2.3.7 Representing isolated nodes in separate spanning trees

Given the set $\Upsilon_k^I = \Upsilon_k \setminus \text{nodes}[\ddot{\mathcal{T}}_k]$ of nodes that represent the newly generated nodes which are not connected to any spanning tree in $\ddot{\mathcal{T}}_k$, it can be easily deduced that for all $\gamma \in \Upsilon_k^I, \mathcal{N}^\gamma = \phi$, (i.e. all nodes γ are isolated nodes with no neighboring

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

nodes). Hence, they can be added to $\ddot{\mathcal{T}}_k$ trivially by making them the roots of their own spanning trees.

3.2.3.8 Merging trees in $\ddot{\mathcal{T}}_k$ whose nodes belong to the same connected component

New nodes in $\hat{\Upsilon}_k$ and Υ'_k are connected to one of neighboring nodes greedily and do not guarantee the connectedness of the connected components in G_k . Therefore some of the trees in $\ddot{\mathcal{T}}_k$ represent only subsets of nodes in connected components of G_k . Therefore, trees in $\ddot{\mathcal{T}}_k$ must be checked for their connectedness and merged to generate the final set of spanning trees $\mathcal{T}_k$.

Alg. 9 inserts new nodes only to a single tree and does not check newly added nodes connecting two previously disconnected connected components. An example of such a situation is illustrated in Fig. 3.10(b). Consider, $\varphi \in \Upsilon'_k$ be a node attached to $\dot{\mathcal{T}}_k^i$ during spanning tree augmentation step in Alg. 9. Any new connectivity of the corresponding tree $\ddot{\mathcal{T}}_k^i$ to another spanning tree $\ddot{\mathcal{T}}_k^j \in \ddot{\mathcal{T}}_k$ through φ can be identified by checking the adjacency information $\Lambda(\varphi)$.

The Alg. 10 iterates through all the nodes $\varphi \in \Upsilon'_k$ to identify other spanning trees connected to φ 's spanning tree. For each node φ , it iterates through all its adjacent nodes in $\Lambda(\varphi)$ in the loop at line 3. If the root nodes of the spanning trees of nodes φ and $a \in \Lambda(\varphi)$ are different (checks at lines 5, 7, 9), then the spanning trees can be merged. If one of the spanning trees represents the reachable connected component, then the other spanning tree is re-rooted and merged to the first one (lines 6 & 8). If neither of the spanning trees represent the reachable connected component, the spanning tree with the least height is merged to the other tree, lines 9-13, so the re-rooting operation can be executed quickly on the tree with the least height. The function `MERGETOTREE( $a, b$ )` merges the spanning tree containing node b to spanning tree containing node a by adding the edge (a, b) after making b the root of its spanning tree.

A spanning tree $\mathcal{T}_{k-1}^i \in \mathcal{T}_{k-1}$ could be split to several sub-trees when a node is replaced by several non-connected nodes during a splitting of a safe-patch (Sec. 3.2.3.3). However the resulting sub-trees could be connected through some other

3.2 Incremental detection of reachability

Algorithm 10 Connect-Spanning-Trees-Through-New-Nodes(Υ'_k)

```

1: for all  $\varphi \in \Upsilon'_k$  do
2:    $(r^\varphi, h^\varphi) \leftarrow \text{GETROOT}(\varphi)$ 
3:   for all  $a \in \Lambda(\varphi)$  do
4:      $(r^a, h^a) \leftarrow \text{GETROOT}(a)$ 
5:     if  $r^\varphi \equiv \nu$  and  $r^\varphi \neq r^a$  then
6:        $\text{MERGETOTREE}(\varphi, a)$ 
7:     else if  $r^a \equiv \nu$  and  $r^a \neq r^\varphi$  then
8:        $\text{MERGETOTREE}(a, \varphi)$ 
9:     else if  $r^\varphi \neq r^a$  then
10:      if  $h^a > h^\varphi$  then
11:         $\text{MERGETOTREE}(a, \varphi)$ 
12:      else
13:         $\text{MERGETOTREE}(\varphi, a)$ 
14:      end if
15:    end if
16:  end for
17: end for
 $\text{MERGETOTREE}(\mathbf{a}, \mathbf{b})$ 
1: Make  $b$  the root of it's tree
2: Make  $a$  parent of  $b$ 

```

nodes which are not in Υ'_k . Therefore these sub-trees need to be checked for merging with other trees through nodes not in Υ'_k , after Alg. 10 has failed to merge them.

Theorem 5. Assume $\mathcal{T}_{k-1}$ contains the spanning trees representing the connected components of G_k . Let $\dot{\mathcal{T}}_k^i$ and $\dot{\mathcal{T}}_k^j$ be two distinct trees in $\dot{\mathcal{T}}_k$ s.t. $\forall \varphi \in \Upsilon'_k, \mathcal{N}^\varphi \cap \{\text{nodes}[\dot{\mathcal{T}}_k^i] \cup \text{nodes}[\dot{\mathcal{T}}_k^j]\} = \emptyset$ and $\exists$ some $e = (\varphi_k, \varphi_l) \in E_k$ s.t. $\varphi_k \in \text{nodes}[\dot{\mathcal{T}}_k^i]$ and $\varphi_l \in \text{nodes}[\dot{\mathcal{T}}_k^j]$. Then at least one of the nodes r^i, r^j is in the set $\aleph_k$, where r^i and r^j are the root nodes of $\dot{\mathcal{T}}_k^i$ and $\dot{\mathcal{T}}_k^j$ respectively.

Proof. Since none of the nodes in Υ'_k connect $\dot{\mathcal{T}}_k^i, \dot{\mathcal{T}}_k^j$ together, then both φ_k and φ_l must be in $\{V_{k-1} \cup \hat{\Upsilon}_k \setminus \Upsilon_k^s\}$. From Lemma 3 it can be deduced that at most only one of φ_k, φ_l can be in $\hat{\Upsilon}_k$. a) If both φ_k and $\varphi_l \in \{V_{k-1} \setminus \{\hat{\Upsilon}_k \cup \Upsilon_k^s\}\}$, then φ_k, φ_l must be in the same tree $\mathcal{T}_{k-1}^m \in \mathcal{T}_{k-1}$, based on the assumption of $\mathcal{T}_{k-1}$ representing all connected components of G_{k-1} . Then φ_k and φ_l can be nodes of distinct $\dot{\mathcal{T}}_k^i, \dot{\mathcal{T}}_k^j$ only through safe-patch splitting. Since the two trees are distinct, according to rules 5 and 6 in Sec. 3.2.3.3, at least one of r^i or r^j must be in $\aleph_k$. b) Without lost of generality, assuming only $\varphi_k \in \hat{\Upsilon}_k$, then $r^i \neq r^j$, therefore according to rule 6 in Sec. 3.2.3.3, r^i must be in $\aleph_k$. $\square$

Theorem 5 leads to the result that, if two trees in $\ddot{\mathcal{T}}_k$ are directly connected through two nodes in $\{V_{k-1} \cup \hat{\Upsilon}_k \setminus \Upsilon_k^s\}$, then the connecting edge can be retrieved

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

by examining the trees stored in the set $\aleph_k$. Therefore after executing Alg. 10 iterating through the nodes of trees in $\aleph_k$ to check for possible tree connectedness ensures generation of the set of spanning trees $\mathcal{T}_k$ representing connected components of G_k , given $\mathcal{T}_{k-1}$.

The Alg. 11 details this operation. It iterates through all the nodes of trees identified in $\aleph$ and merges all the spanning trees in $\ddot{\mathcal{T}}_k$ that are connected through these old nodes. Initially, all the nodes of the trees identified in $\aleph_k$ are extracted in lines 1-3. For each node φ of each identified tree, its adjacent nodes $\Lambda(\varphi)$ are checked in line 10 for connectivity to a different spanning tree. If a connecting edge is not already found between the two trees, then the newly found connecting edge is added to set $\mathcal{R}^i$. After iterating through all the nodes of a spanning tree i , $\mathcal{R}^i$ contains the edges connecting spanning tree i to other spanning trees and are used for merging in lines 15-17. At the end of this step, the collection of spanning trees $\mathcal{T}_k$ contains the connectivity information among safe-patches after update step k .

Given all the trees in $\mathcal{T}_1$ at first map update step are valid spanning trees representing connected components of G_1 , it is deduced that the provided operations incrementally maintain the spanning trees representing connected components of G_k for each k and can be used to check the reachability of the extracted mapped-free-space boundary cells. This is then used to retrieve only the reachable frontiers according to Theorem 1.

Algorithm 11 Connect-Spanning-Trees-Through-Old-Nodes($\aleph$)

```

1: for all  $r \in \aleph$  do
2:    $\mathbf{V}^r \leftarrow \text{EXTRACTNODES}(r)$ 
3:    $\mathbf{V} \leftarrow \mathbf{V} \cup \{\mathbf{V}^r\}$ 
4: end for
5: for  $i \leftarrow 1 \dots |\mathbf{V}|$  do
6:    $\mathcal{R} \leftarrow \phi$ 
7:   for all  $\varphi \in \mathbf{V}[i]$  do
8:     for all  $n \in \Lambda(\varphi)$  do
9:        $(r^n, h^n) \leftarrow \text{GETROOT}(n)$ 
10:      if  $r^n \neq \aleph[i]$  and  $\nexists(u, w)$  s.t.  $r^w = r^n$  then
11:         $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\varphi, n)\}$ 
12:      end if
13:    end for
14:  end for
15:  for all  $e \in \mathcal{R}$  do
16:     $\text{MERGETOTREE}(e[1], e[2])$ 
17:  end for
18: end for

```

Figure 3.12: Comparison of safe-patch sizes generated due to (a) straight line motion and (b) curved motion.

3.2.3.9 Computational complexity of the algorithms

The computational complexity of the UPDATEV function is dependant on the maximum possible size of the safe-patches generated. Considering δm to be the map resolution and δd to be the avg. change in robot's position between to update steps, the maximum possible safe-patch size in number of grid cells can be computed as follows.

Given safe-patches are generated at each δd motion of the robot, the maximum safe-patch size is the maximum number of new safe-grid cells generated during a robot's δd motion. It can be trivially understood that the number of newly generated safe-cells are maximized when the robot follows a straight line trajectory. If the robot follows a curved trajectory, the robot would remap section of already mapped space resulting in the reduction of the newly generated safe cells as illustrated by the Fig. 3.12.

Therefore the maximum possible safe-patch size is given by,

$$\begin{aligned}
 &= \pi \frac{R_{max}^2}{2} + \delta d 2R_{max} - \pi \frac{R_{max}^2}{2} \\
 &= 2\delta d R_{max}
 \end{aligned}$$

Hence the maximum size of a safe-patch in grid cells is given by $2\delta d R_{max} / \delta m^2$.

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

In the Alg. 7, the worst case complexity of processing updated safe-patches (lines 1-8) due to obstacle information includes iterating through each element in Ξ_k and generating a single safe-patch for each such element. Hence the complexity is given by $\mathcal{O}(2|\Xi_k|\delta dR_{max}/\delta m^2)$. The complexity of executing the node connectivity Rules 1-6 in Sec. 3.2.3.3 is given by $\mathcal{O}(deg_{max}(G_k))$ where $deg_{max}(G_k)$ is the maximum node degree in graph G_k and $deg_{max}(G_k) \ll \delta dR_{max}/\delta m^s$. The root extraction operations in line 36 also has a complexity of $\mathcal{O}(s_u + 1)$ (Sec. 3.2.2) and $s_u \ll \delta dR_{max}/\delta m^2$. Therefore the complexity of Alg. 7 is given by $\mathcal{O}(2|\Xi_k|\delta dR_{max}/\delta m^2)$.

Considering Alg. 8 generates $|\mathbf{S}|$ number of new safe-patches, the complexity can be given as $\mathcal{O}(2|\mathbf{S}||\Xi_k|\delta dR_{max}/\delta m^2)$. However considering that only the newly generated safe cells $\Delta\mathcal{S}_k$ is processed to generate the new safe-patches, the complexity of Alg. 8 can be simplified to $\mathcal{O}(|\Delta\mathcal{S}_k|)$. Alg. 9 uses the resulting Φ_k from Alg. 8 to attach non-visited new nodes in Υ_k to existing trees in $\dot{\mathcal{T}}_k$ hence its complexity is given by $\mathcal{O}(|\Phi_k|)$.

Since maximum branching factor of trees is less than $deg_{max}(G_k)$ and the root extraction operations (in lines 2, 4) and tree merging operations have complexities of $\mathcal{O}(s_u + 1)$, the complexity of Alg. 10 is given by $\mathcal{O}(|\Upsilon'_k|deg_{max}(G_k)(s_u + 1))$. Similar to Alg. 10, the complexity of Alg. 11 is given by $\mathcal{O}(|V|deg_{max}(G_k)(s_u + 1))$.

Therefore the total complexity of the incremental reachability analysis algorithms is give by,

$$\begin{aligned}
 &= \mathcal{O}(2|\Xi_k|\delta dR_{max}/\delta m^2) + \mathcal{O}(|\Delta\mathcal{S}_k|) + \mathcal{O}(|\Phi_k|) + \\
 &\quad \mathcal{O}(|\Upsilon'_k|deg_{max}(G_k)(s_u + 1)) + \mathcal{O}(|V|deg_{max}(G_k)(s_u + 1)) \\
 &= \mathcal{O}(2|\Xi_k|\delta dR_{max}/\delta m^2) + \mathcal{O}(|\Delta\mathcal{S}_k|) + \mathcal{O}(|\Phi_k|) + \\
 &\quad \mathcal{O}((|\Upsilon'_k| + |V|)deg_{max}(G_k)(s_u + 1))
 \end{aligned}$$

3.3 Experimental results

The proposed work flow of algorithms to incrementally generate the safe and reachable cells during exploration missions is validated using real world experiments in two environments. The environments used for the experiments are depicted in Figs.

Figure 3.13: (a) Map of the IRL environment generated after an exploration mission.
(b) View of the IRL environment from V1 viewpoint

3.13 and 3.14. The IRL environment is a typical indoor lab setting with narrow walkways, shiny surfaces (e.g. glass doors) and cluttered obstacles. ARRL environment illustrates a hybrid environment that contains both indoor and outdoor elements. The indoor section of the environment contains two typical office rooms. The outdoor section of the environment contains several corridors leading to more open spaces and vegetation that is visible to the sensor (e.g. bushes).

Real world data is gathered by tele-operating a Pioneer 3 AT robot equipped with a Hokuyo Laser Range Finder (depicted in Fig. 3.15) and running ROS navigation suite[77]. The localization of the robot is achieved by executing a popular SLAM implementation, GMapping[34]. We used our own implementation of occupancy grid mapping algorithm in [14] with a maximum sensor range set to 4m. This combined mapping approach is identical to systems that use a separate feature based SLAM component for precision localization and a separate grid based mapping component used for motion planning. The occupancy grid maps produced in this manner are generally less accurate than integrated grid-based SLAM solutions. However, this provides a valuable opportunity to evaluate the performance of the proposed algorithm in all types of environments including challenging environments with less accurate map information. For all situations, map update is triggered when the robot has moved over 0.5m or rotated over 0.1 radians. The resolution of the grid cells in the map is set to 0.05 m. Data for the indoor type environment “IRL” contains 103 map update steps and the hybrid environment type “ARRL”

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

Figure 3.14: (a) Map of the ARRL environment generated after the exploration mission. (b)-(d) View of the ARRL environment from the three viewpoints V_1, V_2, V_3 marked in (a)

Figure 3.15: P3AT robot equipped with a Hokuyo LRF used during experiments

3.3 Experimental results

Map & Method		Update Time(ms)			# of update cells		
		Mean	Min	Max	Mean	Min	Max
IRL-800x800	Proposed	5.57	2.65	10.09	1203	128	2566
	TRA	84.55	28.35	127.40	667045	641658	684889
IRL-1000x1000	Proposed	6.91	3.72	10.19	1200	114	2564
	TRA	99.26	42.37	150.57	1026361	1001658	1043934
IRL-1200x1200	Proposed	8.75	5.08	15.62	1175	214	2662
	TRA	115.49	58.68	158.21	1466813	1441658	1484722
ARRL-1200x1220	Proposed	12.33	5.12	40.58	942	0	3903
	TRA	227.40	60.02	369.24	1495816	1441566	1539861
ARRL-1600x1600	Proposed	17.11	9.36	33.01	947	0	3515
	TRA	269.43	102.69	411.91	2615765	2561523	2660035

Table 3.1: Comparison of efficiency between traditional and proposed method in generating $\mathcal{J}_k$

contains 315 map update steps. The algorithms implemented using C++ with the standard ROS and Linux libraries and are executed on a computer with Intel Core 2 Duo 3GHz processor and 4GB of RAM running Ubuntu 12.04.

3.3.1 Evaluation of efficiency of incremental obstacle inflation

The execution time and the number of grid cells processed at each update step to generate the inflated obstacles $\mathcal{J}_k$ is evaluated. The performance of the proposed method is compared with the traditional (TRA) approach of generating the set $\mathcal{J}_k$, which is described in Sec. 2.2.2. The effect of the map size on the two compared methods is examined using different map sizes for the two environments. For IRL, three maps with dimensions of 800x800, 1000x1000 and 1200x1200 grid cells were used. Similarly, for ARRL, maps with dimensions of 1200x1200 and 1600x1600 grid cells were used. Tab. 3.1 summarizes the performance comparison of the two methods.

It is evident that in all situations the proposed method is capable of extracting inflated obstacle information with the minimum processing required compared to the traditional approach by providing a bound on the number of cells processed, due to the processing of only locally updated cell information. In addition, during

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

exploration missions in the ARRL environment, there are instances where the robot ventures into completely open spaces (with relative to the sensor’s maximum sensor range) which results in non-detection of any new obstacle information. Therefore in these situations, the incremental inflated obstacle extraction algorithm does not process any grid cell. This is depicted by the minimum number of processed cells for the ARRL environment being reported as zero. Fig. 3.16 depicts the different portions of grid cells processed for the three main tasks of deflation, repair and inflation in the proposed incremental inflation algorithm throughout exploration missions, and verify that the total processing throughout the exploration mission is bounded.

Figure 3.16: Total number of cells processed for each deflation, repair and inflation tasks throughout the mission

3.3.2 Evaluation of efficiency of reachability computation

The proposed method for extracting reachable cells from the map is compared with the traditional approach (TRA). In the TRA approach, reachability of the grid cells is computed by executing a flood-fill operation on the grid from the robot’s current pose grid cell. This operation is dependant on the number of currently mapped grid cells as the flood-fill operation terminates when it has reached unmapped grid cells. Therefore, the execution time does not depend on the dimensions of the occupancy grid map. Figs. 3.17 and 3.18 illustrate number of processed cells and execution times for both IRL and ARRL environment in extracting the reachability

3.3 Experimental results

information. They illustrate that the TRA approach continues to process all the mapped grid cells resulting in an increase in the execution time as the exploration mission progresses. In comparison, the proposed approach processes only the locally updated map data to incrementally update the safe-patch graph that maintains the connectivity information of the grid cells. This is corroborated by the bounded processing and execution times reported in Figs. 3.17 and 3.18.

Figure 3.17: Evaluation of efficiency of reachability computation for IRL environment

Figure 3.18: Evaluation of efficiency of reachability computation for ARRL environment

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

3.3.3 The search depth of reachability graph

Retrieving the reachability of the cells in the occupancy grid map requires searching the spanning tree proposed in Sec. 3.2.2 and it is important to minimize the number of search steps involved. Cells abstracted by visited nodes in G_k require only one search step whereas, reachable non-visited cells require multiple search steps and the theoretical bounds on the depths of associated nodes are given in Sec. 3.2.2.1. However, these bounds are not explicitly enforced during the update algorithm as it requires additional operations to update depths of each node in the tree. Therefore, the search depths of the trees generated without the node depth checks are recorded during experiments to evaluate their conformance to the theoretical values. For both environments, the average number of search steps and the standard deviation to check for reachability for each cell in reachable non-visited safe-patches are recorded. This is iterated for exploration missions with three different maximum sensor range values of 3m, 4m and 5m that provide theoretical search step bounds of 7, 9 and 11 respectively, considering $\delta d = 0.5\text{m}$.

Figs. 3.19 and 3.20 depict the percentage of reachable non-visited cells, out of mapped cells, at each map update step for the three different sensor range values for both environments. The considerable percentages of non-visited reachable cells present throughout the missions confirm the requirement to minimize the search steps used for reachability check. Fig. 3.19 illustrates that the progression of the percentage of non-visited reachable cells for both missions with 4m and 5m are identical to each other. This is due to the narrow nature of the room type IRL environment where increasing the maximum sensor range beyond 4m does not increase the mapping capability.

It is observed in Fig. 3.21 and 3.22 that for both environments and the three different sensor range values, the average number of search steps and its deviation are within the theoretical bounds of 7, 9 and 11 respectively even without explicitly enforcing them in the update algorithms. It can also be seen that the standard deviations continue to increase with the increase in the maximum sensor range. As the sensor range increases, the possibilities of connecting safe-patches to visited patches increases, thus inducing more variability on the search depth. Overall, the results confirm that the proposed incremental graph data structure is efficient

Figure 3.19: Comparison of the percentage of non-visited reachable grid cells, across map update steps, in occupancy grid maps generated with 3m, 4m and 5m sensor ranges for IRL environment

Figure 3.20: Comparison of the percentage of non-visited reachable grid cells, across map update steps, in occupancy grid maps generated with 3m, 4m and 5m sensor ranges for ARRL environment

at identifying the reachability of grid cells and provide a bound on the execution times.

3.4 Summary

This chapter addressed the issue of extracting the safe and reachable grid cells using efficient incremental update procedures. Modified Dynamic Brushfire algorithm is used to incrementally update the safe grid cell information across map updates. This information is then used to incrementally extract the reachability information. Notion of safe patch is introduced to refer to a contiguous traversable region in the

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

(a) Range 3m

(b) Range 4m

(c) Range 5m

Figure 3.21: Progression of the avg. no. of search steps for reachability check for IRL environment

(a) Range 3m

(b) Range 4m

(c) Range 5m

Figure 3.22: Progression of the avg. no. of search steps for reachability check for ARRL environment

3. INCREMENTAL EXTRACTION OF SAFE AND REACHABLE MAP INFORMATION

map that is created at a single update step. Connectedness of these safe patches created during different map update steps are abstracted to a graph data structure and is managed throughout exploration missions using a forest of spanning trees. Accurate incremental update procedures for the spanning trees to store the reachability information is derived and are reported as algorithms.

Experimental results indicate that the proposed incremental approach is capable of extracting safe and reachable cells from the occupancy grid map more efficiently compared to the traditional approach and provides a bound on the total processing required. The next chapter utilizes the proposed safe and reachable cell extraction method to develop an efficient incremental frontier cell extraction procedure.

Chapter 4

Incremental Safe and Reachable Frontier Detector

The previous chapter described a set of incremental update procedures to mark safe and reachable grid cells in an occupancy grid map and retrieve these information at each map update step in an efficient manner. Extracting valid frontier cells require the safe and reachability information about the grid cells in the map, as only the mapped-free-space boundary cells that are safe and reachable are reported as frontier cells according to the Definition 15 given in Sec. 2.1.2. The reachability of mapped-free-space boundary cells could change during an exploration mission as changes to the mapped space could result in previously non-reachable sections of the map becoming reachable and vice versa. Therefore, mapped-free-space boundary information must be managed separately throughout exploration missions irrespective of their individual reachability property at different map update steps. This chapter describes incremental extraction of mapped-free-space boundary cells as contours and management of all the extracted contours during the exploration mission efficiently using a new spatial data structure at each map update step. Incremental safe and reachability information extraction proposed in the previous chapter is then combined with the incremental mapped-free-space boundary extraction presented in this chapter to provide a unified incremental frontier extraction approach named Safe and Reachable Frontier Detector (SRFD).

This chapter is organized as follows. First, incremental extraction of mapped-

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

free-space boundary information as contours is presented in Sec. 4.1. Next, a spatial data structure that supports storing and updating of individual contours made up of 2D points is presented in Sec. 4.2 as the contour database. The spatial data structure is based on standard MX-Quadtree data structure. Complexity analysis of access and update (insert and delete) operations are provided to illustrate the operational efficiency of the contour database during frontier extraction process. Sec. 4.3 details the unification of the proposed reachability extraction and mapped-free-space boundary management to extract valid frontier cell information in a computationally efficient manner during each map update step. Finally, the proposed SRFD method is compared for its efficiency and accuracy with the traditional frontier extraction method (TRA) and several competing incremental frontier extraction strategies in Sec. 4.4 using real world experimental data.

4.1 Incremental extraction of mapped-free-space boundary cells as contours

All the newly updated cells in U_k (see Sec. 2.1.3) that satisfy the condition of being a boundary cell according to the Definition 14 are extracted to the set $\mathcal{B}_k^{new}$ as newly generated boundary cells using the safe-area information generated according to Sec. 3.1. The Definition 14 uses only the 4-connected adjacent cells (i.e. $\mathcal{N}_4(c)$) when extracting the set $\mathcal{B}_k^{new}$. This is done to ensure that only single grid cell thick boundary contours are extracted and stored in the boundary cell database. In addition to simplifying the contour extraction procedure, the reduction in the number of reported boundary cells improves the memory efficiency of the frontier extraction process. Fig. 4.1(a) illustrates extraction of boundary cells using 8-connectivity and Fig. 4.1(b) illustrates the grid cells that are extracted as boundary cells using 4-connectivity. The grid cells that are omitted from being extracted as boundary cells, that would have been extracted had 8-connectivity been used, is marked by cross signs. Comparison between the two shows that the 4-connectivity based boundary cell extraction allows generating of single cell thick contours.

However, not all cells extracted to the set $\mathcal{B}_k^{new}$ represent valid mapped-free-space boundary cells due to errors in the generated map. Therefore, erroneously

4.1 Incremental extraction of mapped-free-space boundary cells as contours

Figure 4.1: The check marks indicate extracted boundary cells in an example occupancy grid map (a) When using 8-connectivity, the extracted boundaries have a two grid cell thickness (b) Using 4-connectivity ensures single cell thickness for extracted boundary contours. The cross marks indicate grid cells that are not extracted but would have been extracted if 8-connectivity is used.

generated boundary cells must be filtered from $\mathcal{B}_k^{new}$ before they are considered for extracting boundary contours.

4.1.1 Filtering erroneous boundary cells from $\mathcal{B}_k^{new}$

Sensor errors could result in erroneously introducing isolated free grid cells to the occupancy grid map. These isolated free grid cells result in isolated boundary cells that get extracted to the set $\mathcal{B}_k^{new}$. Isolated boundary cells are grid cells that satisfy the Definition 14, but cells that do not bound any other mapped-free cells in the occupancy grid map. Fig. 4.2(a) illustrates the simplest occurrence of isolated boundary cells. The two free grid cells (white/unshaded) are surrounded by unknown cells. Both of these cells become mapped-free-space boundary cells. However they do not bound any other free space, hence are erroneous boundary cells.

Fig. 4.2(b) illustrates another example of a set of grid cells that get extracted as boundary cells, but do not bound any other free cells. All the free cells are extracted as boundary cells. This happens when thin free lines of grid cells are introduced due to mapping errors. For example, certain surfaces do not reflect back or deflect laser beams with certain incident angles. This results in LRF reporting incorrect

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

Figure 4.2: (a) Isolated boundary cells that do not bound any safe region (b) Occurrence of thin lines in the occupancy grid map due to incorrect mapping

distance values thus the mapping component incorrectly introducing thin mapped free lines to the occupancy grid map.

These erroneous boundary cells can be filtered by considering the 8-connected neighbors $\mathcal{N}_8(c)$ of each reported boundary cell in $\mathcal{B}_k^{new}$. Each reported cell in $\mathcal{B}_k^{new}$ is extracted to be a valid boundary cell only if at least one of its 8-connected neighboring cells is free and is not an existing boundary cell, as defined below.

Definition 20. $\hat{\mathcal{B}}_k^{new} = \{c : c \in \mathcal{B}_k^{new} \text{ s.t. } \exists c' \in \{\mathcal{N}_8(c) \cap \mathcal{S}_k\} \setminus \{\mathcal{B}_{k-1} \cup \mathcal{B}_k^{new}\}\}$ where $\mathcal{B}_{k-1}$ is the set of free-space-boundary cells after step $k - 1$.

Therefore, the set of free-space-boundary cells can be defined incrementally as,

$$\mathcal{B}_k = \bigcup_{i=1 \dots k} \hat{\mathcal{B}}_i^{new}$$

Fig. 4.3 illustrates the effect of the filtering operation on extracting the set $\hat{\mathcal{B}}_k^{new}$. Figs. 4.3(a) and 4.3(b) illustrate extraction of newly generated boundary cells at two subsequent map update steps. The new boundary cells extracted are depicted in red. The yellow rectangle marks an erroneously mapped thin line of free cells and shows that this line is filtered from the set $\hat{\mathcal{B}}_k^{new}$.

While the resulting set $\hat{\mathcal{B}}_k^{new}$ of new boundary cells can be treated individually for storage and management across map update steps, the connectedness of the

4.1 Incremental extraction of mapped-free-space boundary cells as contours

Figure 4.3: (a) Newly detected boundary cells at update step k , marked in red color. (b) Newly detected boundary cells at update step $k+1$. The area marked by the yellow box indicates a thin line of free cells generated, due to some error in mapping. The cells in this thin line are filtered out from being detected as new boundary cells

boundary cells to each other to create contours, can be utilized for efficient update operations on the database that stores boundary cell information.

4.1.2 Chaining new boundary cells to contours

Chaining of the new boundary cells allows to represent the partial contours of the boundaries of mapped-free-spaces in the occupancy grid map. These contours are formally defined as follows.

Definition 21. A partial boundary contour C extracted from $\hat{\mathcal{B}}_k^{new}$ is an ordered set of mapped-free-space boundary cells with the following properties.

1. For each two subsequent cells a, b in the ordered set C , $a \in \mathcal{R}(b)$ and $b \in \mathcal{R}(a)$
2. For each two subsequent cells a, b in the ordered set C , the set $\{\mathcal{R}(a) \cap \mathcal{R}(b)\} \setminus \mathcal{B}_k$ is non-empty.

Property 1 ensures the connectivity of the elements in the contour and Property 2 ensures that each extracted partial contour is part of a boundary of a single contiguous safe region in the occupancy grid map. Given that the single cell thickness of boundary cells are guaranteed (Sec. 4.1) and that the above definition restricts

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

contour cells to bound a single contiguous safe region, a simple neighboring-cell chaining algorithm similar to the ones found in image processing literature [36] is utilized to extract contours from the identified boundary cells.

The input to the contour extraction operation is the set $\hat{\mathcal{B}}_k^{new}$. All of the existing cell chaining algorithms are based on flood-fill [36] type operations, hence require seed points. Since the contours are not known a priori, all cells in $\hat{\mathcal{B}}_k^{new}$ must be considered for being a seed cell. This results in the basic contour extraction procedure given in Alg. 12. The check in line. 3 ensures that only the cells in $\hat{\mathcal{B}}_k^{new}$ that are not already included in any extracted contour are used as seed cells to extract a new contour. $C_{\mathcal{B},k}^{new}$ is the resultant set of newly extracted boundary contours.

Algorithm 12 Extract Contours

```

1:  $C_{\mathcal{B},k}^{new} \leftarrow \phi$ 
2: for all  $b \in \hat{\mathcal{B}}_k^{new}$  do
3:   if  $b \notin \text{cells}[C_{\mathcal{B},k}^{new}]$  then
4:      $C \leftarrow \text{EXTRACTCONTOUR}(b)$ 
5:      $C_{\mathcal{B},k}^{new} \leftarrow C_{\mathcal{B},k}^{new} \cup \{C\}$ 
6:   end if
7: end for
```

A typical operation of extraction of a single contour is illustrated in Fig. 4.4. Any new boundary cell from $\hat{\mathcal{B}}_k^{new}$ that is not already chained to a contour can be used as the seed cell to initialize the chaining process. Then the two neighboring cells are identified to start the chaining of cells in the two opposite directions, Fig. 4.4(a). Chaining is continued until no neighboring new boundary cells are detected in both sub chains as depicted in Fig. 4.4(b). The two sub chains are merged to form a single mapped-free-space contour. The two corner cells of the extracted contour are depicted by the filled circle markings. The two corner cells of the extracted contour are checked during the contour insertion procedure (explained in Sec. 4.2.3.2) to check for merging with existing boundary contours.

Alg. 13 summarizes the contour extraction operation. First, in line. 2, the two neighboring boundary cells of seed s that initialize the chaining of the contour are retrieved. In line. 5, the algorithm extracts portion of the contour originating from the first neighboring cell of s , $P_I[1]$. If s has only one neighboring boundary cell, the extracted contour segment C_1 is returned as the complete new contour containing

4.1 Incremental extraction of mapped-free-space boundary cells as contours

Figure 4.4: (a) Contour chaining can be initialized in any boundary cell, marked with circle, and the two directions in which chaining proceeds are indicated in arrows. (b) At the termination of contour chaining, the chains of cells generated in the two directions are merged to form a single contour. The two cells with wavy lines are existing boundary cells discovered in previous steps which makes the cells identified by filled dots the two corners of the extracted contour.

Algorithm 13 Extract contour given seed cell s

```

EXTRACTCONTOUR( $s$ )
1:  $C \leftarrow \{s\}$ 
2:  $P_I \leftarrow \text{FINDCHAININITIALIZATIONCELLS}(s)$ 
3: if  $|P_I| > 0$  then
4:    $C_1 \leftarrow \text{CHAINBOUNDARYCELLS}(P_I[1])$ 
5:    $C \leftarrow C \cup C_1$ 
6: end if
7: if  $|P_I| == 2$  then
8:    $C_2 \leftarrow \text{CHAINBOUNDARYCELLS}(P_I[2])$ 
9:    $C'_2 \leftarrow \text{REVERSEELEMENTS}(C_2)$ 
10:   $C \leftarrow C'_2 \cup C \cup C_1$ 
11: end if
12: return  $C$ 

```

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

seed cell s . If there are two neighboring boundary cells, then the algorithm enters the second if block (in line 7) to generate the other portion of the contour C_2 . Since a boundary contour is an ordered set, the order of C_2 is reversed to generate C'_2 in line. 9 which in turn generates the final extracted contour C in line. 10. Sec. 4.1.2.1 and Sec. 4.1.2.2 details the operations of selecting the two neighboring boundary cells of the seed cell and how they are used to chain the boundary cells to generate the two sub-contours respectively.

4.1.2.1 Selecting the two neighboring cells of the seed cell for initial chaining

In most situations, the seed cell has only two neighboring boundary cells. Hence, they can be directly used to extract the two sub-chains going in the opposite directions, from the seed cell, to generate the complete contour as depicted in Fig. 4.4(a). However, in some occasions, boundary cells have more than 2 neighboring boundary cells. Therefore, selection of the appropriate two neighboring boundary cells of the seed cell to initialize the chaining process requires clustering and ordering of these neighboring boundary cells.

Definition 22. A cluster of neighboring boundary cells of an existing boundary cell s , denoted by C_s , is defined as a maximal connected set of cells in $\{\mathcal{N}_8(s) \cap \hat{\mathcal{B}}_k^{new}\}$ where all the cells in C_s bound the same set of contiguous safe cells in $\mathcal{S}_k \setminus \mathcal{B}_k$.

Lemma 5. *Given a boundary cell $s \in \hat{\mathcal{B}}_k^{new}$, there exists a maximum of only two clusters C_s^1, C_s^2 of neighboring boundary cells of s that bound the same contiguous safe cells in $\mathcal{S}_k \setminus \mathcal{B}_k$.*

Proof. Since cell s is a valid boundary cell, there exists at least one connected set of safe cells in $\{\mathcal{N}_8(s) \cap \mathcal{S}_k\} \setminus \mathcal{B}_k$. Let S be that set. Since $S \subseteq \mathcal{N}_8(s)$, S is bounded by two other grid cells in $\mathcal{N}_8(s)$. Since neighboring boundary clusters are also in $\mathcal{N}_8(s)$, this implies that, the set S can be bounded by a maximum of two neighboring boundary clusters. Conversely, there exists a maximum of two neighboring boundary cell clusters for set S .

□

Figure 4.5: (a) Seed cell s has only two neighboring boundary cell clusters that bound the same safe cells. The free cell to the bottom of cell s is the common safe cell set bounded by the two boundary cell clusters. (b) Seed cell has two, sets of two neighboring boundary cell clusters (i.e. striped cells to the bottom of s and striped cells to the top of s). The red/bold square marks the neighborhood of cells s , $\mathcal{N}_8(s)$

The above result indicate that clustering of boundary cells, in the neighborhood of a seed cell s , $\{\mathcal{N}_8(s) \cap \hat{\mathcal{B}}_k^{new}\}$, using the Definition. 22, allows to identify the two subset of cells the seed s must be initially connected, to extract the contour containing cell s . Fig. 4.5 illustrates the clustering of the neighboring boundary cells of a seed cell s . In both figures, striped cells depict the boundary cells included in the clusters. The individual clusters are differentiated by their stripe patterns.

The following theorem states the maximum number of grid cells possible in a boundary cell cluster extracted according to Definition. 22. This result is then used to formalize the operation that selects the next boundary cell to be added from each cluster to the chain.

Theorem 6. *Given a boundary cell $s \in \hat{\mathcal{B}}_k^{new}$ and one of its neighboring boundary cell clusters C_s extracted according to Definition 22, $|C_s| \leq 2$.*

Proof. The proof directly follows from the observations that any arrangement of three grid cells for cluster C_s in $\mathcal{N}_8(s)$ is inconsistent with the single cell thickness of boundary cell extraction, generated according to Definitions. 14,20 and 21.

□

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

Theorem 7. *Given a boundary cell $s \in \hat{\mathcal{B}}_k^{new}$, and one of its neighboring boundary cell clusters C_s with $|C_s| = 2$ where $C_s = c_1, c_2$ and $c_1 \in \mathcal{N}_4(s)$ and $c_2 \in \mathcal{N}_8(s) \setminus \mathcal{N}_4(s)$, then the valid order of the boundary cells in the extracted contours is s, c_1, c_2 .*

Proof. Consider for the purpose of contradiction that the valid order of the cells in the extracted contours is s, c_2, c_1 . Consider the coordinates of s to be (x, y) . Then without loss of generality assume cell $c_1 = (x + 1, y)$, $c_2 = (x + 1, y - 1)$ and cell $f = (x, y - 1)$ be one of the cells in $\{\mathcal{S}_k \setminus \mathcal{B}_k\}$, which is bounded by the contour containing s, c_2 and c_1 . Since c_2 is a valid boundary cell and f is a safe cell, either (a) $(x + 2, y - 1)$ must be an unknown cell or (b) $(x + 1, y - 2)$ must be an unknown cell.

- (a) If $(x + 2, y - 1)$ is an unknown cell, then the grid cells that can become a boundary cell after c_1 in the contour, is one of $(x - 1, y + 1)$, $(x, y + 1)$, $(x + 1, y + 1)$ and $(x + 1, y)$. None of these cells satisfy the property 2 of Def. 21 with cell c_1 . Hence the order s, c_1, c_1 is invalid.
- (b) Similarly, if $(x + 1, y - 2)$ is an unknown cell, then none of the remaining adjacent cells of c_1 that are not added to the contour (i.e. $\mathcal{N}_8(c_1) \setminus \{s, c_2, f\}$) satisfy the property 2 of Def. 21 with cell c_1 . Hence the order s, c_1, c_1 is invalid.

Therefore, the valid order of the cells in the contour is s, c_1, c_2 . □

Fig. 4.5(a) illustrates a situation where a cluster (the one to the right of cell s) has two cells to be considered when initializing the chaining to s . Theorem 7 indicate that the 4-connected neighboring boundary cell must be first chained to cell s . For neighboring boundary clusters with only a single cell, that cell is directly chained with the seed cell. Fig. 4.6 illustrate in red arrows the two cells that are selected to initialize the chaining process from the seed cell.

4.1.2.2 Chaining boundary cells

Once the chaining of the contour is initialized according to the previous section, the ordering of neighboring grid cells to be chained to the last element in the partially extracted contour can be done based on theorem. 7. This chaining of neighboring

Figure 4.6: Selection of the neighboring boundary cells to initialize contour chaining. S is the seed cell. The red arrows indicate the chaining of two selected cells

boundary cells is continued iteratively until no more neighboring boundary cells in $\hat{\mathcal{B}}_k^{new}$ is available, at which point the extracted partial contour is returned to be merged to a single contour in line. 5 or 10 in Alg. 13. At the end of contour extraction step, all the extracted boundary contours $C_{\mathcal{B},k}^{new}$ are added to the contour database, which is explained next.

4.2 Contour Database

The information about newly generated map boundary contours and disappeared contours, retrieved during the last update step, is used to incrementally update an in-memory contour database. This database maintains all the valid map boundary contours across update steps. The data structure that implements the database must be able to support three key operations efficiently. These are, deletion of disappeared contours, insertion of newly identified contours and extraction of the contours. All three of these operations require efficient access to the 2-dimensional spatial data of contours. Therefore spatial data structures are considered for the implementation of the contour database.

While there are spatial data structures to store and query information about properly defined line segments such as interval trees and segment trees [78], these are not suitable to maintain non-parameterized contour information of map boundary cells. Therefore a spatial data structure that is based on MX-Quadtree [79] which

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

is capable of maintaining all the cell information of valid contours is presented. Sec. 4.2.1 provides a brief summary of the MX-Quadtree before Sec. 4.2.2 details the proposed L-MX-Quadtree data-structure.

4.2.1 MX-Quadtree

MX-Quadtree (MX for Matrix) is a special type of Quadtree used to represent discrete point data. MX-Quadtree is organized in a similar way to a Quadtree where the 2D space is iteratively sub-divided to four equally sized squares, called quadrants, starting from the main square. The major difference between Quadtree, that is predominantly used to represent region data, and MX-Quadtree is that the leaf nodes in an MX-Quadtree corresponds to 1x1 squares and indicate the presence of valid discrete points/cells in the corresponding position in the 2D-grid. Therefore, an MX-Quadtree generally contains more intermediate nodes compared to traditional Quadtrees. As with any tree data structure, the computational complexity of the operations are determined based on the shape and height of the tree. It is important to note that the height of a non-empty MX-Quadtree is $\log_2(s_m)$ where s_m is the size of the (square) map in number of grid-cells and that the shape of the tree is independent of the order of insertion of cell data. Fig. 4.7 illustrates an MX-Quadtree and the grid information it represents. Reader is referred to [79] for a detailed explanation on Quadtrees, MX-Quadtrees and related data-structures.

While MX-Quadtree provides an easy way of representing all the individual cells of the valid mapped-free-space contours, MX-Quadtree, in its current form, is not an efficient data structure to store and manage contour information, due to the large number of intermediate nodes. Therefore a skip-list [80] inspired modification is introduced to generate the Linked MX-Quadtree (L-MX-Quadtree) to efficiently represent map boundary contour information and is described next.

4.2.2 Linked MX-Quadtree (L-MX-Quadtree) to store contours

In order to preserve contour information during the insertion of individual grid cells to the MX-Quadtree, the leaves in the MX-Quadtree, which represent the grid cells

Figure 4.7: A 8x8 grid, black cells indicating valid data cells, and the corresponding MX-Quadtree of the data. The solid lines in the grid indicates the boundaries of quadrants generated through iterative sub-division.

that are next to each other in the contour, are linked with each other. In addition, the two corners of each contour are stored in a list, called *corner list*, with each list item directly linking to its corresponding leaf node in the MX-Quadtree. This is depicted in the linked list below the quadtree in Fig. 4.8 and is similar to the operation of a skip-list [80]. Associating a corner-list item with a leaf node in L-MX-Quadtree provides an easy check for detecting corner nodes of stored contours.

4.2.2.1 Retrieving contours

The corner-list is used to index into nodes representing corner cells of stored contours. The rest of the cells of the contour can be retrieved by following the chained links among the nodes. When the other corner of the contour is encountered, the retrieving of the contour is terminated and the associated corner node is marked off in the corner-list so that it is not used again to retrieve the same contour twice. Considering $C_{\mathcal{B},k}$ be the set of stored boundary contours, then the complexity of retrieving the cells in contours is given by $\mathcal{O}(\sum_{C \in C_{\mathcal{B},k}} |C|)$, hence is linear in the number of contour cells.

4.2.2.2 Improving Indexing Complexity

While the corner list improves the efficiency of retrieving all the contour information from the data structure, it does not improve the execution time needed to index

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

Figure 4.8: Linking neighboring leaves and introduction of a corner list to MX-Quadtree. C_1, C_2, C_3 and C_4 are the corners of the two contours.

into a specified contour or a specified contour cell. These index operations occur frequently during the updating of the contour database at each map update step. Since the information about the local area on which all the map updates are conducted is known (i.e. U_k), this information can be used to skip most of the intermediate nodes during search operations of the quadtree.

A skip-tree is generated at each map update step k , using the bounding box information of the set U_k , denoted by $\mathcal{BB}(U_k)$, to achieve this task. The area of the bounding box is divided to a maximum of four rectangles based on quadrant margins from the quadtree, an operation which is similar to area subdivisions in kd-trees [78, 81].

An example of a subdivision of a bounding box is depicted by the orange/bold rectangles in the grid of Fig. 4.9. The orange rectangle is the bounding box of U_k and it is divided to four rectangles A, B, C, D based on the two quadrant boundary lines at $X = x$ and $Y = y$. Each leaf node in the skip-tree, that represent these sub-rectangles links to the smallest quadrant in the quadtree that fully encloses the sub-rectangle in question. Therefore search operations, involving cells within the bounding box, is able to skip large number of intermediate nodes from the root of the quadtree by retrieving information about a closer ancestral quadrant from the skip-tree.

The generation of the skip-tree is conducted during the first indexing operation for each update step. Hence, the complexity of this initial indexing operation be-

Figure 4.9: Linked MX-Quadtree (L-MX-Quadtree) that represents the cell data (right). Leaf nodes of skip-tree contains links to the nodes that represent the smallest quadrants that enclose the respective sub-rectangles (A,B,C,D) represented by the leaf nodes of skip-tree.

comes $\mathcal{O}(\log_2 s_m)$ where s_m is the dimension of the map. ($s_m = \max(\text{width}(\mathcal{M}_k), \text{height}(\mathcal{M}_k))$). For all the subsequent cell indexing operations, the skip-tree can be used to skip intermediate nodes in the MX-Quadtree to reach the node in the quadtree that represent the smallest quadrant enclosing the sub-rectangle of $\mathcal{BB}(U_k)$ containing the grid cell. Considering $d_k^{u,i}$ representing the dimension of the smallest quadrant enclosing sub-rectangle i of $\mathcal{BB}(U_k)$, the complexity of each of the subsequent indexing operation in the L-MX-Quadtree is given by,

$$\underbrace{\mathcal{O}(1)}_{\text{indexing to skip-tree}} + \underbrace{\mathcal{O}(\log_2 \max_i(d_k^{u,i}))}_{\text{indexing inside quadrant retrieved from skip-tree}} = \mathcal{O}(\log_2 \max_i(d_k^{u,i}))$$

Since $d_k^u \geq \max_i(d_k^{u,i})$ where d_k^u is the dimension of the smallest quadrant enclosing $\mathcal{BB}(U_k)$, the above complexity can be bounded to,

$$\mathcal{O}(\log_2 d_k^u)$$

This complexity however can be rewritten using the dimensions of the bounding box of U_k to relate it to the rest of the incremental operations. Consider $d_k^b = \max(\text{width}(U_k), \text{height}(U_k))$ be the dimension of the bounding box $\mathcal{BB}(U_k)$. The

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

inequality $d_k^b < d_k^u$ is trivial from the definition of d_k^u . However, the bounding box must be larger than the sub-quadrants of its enclosing quadrant. This relationship provides an upper bound on the dimensions of the enclosing quadrant.

$$\begin{aligned} \frac{d_k^u}{2} &< d_k^b \\ \therefore d_k^u &< 2d_k^b \end{aligned}$$

Therefore, the complexity of each indexing operation after the initial indexing operation can be written using the dimensions of the bounding box of the current local updated region U_k as $\mathcal{O}(\log_2 2d_k^b)$.

4.2.3 Updating L-MX-Quadtree

At each map update step, some number of boundary cells are deleted from the database and some new set of boundary cells are inserted. Each of these operations require indexing in to the correct node in the L-MX-Quadtree. Previous section detailed the indexing operation and provided the computational complexity of each of the indexing operations after the first indexing operating. Next sections describe how invalid boundary cells are deleted and how new boundary cells are inserted to the database efficiently.

4.2.3.1 Deleting contours

Some mapped-free-space boundary cells extracted in previous update steps disappear during the new map update step. These cells must be identified later to be deleted from the database. According to Definition 14 of mapped-free-space boundary cells, there are two ways in which a boundary cell could disappear. These two cases are listed below.

1. The cell gets classified as an obstacle cell.
2. All of its 4-connected neighboring cells gets mapped as either obstacle or free cells.

Algorithm 14 Delete-cancelled-boundary-contour-cells($\mathcal{B}_k^{del}$)

```

1: for all  $c \in \mathcal{B}_k^{del}$  do
2:   Node  $n \leftarrow \text{FINDQTNode}(c)$ 
3:   Node  $nn \leftarrow \text{next}[n]$ 
4:   Node  $pn \leftarrow \text{prev}[n]$ 
5:    $\text{DELETENODE}(n)$ 
6:    $\mathcal{B}_k^{del} \leftarrow \mathcal{B}_k^{del} - \{\text{cell}[nn]\}$ 
7:   while  $nn \in \mathcal{B}_k^{del}$  do
8:     Node  $n \leftarrow \text{next}[nn]$ 
9:      $\text{DELETENODE}(nn)$ 
10:     $\mathcal{B}_k^{del} \leftarrow \mathcal{B}_k^{del} - \{\text{cell}[nn]\}$ 
11:     $nn \leftarrow n$ 
12:   end while
13:   while  $pn \in \mathcal{B}_k^{del}$  do
14:     Node  $n \leftarrow \text{prev}[pn]$ 
15:      $\text{DELETENODE}(pn)$ 
16:      $\mathcal{B}_k^{del} \leftarrow \mathcal{B}_k^{del} - \{\text{cell}[pn]\}$ 
17:      $pn \leftarrow n$ 
18:   end while
19: end for

```

It can be seen that all cells that belong to first case are guaranteed to be in the set U_k of updated cells. However, since the boundary cell is not updated and only its neighboring cells are updated in the second type, the disappeared boundary cell is not guaranteed to be in the set U_k . Therefore, the set U_k is augmented to include all the neighboring cells of existing cells of U_k . This guarantees that all the disappeared boundary cells are in the set U_k when it is processed. The extracted set of disappeared boundary cells are denoted by $\mathcal{B}_k^{del}$ and is the input to the Alg. 14.

Since, all the cells in $\mathcal{B}_k^{del}$ are parts of contours, the contour information stored in L-MX-Quadtree is utilized to improve the efficiency of the deletion operation as illustrated in Alg. 14. In line 2, the node in the L-MX-Quadtree that represents a disappeared boundary cell is located. The skip-tree based search is used to efficiently retrieve the node. Once the node is found, its next and previous nodes in the contour are retrieved using the links among leaf nodes, lines (3-4). The contour is traversed in both directions from the seed nodes nn and pn to delete nodes until non-disappeared nodes are encountered, lines 7-12 and 13-18. The deletion operation slightly differs from standard quadtree deletion where collapsing of empty nodes is conducted only up to the quadrant enclosing the bounding box. This is to ensure the validity of the links in the skip-tree. During the deletion step of a node, if it is a node representing a corner cell of a contour, the associated item in the corner-list is also deleted.

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

Considering, $C_{\mathcal{B},k}^{del}$ to be the set of partial-contours being deleted from the contour database, the complexity of indexing to each of the deleted partial-contours is given by,

$$\mathcal{O}(|C_{\mathcal{B},k}^{del}| \log_2(2d_k^b))$$

According to the algorithm, for each indexed partial-contour, the node corresponding to the deleted cell is also deleted from the tree. Since the deletion occurs only up to the quadrant that encloses $\mathcal{BB}(U_k)$, the complexity of the deletion operations of each cell in $\mathcal{B}_k^{del}$ is given by,

$$\mathcal{O}(|\mathcal{B}_k^{del}| \log_2(2d_k^b))$$

Therefore the total computational complexity of the deletion of cancelled boundary cells is given as,

$$\mathcal{O}(|C_{\mathcal{B},k}^{del}| \log_2(2d_k^b)) + \mathcal{O}(|\mathcal{B}_k^{del}| \log_2(2d_k^b)) = \mathcal{O}((|C_{\mathcal{B},k}^{del}| + |\mathcal{B}_k^{del}|) \log_2(2d_k^b))$$

4.2.3.2 Inserting contours

The newly appeared boundary cells $\mathcal{B}_k^{new}$ are added to the L-MX-Quadtree using the extracted contour information, $C_{\mathcal{B},k}^{new}$. For each contour, the nodes representing the constituting cells are inserted by first indexing to the smallest available quadrant enclosing the 1x1 quadrant representing the node, using the skip-tree. Then a repetitive division is conducted until the corresponding 1x1 quadrant is created. Alg. 15 details the important steps of the insertion procedure. In lines 2-5, all the cells of new contours are added to the L-MX-Quadtree and the nodes representing adjacent cells in the contours are linked together to preserve contour information. For each new contour, if a corner cell c of contour has an adjacent valid old boundary cell c' , (the check in line 8), then nodes representing c and c' are linked to merge the two contours (lines 9). Since c' was previously a corner and now it is merged to another contour, invalidating its “corner” status, the associated node is deleted from the corner list (line 10). If a corner cell c for a new contour does not have an adjacent old contour cell, c marks an actual corner of a contour. Therefore a corner node associated with c is added to the corner list (line 12).

4.3 Safe and Reachable Frontier Detector (SRFD)

Algorithm 15 Insert-new-boundary-contours($C_{\mathcal{B},k}^{new}$)

```

1: for all contour  $C \in C_{\mathcal{B},k}^{new}$  do
2:   for all  $c \in C$  do
3:     Node  $n \leftarrow \text{INSERTNODE}(c)$ 
4:     Link previous and next node in chain
5:   end for
6:    $c_{start} \leftarrow C[1], c_{end} \leftarrow C[end]$ 
7:   for all  $c \in \{c_{start}, c_{end}\}$  do
8:     if  $\exists c' \in \{\mathcal{B}_{k-1} - \mathcal{B}_k^{del}\}$  s.t.  $c' \in \mathcal{N}_8(c)$  then
9:       Link( $node[c'], c$ )
10:      Remove  $node[c']$  from corner list
11:     else
12:       Insert  $c$  to corner list
13:     end if
14:   end for
15: end for

```

Since, each node representing a new boundary cell is inserted to the tree, the complexity of node insertions become

$$\mathcal{O}(|\mathcal{B}_k^{new}| \log_2(2d_k^b))$$

The complexity of linking nodes corresponding to adjacent cells in contours is linear in the number of new boundary cells, therefore, already included in the earlier term. Since, new corner-list items are added during the insertion process, the complexity of this step becomes $\mathcal{O}(|C_{\mathcal{B},k}^{new}|)$. Since, $|C_{\mathcal{B},k}^{new}| \ll |\mathcal{B}_k^{new}|$, this complexity is absorbed by the complexity of adding nodes for newly extracted boundary cells. Therefore, total complexity of the contour insertion process is given by $\mathcal{O}(|\mathcal{B}_k^{new}| \log_2(2d_k^b))$.

4.3 Safe and Reachable Frontier Detector (SRFD)

Previous sections of this chapter detailed the incremental extraction of mapped-free-space boundary contours from the locally updated cell set U_k . An in-memory data structure that serves as the database for managing these extracted contours across

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

multiple map update steps is also presented. In Chap. 3, a set of procedures to compute the safe and reachable cells in the current occupancy grid map and retrieve this information for each grid cells is presented. In this section, incremental management of mapped-free-space boundary contours and incremental management of safe and reachable cells in the map are combined to develop the incremental strategy for Safe and Reachable Frontier Detection. The following corollary formalizes the incremental frontier extraction approach.

Corollary 3. *Given a mapped-free-space boundary contour C retrieved from the Contour Database and the forest of spanning trees $\mathcal{T}_k$ that stores the reachability of safe grid cells in the occupancy grid at map update step k , the contour C can be reported as a frontier contour if and only if any one grid cell $c \in C$ is reachable by the robot.*

The above result follows directly from the property 2 of the definition of mapped-free-space boundary contours (Def. 21). According to property 2, each boundary cell in a given contour bounds a contiguous set of safe cells. Therefore each cell in the extracted contour also belong to the same contiguous safe cell set. Therefore, it follows that if any one of the grid cells in the extracted contour is reachable, then the rest of the cells in the contour are also reachable due to the connectedness of all boundary cells in the contour. Therefore, a check of reachability for a single cell is enough to check for a contour being a frontier or not.

4.3.1 Retrieving frontier contours

Once the reachability of the contour is established, the set of cells in the contour can be retrieved as detailed in Sec. 4.2.2.1. The run-time complexity of the frontier contour retrieving operation can be computed as follows.

Let us consider $C_{\mathcal{B},k}$ be the set of boundary contours stored in the tree. Each boundary contour is checked for reachability of its cells by the robot through a single check. In Sec. 3.2.2.1 and Sec. 3.2.2.2 it was shown that this check's complexity is bounded by the maximum depth of the spanning trees representing the reachability graph, which is $\mathcal{O}(R_{max}/\delta d)$. Therefore complexity of checking all the contours in the database at step k to extract frontier contours is given by,

$$\overbrace{\mathcal{O}(R_{max}/\delta d) \cdot \mathcal{O}(|C_{\mathcal{B},k}|)}^{\text{contour reachability check}} + \overbrace{\mathcal{O}(|C_{\mathcal{B},k}|)}^{\text{iterate through corner list}} = \mathcal{O}\left(\frac{R_{max}}{\delta d} \cdot |C_{\mathcal{B},k}|\right)$$

Since each cell in contours validated to be reachable, is traversed to report all the valid frontiers, the total complexity of retrieving all the valid frontiers at step k is given by,

$$\mathcal{O}\left(\frac{R_{max}}{\delta d} \cdot |C_{\mathcal{B},k}|\right) + \mathcal{O}(|\Gamma_k|)$$

Since the number of frontier cells in the occupancy grid map is several magnitudes larger than the number of boundary contours in the contour database,

$$|C_{\mathcal{B},k}| \ll |\Gamma_k|$$

Therefore, the complexity of retrieving all frontier cells becomes $\mathcal{O}(|\Gamma_k|)$ which is linear in the number of frontier cells available in the occupancy grid map at step k . The Alg. 16 summarizes the incremental frontier detector’s main operations.

4.4 Experimental results

Experiments are conducted to validate efficiency and the accuracy of the proposed frontier generation strategy, Safe and Reachable Frontier Detection (SRFD), using real world data. The results are compared with the traditional method of frontier generation (TRA), WFD, WFD-INC and OFD methods. The maps of the data sets used for evaluation are depicted in Fig. 4.10. Data for the IRL and ARRL environments are collected by tele-operating a Pioneer-3AT robot equipped with a Hokuyo 2D laser range finder. Data for FR101, CMU-NSH and Seattle-R environments were obtained from the Robotics Data Set Repository (Radish) [82]. The localization of the robot is achieved by executing a popular SLAM implementation GMapping [34] and an implementation of occupancy grid mapping algorithm given in [14] is used with a maximum usable sensor range set to 4m following [41] to control the effect of ray scattering and specular reflection, similar to experiments in Chap. 3. This approach for map generation avoids reinitialization of data structures for the WFD-INC method otherwise needed when “best” map changes in particle based

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

Algorithm 16 Safe-and-Reachable-Frontier-Detector(U_k)

- 1: compute U_k
 - 2: augment U_k by adding the set's neighboring cells
 - 3: compute $\mathcal{O}_k^{new}$ and $\mathcal{O}_k^{del}$ from U_k
 - 4: incrementally compute $\mathcal{I}_k$ using $\mathcal{O}_k^{new}$ and $\mathcal{O}_k^{del}$
 - 5: generate the set of nodes $\hat{\Upsilon}_k$ that replaces the nodes representing split safe-patches
 - 6: generate the set of nodes Υ' that represent safe-patches due to new safe cell data $\Delta\mathcal{S}_k$
 - 7: generate $\dot{\mathcal{T}}_k$ from $\mathcal{T}_{k-1}$ by removing split nodes and inserting $\hat{\Upsilon}_k$
 - 8: generate $\ddot{\mathcal{T}}_k$ by inserting the nodes in Υ'
 - 9: generate $\hat{\mathcal{T}}_k$ from isolated nodes
 - 10: merge $\ddot{\mathcal{T}}_k$ and $\hat{\mathcal{T}}_k$ to generate $\mathcal{T}_k$
 - 11: compute $\mathcal{B}_k^{new}$ and $\mathcal{B}_k^{del}$ from U_k
 - 12: delete contours from Contour Database with $\mathcal{B}_k^{del}$ as input
 - 13: filter erroneous boundary cells to generate $\hat{\mathcal{B}}_k^{new}$
 - 14: incrementally extract boundary contours $C_{\mathcal{B},k}^{new}$ from $\hat{\mathcal{B}}_k^{new}$
 - 15: insert contours to Contour Database with $C_{\mathcal{B},k}^{new}$ as input
 - 16: $\Gamma_k \leftarrow \phi$
 - 17: **for all** contours C from Contour Database **do**
 - 18: **if** any cell $c \in C$ is reachable by robot according to $\mathcal{T}_k$ **then**
 - 19: $\Gamma_k \leftarrow \Gamma_k \cup C$
 - 20: **end if**
 - 21: **end for**
 - 22: **return** Γ_k
-

Figure 4.10: Maps of data-sets used for experimental validation

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

systems. Hence WFD-INC and WFD-IP method are identical in this evaluation and is referred to as $WFD_{INC/IP}$ here onwards.

4.4.1 Comparison of execution time for frontier generation operations

4.4.1.1 Comparison with TRA method

The execution time to generate frontiers using SRFD method is first compared with traditional (TRA) method using IRL and ARRL environments. Grid maps with different sizes are used for IRL and ARRL environments respectively to evaluate the effect of the grid map size on SRFD and TRA methods. For IRL environment grid maps with dimensions 800x800, 1000x1000 and 1200x1200 and for ARRL environment grid maps with dimensions 1200x1200 and 1600x1600 are considered with a fixed grid cell size of 5cm/cell. For each map update step, the time taken to generate and publish the available frontier information to other components is recorded as execution time.

Fig. 4.11 and 4.12 compare the progression of the execution time along the mission between SRFD and TRA methods for IRL and ARRL environments respectively. Comparisons indicate that the execution time to generate and publish frontier information using the TRA approach increases with the map update step. And this increment closely follows the progression of the number of mapped cells in the environment (Fig. 3.17(a) and 3.18(a)). As the number of mapped cells increase in the map, TRA method which processes the entire map has to process more grid cells, justifying the increase in the execution time. In addition, for the same environment with increasing grid map sizes the execution time increases for the traditional approach. This is again due to processing of the entire map. In comparison, for both environment types the execution time for generating frontiers using SRFD method is bounded across map update steps from the start to the end of the mission for all grid map sizes. This is directly due to processing of only local map information by SRFD.

Figure 4.11: Comparison of execution times for frontier generation between SRFD and TRA methods with different map dimensions for IRL environment.

Figure 4.12: Comparison of execution times for frontier generation between SRFD and TRA methods with different map dimensions for ARRL environment.

4.4.1.2 Comparison with other incremental methods

SRFD method's execution time is also compared with WFD, OFD and $WFD_{INC/IP}$ methods. Figs. 4.13-4.15 report the execution times of each method averaged over a mapping mission on corresponding data-sets from start to end. Considering OFD and $WFD_{INC/IP}$ methods are similar in approach to SRFD, reachability detection by processing the full grid map is integrated to them in order to evaluate the effect of this operation on their execution times. The resultant methods are referred to as OFD^* and $WFD_{INC/IP}^*$ respectively and the associated execution times are also reported.

Different combinations of map dimensions (given in meters) and map resolutions (give in meters per cell) result in grid maps with different sizes. Therefore, in order to evaluate the effect of the changes in map dimension and map resolution on the

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

Figure 4.13: Execution time comparison for IRL environment

execution time, different map dimensions (40x40m and 60x60m for IRL and 60x60m, 80x80m for ARRL) and map resolutions (5cm/cell and 2.5cm/cell) are used. Both Figs. 4.13, 4.14 illustrate that increasing the map size while keeping a fixed resolution does not show a significant increase in avg. execution time for all methods. This is because all incremental methods do not process unmapped grid cells thus simply increasing the map size does not increase the execution time. However for the same map size, increasing the resolution (i.e. decreasing the grid cell size from 5cm to 2.5cm) significantly increases the execution time. This is due to the increase in the number of mapped cells with the increase in map resolution. Comparisons on CMU-NSH, FR101 and Seattle-R environments are conducted on maps with dimensions of 40x40m, 80x50m and 80x40m with a map resolution of 5cm/cell. Hence, the sizes of grids used for these three environments are 800x800, 1600x1000 and 1600x800 respectively. Comparisons among the different methods is given next.

Figure 4.14: Execution time comparison for ARRL environment

Since WFD method processes the entire mapped space, it degenerates to TRA method over time and requires higher avg. execution time as expected. Compar-

Figure 4.15: Execution time comparison for CMU-NSH, FR101 and Seattle-R environments

atively OFD method requires significantly low execution time for all environments and map sizes. SRFD and $WFD_{INC/IP}$ requires the lowest execution time of all methods. Avg. execution times for SRFD and $WFD_{INC/IP}$ are comparable for the IRL environment. $WFD_{INC/IP}$ method performs slightly better than SRFD method in CMU-NSH and Seattle-R environments whereas SRFD method slightly outperforms $WFD_{INC/IP}$ in ARRL and FR101 environments. It must be noted that SRFD method performs additional computation on maintaining the reachability information, hence is expected to require more execution time. Comparing the maps of the environments, it can be seen that ARRL and FR101 environments are relatively open whereas CMU-NSH and Seattle-R mainly consists of narrow corridors. This directly translates to how the frontier information is managed incrementally. With relatively long frontier contours in ARRL and FR101 environments, the frontier database employed by $WFD_{INC/IP}$ becomes inefficient with huge number of single cell updates resulting in an increase in execution time. This offsets the additional computation of SRFD for reachability maintenance. In addition the L-MX-Quadtree based frontier database employed by SRFD is able to handle contour-wise update more efficiently thus results in a better performance. However in CMU-NSH and Seattle-R environments, the frontier contours are relatively small hence does not adversely increase the execution time of $WFD_{INC/IP}$ whereas the additional reachability computation of SRFD in these environments offsets the modest improvements in small contour management, thus requires slightly higher execution time compared to $WFD_{INC/IP}$. A more fair comparison on execution times between SRFD, OFD* and $WFD_{INC/IP}^*$ where all methods perform reachability computation reveals that

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

SRFD is significantly efficient in generating reachable valid frontier cells.

4.4.2 Evaluation of accuracy

Efficient generation of the frontier information should not come at the expense of accuracy. While the correctness of the proposed method is analytically provided, it is validated using real world data and the results are provided in this section. The accuracy of SRFD method is evaluated with respect to the traditional method both quantitatively and qualitatively. Accuracy of the WFD, $WFD_{INC/IP}$ and OFD with respect to TRA are also provided.

4.4.2.1 Quantitative analysis of accuracy

For each environment type, both false positive detections and false negative non-detections using SRFD, WFD, $WFD_{INC/IP}$ and OFD methods are recorded with respect to the frontier information generated using the TRA method. False positive detections are cells erroneously reported as frontiers whereas false negative non-detections are cells erroneously not reported as frontier cells with respect to the frontier cells reported by TRA method. This is done across all the map update steps. Comparison of the false positive and false-negative percentages for all environments are given in Figs. 4.16-4.20. It can be seen that SRFD method exhibits the lowest false-positive error percentage and is consistently close to 0%. This is due to SRFD's ability to differentiate between reachable and non-reachable areas allowing the filtering out of "phantom-frontiers". WFD, $WFD_{INC/IP}$ and OFD methods do not specifically check for reachability of the generated frontiers thus result in generation of more false-positive frontiers. Hence they must be filtered out before being used to generate the next exploration decision. Since the reachability is not computed incrementally, the entire mapped space has to be processed at each update step. This requires additional execution time which increases with the increase in the number of mapped cells as described using Figs. 3.17(a), 3.18(a) and reported for OFD* and $WFD_{INC/IP}^*$ in Figs. 4.13, 4.14 and 4.15. Hence it defeats the purpose of efficient incremental frontier generation. However, all methods indicate relatively low

false-negative percentages and SRFD method's false-negative percentage is comparable to $WFD_{INC/IP}$ and WFD for most of the mission time steps and its effect on exploration missions is evaluated qualitatively.

During decision phases of an exploration mission, evaluation of a frontier cell requires information gain calculation which is typically achieved using a ray-casting or cell-counting operation, which is expensive when conducted for each and every frontier cell. Having a high false-positive error percentage in generated frontier cells induce a higher computation time due to the evaluation of invalid frontiers. SRFD's ability to generate frontier cells with low false-positive error percentage, thus reduces this unnecessary computation time and improves the efficiency of the decision steps in exploration missions.

Figure 4.16: (a) Comparison of false positives for IRL environment (b) Comparison of false negatives for IRL environment

Figure 4.17: (a) Comparison of false positives for ARRL environment (b) Comparison of false negatives for ARRL environment

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

Figure 4.18: Comparison of (a) false positives and (b) false negatives for CMU-NSH environment

Figure 4.19: Comparison of (a) false positives and (b) false negatives for FR101 environment

Figure 4.20: Comparison of (a) false positives and (b) false negatives for Seattle-R environment

4.4.2.2 Qualitative analysis of accuracy

The frontiers generated by the SRFD method are qualitatively compared with the ones generated by the TRA method and it is revealed that the SRFD method generates all the major frontier contours identified by the traditional approach. Fig. 4.21 depicts the frontiers generated by both SRFD and TRA methods in IRL environment at map update steps 6 and 15 corresponding to maximum false positive and false negative detection percentages respectively. Frontier cells generated towards the end of the mission at map update step 95 are also compared and it confirms that in all situations, all the major frontier contours are detected. A similar comparison is conducted for the outputs in update steps 19 and 12, from the AARL environment, corresponding to maximum false-positive and false negative detection percentages. Frontiers generated towards the end of the mission at step 300 are also compared to validate that all the major frontier contours are extracted.

Figure 4.21: Qualitative comparison of frontiers generated in the IRL environment by SRFD (a)-(c) and TRA (d)-(f) methods. Detected frontiers are marked in red color

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

Figure 4.22: Qualitative comparison of frontiers generated in the AARL environment by SRFD (a)-(c) and TRA (d)-(f) methods. Detected frontiers are marked in red color

The false negative errors are due to missing individual frontier cells that do not contribute to any significant degradation of the exploration missions as the information about all the major frontier contours are still available. Two frontier based exploration missions are conducted, one with TRA and the other with SRFD as the frontier generation method in an indoor environment to validate that the frontiers generated by SRFD method are sufficient for complete exploration. Utility based on the trade-off between information gain and cost is used for target selection at each step [11]. For both experiments the maximum usable sensor range is set to 4m during mapping and the map resolution is set to 5cm/cell. The mission with SRFD as frontier generation method also utilized its fast frontier refreshing rate to invoke the exploration decision at a higher frequency (i.e. at each frontier refreshing

step) as suggested in [28]. Fig. 4.23 depicts the results of the two exploration missions together with the paths traversed by the robot. Both missions had the same starting pose and generally followed the same exploratory path. The exploration mission with frontiers generated using TRA method and without frequent decision invocation required 224.16m to complete exploration whereas the mission with SRFD method and frequent decision invocation based exploration required only 207.31m. It can be observed that the traditional approach for exploration sometimes lead the robot to the very end of a dead-end/wall (Fig. 4.23(a)) whereas frequent decision making facilitated by SRFD is able to return quickly from such instances thus leading to a more efficient exploration mission, hence confirming the utility of efficient frontier generation for exploration missions.

Figure 4.23: Exploring an indoor environment using frontiers generated from (a) TRA and (b) SRFD method. TRA method without frequent decision invocation required 224.16m. SRFD with frequent decision invocation required only 207.31m

4.5 Summary

This chapter detailed the operations of incrementally extracting mapped-free-space boundary cells as contours and managing these contour information across multiple map update steps during an exploration mission using an in-memory data structure

4. INCREMENTAL SAFE AND REACHABLE FRONTIER DETECTOR

serving as a Contour Database. Mapped-free-space boundary contours are formally defined and the definition is used to formalize the rest of the contour extraction operations. The computational complexity of extracting boundary contours is shown to be asymptotically linear in the number of boundary cells in the local update window. Modification of an MX-Quadtree data structure inspired by skip-lists is proposed to store and maintain these extracted contour information and its computational complexity in updating and retrieving contour information is analyzed and shown to be bounded. Frontier contour extraction from the Contour Database at each map update step is formalized as checking the reachability of a single cell of each stored boundary contour. These results contributed to the unification of efficient reachability management of grid cells and efficient boundary contour management to propose the incremental Safe and Reachable Frontier Detector (SRFD).

Experiments conducted using real world data-sets confirmed that SRFD provides a bound on the execution time of frontier information extraction operations throughout entire exploration missions compared to the traditional approach. Comparisons with state of the art incremental algorithms indicate that SRFD method is comparable to the best available methods and significantly efficient compared to incremental methods that integrates reachability analysis of the detected frontiers. It was also shown that the accurate representation of safe and reachable map information to filter out phantom frontiers results in a very high level of detection accuracy and provides the lowest false positive detections among the existing incremental approaches. An exploration mission conducted in an indoor environment validated that the frontiers generated by SRFD method is sufficient for complete exploration. Additionally it also demonstrated that low computational footprint of SRFD could be used to invoke the exploration decision at high frequency which leads to efficient exploration. The next chapter utilizes the SRFD method's efficiency to extend this idea to propose a new efficient frontier based exploration algorithm.

Chapter 5

Improving Frontier Based Exploration with Task Cancellation

Previous chapters presented efficient incremental set of algorithms to extract frontier information from occupancy grid maps during exploration missions at high frequency. The ability to extract frontier cells at high frequency allows robots to consider the information about frequently evolving frontiers during exploratory motion, to further improve the efficiency of the mission. Traditional frontier based exploration strategy and its many extensions operate by iteratively selecting the next best sensing location myopically and moving to the specified location, until the entire environment is explored. And it does not consider the new information added to the map through continuous observations by the robot along the way to a selected location. This can sometimes lead to redundant traversal by the robot, such as traveling towards a dead-end when the nearby area is already mapped. This chapter proposes to augment the traditional frontier based exploration strategy to include a probabilistic decision step that decides whether further motion on the planned path to the next sensing location is desirable or not using the frequently available frontier information generated by the proposed SRFD method.

The chapter is organized as follows. In Sec. [5.1](#), the motivation and the background behind the proposed task cancellation strategy is detailed. Next in Sec. [5.2](#),

5. IMPROVING FRONTIER BASED EXPLORATION WITH TASK CANCELLATION

the probabilistic decision step used for task cancellation is formulated. Sec. 5.2.1 details the estimation of the neighborhood frontier cell selection probability and Sec. 5.2.2 details the selection of neighborhood of the intermediate target. Chapter concludes with the experimental results and analysis provided in Sec. 5.3.

5.1 Motivation and background

In almost all the frontier based exploration strategies, the robot is left to travel to the selected target (i.e. completion of current sensing task), before the next decision is made[13, 45]. While this method is simple and reduces the exploration problem to a one finding the best intermediate target to expand the map in discrete steps, it does not consider the changes to the frequently evolving frontiers, which could invalidate the usually over-estimated information gain for the intermediate target. This happens when robots select frontier cells, that are very close to obstacles which are yet to be revealed in the map, as their next intermediate targets. As a robot moves towards such a target, continuous sensing reveals the obstacle, which encloses the explored free space, fully or partially, and stops the expansion of most of the frontier cells in the neighborhood, an extreme situation of which is depicted in Fig. 5.1. This disappearance of frontiers in the current target’s neighborhood results in the further movement of the robot towards the target, largely redundant as large portion of the map does not get expanded.

Therefore it is important to check if the map continues to expand beyond each intermediate target. This expansion is directly reflected by the distribution of the frontier cells near the current intermediate target as the robot moves towards this target. This can then be used to avoid the robots making redundant motions. Improving the efficiency of frontier based exploration using frequently updated frontier information has previously been mentioned in the works of Keidar et. al. [73] and [29]. However, their work focuses only on efficiently generating frontiers in high frequency to support such improved strategies. Holz et. al, [27] discusses the drawback of continuing to move towards the intermediate target without considering the continuously updated frontier information. A repetitive rechecking approach is proposed where, the assigned intermediate target is checked for being a valid frontier cell in order to reduce redundant exploratory motion. The work detailed in

Figure 5.1: (a) Illustrates a situation where a frontier cell that is close to a yet to be revealed obstacle boundary, marked in dashed lines, is selected as the next intermediate target. The white area is the already explored free space while solid black lines are the already discovered obstacle boundaries. Gray represents unmapped area. R_{max} is robot's maximum sensing range and light-blue area illustrate robot's current sensing field of view. (b) As the robot moves towards the target (as shown by the arrow), the entire area becomes fully explored (c) The robot continues to move towards the target in traditional exploration strategies, redundantly

this chapter extends this approach to a more general formulation that considers the utility around the target and to decide when to cancel the current motion.

A formulation for the desirability of the frontier cells in the neighborhood of the intermediate target, using information gain values is presented. Then a probabilistic decision steps based on the relative distribution of the desirability of these frontiers is proposed to decide on whether to continue the current motion or to cancel it and execute a new sensing task. This proposed decision step can be used to augment any frontier based exploration strategy to reduce redundant robot motions and is experimentally validated by augmenting the baseline frontier based exploration strategy.

5.2 Formulation of exploration task cancellation step

Let us define the current intermediate target of the robot as $x_T \in \mathcal{R}^2$ and the final heading of the robot's planned motion at x_T as θ_T . The set of neighboring frontier cells of target x_T of step k is denoted by $\Gamma_k(x_T) \subset \mathcal{R}^2$. Details of computing the neighborhood area of the current intermediate target is given in Sec. 5.2.2. The

5. IMPROVING FRONTIER BASED EXPLORATION WITH TASK CANCELLATION

Figure 5.2: x_T is the current intermediate target, θ_T is the desired heading angle. Both λ_i, λ_j are example depictions of two frontier cells neighboring x_T . Note that frontier cell λ_k is outside the neighborhood of x_T , so not considered for the cancellation decision. The neighborhood is depicted by the dashed semi-circle with a radius of R_{max} . The light-blue shaded area represents robot's current sensor field of view.

random variable $\Theta : \mathcal{R}^2 \rightarrow [0, \frac{\pi}{2}]$ is defined as follows and illustrated in Fig. 5.2.

Definition 23. Given a frontier cell $\lambda \in \Gamma_k(x_T)$ and θ_T the desired orientation of the robot at the intermediate target $x_T \in \mathcal{R}^2$, then the value $\Theta(\lambda)$ provides the absolute angle the vector $\overrightarrow{x_T \lambda}$ makes with respect to the robot's desired heading θ_T .

Considering that the map and the associated frontier cells are updated at discrete steps during the planned motion to x_T , at each map update step k , a probability density function $f_{\Theta}^k : [0, \frac{\pi}{2}] \rightarrow [0, 1]$ can be defined. This distribution describes the probability of selecting a frontier cell, from $\Gamma_k(x_T)$, in a specified absolute angle difference with θ_T , as the next intermediate target. Suppose γ to be the angle tolerance used to consider frontier cells as belonging to robot's final heading direction. Then, the robot's motion towards its current target x_T is cancelled if $P(\Theta \leq \gamma) < p_T$. The cumulative probability on the left hand side of the inequality using f_{Θ}^k provides a measure of desirability of the frontier cells in the 'direction' (i.e. $\theta_T \pm \gamma$) of the robot's final heading for the robot. If this desirability is less than some probability threshold p_T , the robot's motion towards the current target is cancelled.

Fig. 5.3 illustrates how a frontier based exploration strategy can be augmented

5.2 Formulation of exploration task cancellation step

Figure 5.3: (a) Major steps in the base-line exploration strategy (b) Major steps in the proposed augmented exploration strategy.

to include this task cancellation decision. Once an intermediate target for exploration is selected, the augmented strategy enters a loop where frequently updated frontier information, at a given frequency, is used to calculate $P(\Theta \leq \gamma)$. The loop terminates when the current motion towards the intermediate target is cancelled or when the robot has reached the intermediate target.

The probability threshold p_T can be changed according to the level of cancellation desired. A more aggressive form of exploration can be activated by setting p_T to a higher value and a more conservative form of exploration can be activated by a lower p_T value. Setting p_T to zero makes the inequality false for all scenarios and will not cancel the robot motion, hence will make the robot behave identically to the baseline strategy. $P(\Theta \leq \gamma)$ is calculated as follows.

$$\begin{aligned}
 P(\Theta \leq \gamma) &= \int_0^\gamma f_\Theta^k(\theta) d\theta \\
 &= \int_0^\gamma \sum_{\lambda \in \mathcal{X}_\theta^T} f_X^k(\lambda) d\theta
 \end{aligned} \tag{5.1}$$

Here, the set $\mathcal{X}_\theta^T$ is defined as $\{\lambda \in \Gamma_k(x_T) \text{ s.t. } \Theta(\lambda) = \theta\}$ and contains all the neighboring frontier cells of x_T with an absolute angle difference to θ_T equal to θ . $f_X^k : \mathcal{R}^2 \rightarrow [0, 1]$ is the probability mass function (p.m.f.) that provides a measure of desirability of selecting neighboring frontier cell $\lambda \in \Gamma_k(x_T)$ as an alternative target.

5. IMPROVING FRONTIER BASED EXPLORATION WITH TASK CANCELLATION

5.2.1 Estimating neighborhood frontier cell selection probability

The probability of selecting a neighboring frontier cell (i.e. desirability of frontier cell as an alternative intermediate target) is calculated based on their individual utilities. Hence the p.m.f. can be defined as,

$$f_X^k(\lambda) = \frac{J(\lambda)}{\sum_{\lambda_i \in \Gamma_k(x_T)} J(\lambda_i)} \quad (5.2)$$

However, it is reasonable to assume that the effect of the travel cost is negligible for the set of neighboring frontiers of x_T , $\Gamma(x_T)$, as the variation among these travel costs is minimal. Hence Eq. 5.2 can be approximated using the information gain values $I(\lambda)$ as follows.

$$f_X^k(\lambda) \approx \frac{I(\lambda)}{\sum_{\lambda_i \in \Gamma_k(x_T)} I(\lambda_i)} \quad (5.3)$$

5.2.2 Selecting the neighborhood of intermediate target

Since the task cancellation decision is made based on the desirability measure of frontier cells in the neighborhood of the current intermediate target, the selection of this neighborhood influences the task cancellation decision. Setting an arbitrarily large neighborhood on one hand could result in a large set $\Gamma_k(x_T)$ which requires significant number of frontier cells to be present in the direction of the robot's desired motion (based on γ) for the task not be cancelled. This could result in frequent task cancellation which is undesirable. On the other hand, it could also include frontier cells that are far away from the area currently being explored, as belonging to robot's desired direction even though these frontier cells do not reflect the expansion of the map at the area of the map in concern as illustrated in Fig. 5.4(a). Setting a small neighborhood, on the other hand, would result in $\Gamma_k(x_T)$ being empty and would aggressively cancel any exploration task. This results in invoking large number of decision steps, which ultimately is inefficient for an exploration mission.

Considering that the maximum possible expansion of the map for a single sensing task in a certain direction from the intermediate target is R_{max} and that the map gets expanded only in the direction of the robot's motion towards the intermediate

Figure 5.4: Selection of $N(x_T)$. (a) A large neighborhood contains frontiers that do not expand as the robot moves towards the intermediate target (i.e. frontiers marked in orange). A small neighborhood does not contain any frontiers even though frontiers (marked in red) expand in the direction of robot's motion. (b) Selection of $N(x_T)$ and extracting $\Gamma(x_T)$ based on R_{max} and θ_T

target, a semi-circle with a heading of θ_T with a radius of R_{max} provides the best representation for the neighborhood, $N(x_T)$, of the intermediate target, Fig. 5.4(b). Since only reachable frontier cells within the defined neighborhood $N(x_T)$ reflect the options available for the robot to explore in other directions, only the reachable cells in $N(x_T)$ are selected as neighboring frontier set $\Gamma_k(x_T)$.

5.3 Experimental results

Experiments are conducted in both real and simulated environments. A Pioneer 3 AT (P3AT) robot equipped with a Hokuyo LRF for sensing is primarily used to illustrate the effect of the different parameter values on a single sensing task in real environments. Extensive simulation experiments are conducted in Stage simulator with Player drivers using two complex environments, indoor and outdoor, to quantitatively evaluate the effect the different parameter configurations have on complete exploration missions. For all experiments, the occupancy grid map update/access frequency is set to 1Hz. The robot's maximum sensor range, R_{max} , is set to 4m for all the reported experiments. For complete exploration missions, α and β parameter values are set to 1.0 and 4.0 respectively. It must be mentioned that α, β param-

5. IMPROVING FRONTIER BASED EXPLORATION WITH TASK CANCELLATION

ters affect only the intermediate target selection procedure and does not have any effect on the task cancellation decision. Therefore, these values are fixed for all the simulation experiments. A real experiment is also conducted using the P3AT robot to validate the efficiency of the proposed method in a simple indoor environment.

5.3.1 Analyzing the effect of parameters on different environments

In the first type of experiments, the cancellation of exploration tasks is compared with traditional exploration tasks to better understand the effects of the two parameters γ and p_T . The two environments used for comparing parameter effects is shown in Fig. 5.5 and Fig. 5.6. The first environment considers the situation of a robot approaching a narrow passage with a dead end, which are often found in indoor environments. The end of the passage is 4.5m from robot's starting point. Hence, there exists unmapped area towards the dead end and robot selects a target which is 3.5m away and close to the dead-end. The second environment type considers a robot travelling towards an obstacle in a relatively open environment (Fig. 5.6(a)). The two targets used, A and B, are placed about 0.5m and 1.5m away from the room boundary respectively as shown in Fig. 5.6 and separate experiments are conducted to analyze the effect of the distance the intermediate target has to the obstacle boundary, on the task cancellation step and its efficiency.

Fig. 5.7 depicts the travel distance of the robot towards the dead-end with varying γ and p_T values. It can be seen that irrespective of the γ value, when the threshold probability is set to zero, it behaves identically to the classical exploration strategy and continues to move towards the target which is close to the dead, end even though the entire passage gets fully explored by a small forward motion. In all other situations, the robot's motion gets cancelled within 0.5-1.0 m forward motion and behaves identically. This is because, as the robot moves forward about 0.5m-1.0m, due to the narrowness of the passage, the entire environment is fully explored and there are no more frontiers to be selected as intermediate targets. Hence the two parameters do not affect the cancellation decision differently.

During the experiments on the open room type environment, the robot is placed such that both the targets A and B lie on the frontier with the robot's sensor

Figure 5.5: (a) The view of the dead end. (b) Map of the narrow passage with dead-end. The blue box is the starting position of the robot, red circle is the target near the dead end. The red band circling the mapped space is the inflated obstacle grid cells.

Figure 5.6: (a) The panoramic view of the open room space. The robot is sent towards the wall with windows shown at the center of the image in a straight line. (b) Map of the open room environment. Two targets A and B are used.

facing towards the target. In each experiment, the robot is sent to the specified intermediate target in an exploratory motion following a straight trajectory. The travel distance at the time the robot stops is recorded for each run. For each target A and B, the area the robot could explore is measured manually. This value is used to generate the remaining information gain percentage associated with the robot's travelled distance. It can be noted that, the remaining information gain exhibits a

5. IMPROVING FRONTIER BASED EXPLORATION WITH TASK CANCELLATION

Figure 5.7: Travel distance comparisons for single task cancellation in narrow passage with dead-end

diminishing returns property with respect to the travelled distance. This is because as the robot moves towards the targets, most of the area in front of the robot gets mapped. Remaining unmapped area resides to the two sides of the robot's motion. Hence, the straight motion of the robot does not map the unknown area with high efficiency. This results in a lower net-gain with increasing travelled distance. This observation justifies cancelling the motion before the robot reaching the assigned target and reassigning a new target when the robot's motion towards the previous target is less 'desirable'.

The two graphs in Fig. 5.8(a) and Fig. 5.8(b) summarizes the results from these experiments and illustrates the effect of γ and p_T in deciding the desirability of the robot's motion. For both targets, setting p_T to zero, irrespective of the γ value, makes the robot behaves exactly similar to the traditional exploration approach and moves the robot to the target which is 3.5m away. And also as $\gamma = 90^\circ$ includes the entire neighborhood as the robot's 'heading', the robot moves forward until there are no frontiers in the neighborhood, hence the long travel distances. For target A, it can be seen that, travel distances between 1.0m-1.5m explores between 70%-85% of the neighboring area approximately. Cancelling the sensing task during this travel distance interval can be considered more efficient for the exploration mission as the robot could cover the remaining unknown areas more efficiently by employing a different motion from the current one. For targets that lie very close to obstacle boundaries in open environments (relative to R_{max}) such as A, selecting

5.3 Experimental results

$\gamma = 75^\circ$ and $p_T = [0.1 - 0.5]$ or $\gamma = 60^\circ$ and $p_T = [0.1 - 0.3]$ is observed to generate motions that are neither too aggressive nor too conservative in exploring according to Fig. 5.8(a). For target B, the travel distance interval 2.0m-2.75m explores between 77%-90% of the neighboring area approximately. Hence, similar to target A, it can be considered that selecting $\gamma = 60^\circ$ and $p_T = [0.3 - 0.5]$ or $\gamma = 45^\circ$ and $p_T = [0.1 - 0.3]$ generate exploratory motions that exhibit the correct balance of aggressiveness and conservativeness for targets that are not too close or far away from obstacle boundaries relative to the sensor range. It can also be observed that for $\gamma = 75^\circ$, the generated motions are either too conservative, for $p_T = [0.1 - 0.7]$, or too aggressive, for $p_T = [0.8 - 1.0]$ hence not suitable for exploration missions.

Figure 5.8: Travel distance comparisons for single task cancellation in (a) Room environment - Target A (b) Room environment - Target B

5. IMPROVING FRONTIER BASED EXPLORATION WITH TASK CANCELLATION

5.3.2 The effect of task cancellation on exploration missions

The previous section analyzed the effect on exploration of the environment with cancellation of a single sensing task. An exploration mission is a sequence of such sensing tasks. Simulation experiments were conducted to quantitatively analyze the effect of the proposed strategy on complete exploration missions. Simulations are conducted in two environments, indoor and outdoor, which are depicted in Fig. 5.9. The indoor environment contains a typical office layout with narrow corridors and rooms. The outdoor environment contains more open space (relative to the sensor range) with irregular obstacle shapes. The distance the robot traversed to complete the exploration mission is used as the metric of exploration efficiency.

Figure 5.9: Environments used for simulations (a) Indoor (b) Outdoor

For all simulation experiments, $(\gamma = 30^\circ, p_T = [0.1 - 0.3])$, $(\gamma = 45^\circ, p_T = [0.1 - 0.5])$ and $(\gamma = 60^\circ, p_T = [0.1 - 0.7])$ are selected as the parameter space. These parameters are selected roughly based on the single sensing task performance analysis in the previous section. For each (γ, p_T) parameter pair used for experiments, five simulation runs were executed and the average travel distance and the average total number of decision steps to complete exploration missions is recorded together with their standard deviations. The comparison of these two metrics for all the parameter pairs is given below.

Fig. 5.11 and Fig. 5.12 compare the average travel distance efficiency of the proposed method with the traditional approach (TRA) with different parameter value pairs for indoor and outdoor environments respectively. For the indoor environment, it can be observed that the proposed method is able to complete exploration missions with smaller traversed distance for all parameter pairs compared to TRA method. (Travel distance gains of 18.95%-26.14% on average). Since TRA method does not consider the frequently updated frontier information, it is prone to redundantly moving the robot very close to obstacle boundaries resulting in a higher travelled distance. The relatively uniform exploration efficiency of the proposed method is due to the narrowness of the indoor environment. In Sec. 5.3.1, it was illustrated that all parameter pairs have the same effect on a single task cancellation when approaching a narrow dead-end. Since exploring an indoor environment is simply a sequence of such sensing tasks, the effect of the different parameters is identical to what is reported in Fig. 5.7. Fig. 5.10 qualitatively compares the exploratory path of a robot in the indoor environment using TRA and the proposed method. It is clearly visible that the usage of TRA method results in unnecessary motion deep in to rooms whereas the proposed method avoids them resulting in the improved travel distance.

Figure 5.10: Comparison of traversed path during exploration (a) Traditional method (TRA) (b) Proposed method

The travel distance gains for the outdoor environment shows more variation compared to the indoor scenario. This is due to the more open nature of the environment.

5. IMPROVING FRONTIER BASED EXPLORATION WITH TASK CANCELLATION

Figure 5.11: Comparison of avg. travel distance to explore the indoor environment

Figure 5.12: Comparison of avg. travel distance to explore the outdoor environment

Since the environment is relatively open, even during motion towards an obstacle the robot is capable of mapping unknown areas which are visible from the robot's sensor to the sides of the robot, albeit diminishing returns in mapping efficiency. Therefore the travel distance gains are not as pronounced as in the indoor environment and lies in the range 3.75%-14.68%. However, it can be observed that when the task cancellation is made aggressive by increasing the p_T value, the travel distance to complete the exploration decreases as expected. It can also be noted that setting $\gamma = 30^\circ$ in it self results in aggressive task cancellation and uses less travel distance to complete the exploration mission compared to TRA method. Fig. 5.13 compares the exploratory paths taken by P3AT robot while exploring an indoor environment using both the TRA and the proposed method. This qualitative comparison in a real environment also validates that the proposed method avoids sending the robot

very close to obstacle boundaries which result in an efficient exploratory path.

Figure 5.13: Qualitative comparison of (a) TRA : Travel distance 39.91m (b) Proposed approach : Travel distance 32.99m ($\gamma = 45^\circ$, $P_T = 0.3$)

5.3.3 Number of decision steps to complete missions

Increasing the aggressiveness of task cancellation increases the total number of times the decision step need to be executed to complete the exploration mission. Figs. 5.14 and 5.15 summarizes the average number of decision steps required to complete exploration missions under different (γ, p_T) parameter pairs for both indoor and outdoor environments respectively. It can be observed that with increasing p_T values, the number of decision steps increase for all γ values. This result highlights the importance of carefully balancing exploration efficiency with the number of decision steps as very high frequency of task cancellation could result in a stop and move motion of the robot. Based on the results for both environments, the parameter pairs $(30^\circ, 0.1)$, $(45^\circ, 0.3)$ and $(60^\circ, 0.4)$ can be selected to provide the best balance between the decrease in the travel distance and increase in the number of decision steps.

5. IMPROVING FRONTIER BASED EXPLORATION WITH TASK CANCELLATION

Figure 5.14: Avg. number of decision steps required for full exploration in indoor environment

Figure 5.15: Avg. number of decision steps required for full exploration in outdoor environment

5.4 Summary

This chapter utilized the fast frontier refreshing rates introduced in previous chapters to augment the traditional frontier based exploration strategy to include a decision step that cancels sensing tasks if they are no longer desirable. The check for desirability was formulated as a probabilistic decision step and the traditional approach is shown to be a special case of this augmented strategy. It was revealed that the parameters of the task cancellation step has less bearing when exploration is conducted in narrow environments. However, they have a significant effect on the traversed distance gain when exploring relatively open spaces such as outdoor environments. Overall, the experiments concluded that the proposed augmented

exploration strategy that leverages fast frontier availability with task cancellation is capable of conducting exploration missions more efficiently than the traditional approach. The next chapter examines how the frontier based exploration method augmented with task cancellation step described here can be used as an alternative to using dynamic path planning algorithm to navigate in an unknown environment to reach a given goal point.

Chapter 6

Directed Exploration for Goal Oriented Navigation in Unknown Environments

In this chapter, the task cancellation strategy proposed in the previous chapter is integrated to a directed exploration strategy to develop an alternative to sensor based dynamic planning algorithms to navigate a robot in an unknown environment filled with obstacles to a specified goal location. The improvement to the exploratory motion using task cancellation quickly detects dead-ends and avoids the robot from moving deep in to these dead-ends, which is similar to the behavior of sensor based dynamic planning algorithms. The exploration of the robot is directed towards the goal by iteratively selecting intermediate target points in the environment that reduces the distance estimate to the goal from robot's current position, until the goal is reached. The proposed method alleviates the need to have the complete grid space, including the goal location, at the start of the navigation mission as required by many dynamic grid path planning algorithms such as D*. The proposed method inherently supports any mapping application that outputs a grid of mapped information unlike many dynamic planning methods that require additional changes to support particle filter based mapping systems due to map orientation changes.

The chapter is organized as follows. Next section describes the motivation and the background in using directed exploration as an alternative to sensor based dy-

6. DIRECTED EXPLORATION FOR GOAL ORIENTED NAVIGATION IN UNKNOWN ENVIRONMENTS

dynamic planning for navigation to a goal in an unknown environment. Then in Sec. 6.2, the proposed Directed Exploration for Goal Oriented Navigation (DEGON) approach is described. The utility function that directs the robot's exploration towards the goal is presented. Then the estimation of the lower bound on the shortest distance from a frontier cell to goal is formulated by generating a visibility graph through the unknown (i.e. unmapped) area. The shortest distance estimations for frontiers is elaborated in Sec. 6.3. The effect of task cancellation using the frontier cell desirability in intermediate target's neighborhood, presented in previous chapter, to detect and avoid dead-end situations is summarized with illustrations in Sec. 6.4. Section 6.5 describes the conducted experiments and analyzes the results.

6.1 Motivation and background

Autonomously navigating to a specified goal position in an unknown environment is one of the most important tasks expected of many autonomous robot missions such as search and rescue, autonomous rendezvous, autonomous guided coverage etc. The goal position could be a location from which a distress call is originated or any other important position in an environment that requires visitation by a robot. Since the environment is unknown, static planning algorithms such as A* and Dijkstra's shortest path algorithm [39] cannot be used to generate plans for the robot to move towards the goal. Therefore dynamic sensor based planning strategies have been developed to solve the problem of safely navigating a robot in a previously unknown environment filled with obstacles to a goal location. These algorithms differ from static algorithms by incorporating updated map information at each execution step to dynamically alter the existing planned paths to reflect the changes in the map. Real-Time Heuristic Search[83], Lifelong Planning A* (LPA*)[84], D* [85], D* Lite [86] are some of many implementations of this general idea. D*, D*-Lite and many variants of these two algorithms have been extensively used on top of occupancy grid map[12] based environment representations for dynamic path planning by roboticists due to their efficient real-time re-planning capabilities.

However, all of these dynamic planning algorithms require the grid cell representing the target point to be included in the initial planning grid space. Since the

environment is initially unknown, the dimensions of the grid space cannot be set accurately. Therefore, most of the practical robotic mapping applications provide the ability to dynamically change the size of the grid map as the robot uncovers more area[34]. They initially start with a smaller map size corresponding to the currently mapped area and increases the map size in fixed increments in the direction of the map's growth. Therefore, the grid cell corresponding to the goal position is not guaranteed to be in the output map from the start. Hence, all the above mentioned grid based dynamic planning algorithms require significant changes to account for this. In addition, currently most of these maps are generated using particle filter based SLAM solutions[87] due to their high accuracy in estimating the robot's pose and the associated map in challenging environments. In these solutions, each particle contains a separate map of the environment. The map associated with the current best particle is used for all the path planning steps. When the best particle changes, the map exposed to the planning algorithm also gets changed. The change of the map generally results in some changes in the orientations of environmental features already mapped in previous steps. This leads to changes in values of grid cells in the map on a global scale. These global changes invalidate the assumption of local map updates used by dynamic planning algorithms to achieve their efficiency thus degenerating them to planning algorithms that operate from scratch. Fig. 6.1 illustrates the orientation changes of map features due to best particle changes. (Note the orientation changes of obstacle boundaries with respect to the two vertical and horizontal guide lines).

The work presented in this chapter takes a frontier based exploration approach to move the robot towards the goal in the unknown environment and is similar to the approach taken by [50]. Using an exploration based approach alleviates the need to have the complete grid space to be available at the start of the mission. Also, since exploration algorithms do not use grid cells from previous maps to store intermediate distance information, grid cell updates on a global scale induced by map orientation changes do not negatively affect the decision process. In goal directed exploration, intermediate targets are selected in such a way to direct the robot's exploration of the environment towards the goal. Once the goal location is within the mapped area (i.e. robot sees the target), the robot abandons the exploration strategy and moves the robot directly towards the target. The proposed approach differs from [50] by

6. DIRECTED EXPLORATION FOR GOAL ORIENTED NAVIGATION IN UNKNOWN ENVIRONMENTS

Figure 6.1: The red colored horizontal and vertical lines are the same for both figures and are used to provide a visual reference. The blue box is the footprint of the robot (a) Map before best particle change. (b) Map after best particle change at the next map update step. It can be seen that the lower half of the mapped area has been slightly rotated in a clockwise direction.

introducing a method to estimate the lower bound of the distance from a frontier cell to the goal. While [50] focuses on different strategies to recover from trapped situations, the proposed method utilizes the exploration task cancellation, based on fast frontier refreshing rates proposed in the previous chapters, to avoid moving the robot deep into such trapped/dead-end situations.

6.2 Goal Directed Exploration : Problem formulation

Directed exploration for goal oriented navigation is similar in operation to the classical frontier based exploration. At each decision step, the robot selects an intermediate target point to move such that the distance to the goal is reduced. This operation is iterated until the goal point is visible for the robot (i.e. within the mapped area) at which point the exploration procedure is abandoned and a path

6.2 Goal Directed Exploration : Problem formulation

is planned to directly navigate to the visible goal. Hence, the important task of the directed exploration strategy is to find the sequence of best intermediate target points that successively reduce the distance to the goal. A sequence of intermediate target navigation stages during a directed exploration to a goal are illustrated in Fig. 6.2.

Figure 6.2: Illustration of several intermediate steps during a directed exploration towards goal g . At each step (a),(b),(c) and (d) an intermediate target f_c (green circle) that reduces the distance estimate to the goal from robot is selected from the frontier cells (marked in red) in the map.

At a given decision step, the selection of the next intermediate target f_c from the set of frontier cells Γ can be done as follows.

$$f_c = \arg \max_{f \in \Gamma} J(f) \quad (6.1)$$

$$= \arg \max_{f \in \Gamma} \{-(\text{dist}(r, f) + \hat{\text{dist}}(f, g))\} \quad (6.2)$$

The utility $J(f)$ quantifies the desirability of selecting the frontier cell f as the next intermediate target to move the robot closer to the goal. The utility is composed of two distance values. $\text{dist}(r, f)$ is the shortest distance from the robot's position r to frontier cell f . $\hat{\text{dist}}(f, g)$ is the shortest distance between the frontier cell and the goal through the still unknown area. The combined distance of $\text{dist}(r, f) + \hat{\text{dist}}(f, g)$

6. DIRECTED EXPLORATION FOR GOAL ORIENTED NAVIGATION IN UNKNOWN ENVIRONMENTS

Figure 6.3: Estimating the shortest distance from frontier points f_1, f_2 (green dots) to goal. For f_1 , $\hat{\text{dist}}(f_1, g)$ is equal to the euclidean distance between f and g . For f_2 , $\hat{\text{dist}}(f_2, g)$ and $\text{eucl}(f_2, g)$ are different from each other. The distances from robot to frontier cells, $\text{dist}(r, f_1)$ and $\text{dist}(r, f_2)$, are computed through the mapped free space.

indicates the shortest distance the robot is estimated to travel assuming the robot first moves to the frontier cell f and then moves through the unknown region to reach the goal. Replacing $\hat{\text{dist}}(f, g)$ with $\text{eucl}(f, g)$, for ease of distance computation, invalidates this assumption as the straight line connecting f and g could intersect already mapped area indicating the robot going back to the mapped area after reaching f . Since the goal is still in the unmapped region, the assertion of robot going back to the mapped area after reaching frontier cell f contributes to a sub-optimal travel plan. This is because, as the robot is now planned to cross a different frontier, it is expected to reach an additional frontier cell in addition to f to move to the goal location. Therefore, $\hat{\text{dist}}(f, g)$ is used instead of the euclidean distance estimation. Fig. 6.3 illustrates the relationships between the above described distance metrics.

Distances from the robot to frontier cells, $\text{dist}(r, f)$ for each frontier cell f , can be computed based on the existing map information using Dijkstra's shortest path algorithm. If the entire grid of the environment is available, then dijkstra's algorithm can also be executed with goal cell g as the source, through the unknown area, to compute the $\hat{\text{dist}}(f, g)$ distances. However, as mentioned previously, for the major part of the navigation mission, the current grid map output by the mapping module may not include the goal cell due the goal being further away from currently mapped area. This prohibits the exhaustive grid cell search for shortest distance from cell g to all frontier cells. Therefore, when the goal cell is outside the mapped region, the

6.3 Estimating shortest distances to goal through the unknown area

calculation of $\hat{\text{dist}}(f, g)$ for each frontier cell f using only the available map data is proposed and is explained next.

6.3 Estimating shortest distances to goal through the unknown area

6.3.1 Generating goal visibility graph through unknown area

The computation of the shortest distances to the goal from frontier cells are dependent on the presence of the goal cell in the current grid map. When the goal cell is outside the current grid map, the estimates to the shortest distances to goal from frontier cells are computed by generating a visibility graph from frontier cells to the goal through the unknown area. Throughout this chapter, the meaning of the word visibility is abused to refer to the visibility of two grid cells to each other through an unknown/unmapped grid area without any mapped grid cells intersecting the straight line connecting the two cells. In order to generate the visibility graph, first the convex hull of the mapped grid cells set M is calculated [88]. Consider $\mathcal{CH}(M)$ to be the generated convex hull and V to be the set of vertices of the boundary of the convex hull.

$$V = \text{Vertices}(\mathcal{CH}(M)) \quad (6.3)$$

Each vertex $v \in V$ has two adjacent vertices. Connecting these two vertices generates two edges. $|V|$ number of edges can be generated to connect all the vertices in V to generate a connected graph G . Fig. 6.4 depicts generated convex hulls for some example mapped regions. Given vertex v_g (representing the goal cell g) is located outside $\mathcal{CH}(M)$, v_g can be connected to the visible vertices in graph G to generate the visibility graph (through the unknown area) of the set $V^* = V \cup \{v_g\}$. The visibility graph $G_{vis}(V^*)$ is defined as follows.

$$G_{vis}(V^*) = \{V^*, E_{vis}\}$$

$$E_{vis} = \{(u, v) | u, v \in V^*, u \text{ is visible to } v\}$$

6. DIRECTED EXPLORATION FOR GOAL ORIENTED NAVIGATION IN UNKNOWN ENVIRONMENTS

Figure 6.4: Generation of convex hulls for the mapped areas in three successive steps of a navigation mission. The blue lines indicate the boundary of the generated convex hulls.

Fig. 6.5 illustrates the visibility graphs generated through the unknown space using the convex hulls depicted in Fig. 6.4. For each vertex in the visibility graph, the shortest distance from v_g can be calculated by executing Dijkstra's shortest path algorithm on $G_{vis}(V^*)$ from v_g .

6.3.2 Estimating the shortest distance from each frontier cell to goal

For each frontier cell f that is visible from the goal, the shortest distance $\hat{\text{dist}}(f, g)$ through the unknown cells is equal to the euclidean distance between the two cells, $\text{eucl}(f, g)$. It can also be deduced that such frontier cells lie on the boundary of $\mathcal{CH}(M)$. For all other frontier cells f that are inside $\mathcal{CH}(M)$ or lie on the boundary of $\mathcal{CH}(M)$ with being non-visible to goal, the shortest distance through the unknown cells $\hat{\text{dist}}(f, g)$ is calculated using the visibility graph $G_{vis}(V^*)$ described earlier.

When frontier cell f is on the boundary of $\mathcal{CH}(M)$ and is non-visible to the goal, a single vertex v_f representing frontier cell f can be temporarily added to $G_{vis}(V^*)$ to generate $G_{vis}(V^* \cup \{v_f\})$. Considering that this vertex is added on to the convex hull boundary, the shortest distance to v_f from v_g can be easily calculated as

6.3 Estimating shortest distances to goal through the unknown area

Figure 6.5: Generation of the visibility graphs from the computed convex hull depicted in Fig. 6.4. The blue colored circles denote the vertices in V^* of the visibility graph.

$$\hat{\text{dist}}(f, g) = \min\{(\hat{\text{dist}}(c_1, g) + \text{eucl}(c_1, f)), (\hat{\text{dist}}(c_2, g) + \text{eucl}(c_2, f))\} \quad (6.4)$$

where c_1 and c_2 are the grid cells corresponding to incident vertices of the edge on which v_f is added. (i.e. cells corresponding to the neighboring vertices of v_f in $G_{vis}(V^* \cup \{v_f\})$.)

When frontier cell f is inside the convex hull, in order to calculate the shortest distance through $G_{vis}(V^*)$, the visibility of the convex-hull boundary from the frontier cell f is computed. Fig. 6.6(a) illustrates the computation of the visibility region of the convex-hull boundary from a frontier cell f . The intersection points of the computed visibility region's two boundary lines (i.e. yellow lines) with the convex-hull's boundary, i_1, i_2 , are temporarily inserted as two vertices to the graph $G_{vis}(V^*)$ together with the vertex associated with frontier cell f , v_f . The generated temporary visibility graph $G_{vis}(V^* \cup \{v_f, v_{i1}, v_{i2}\})$ is then used to compute the shortest distance estimate.

Since v_{i1} and v_{i2} are on the boundary of the convex hull, the shortest distance estimates from v_g to v_{i1} and v_{i2} can be computed as in eq. 6.4. Again c_1 and c_2 are the two corner cells of the boundary edge on which i_1 and i_2 reside.

6. DIRECTED EXPLORATION FOR GOAL ORIENTED NAVIGATION IN UNKNOWN ENVIRONMENTS

Figure 6.6: (a) Computes the visibility region of the convex-hull boundary from a frontier cell f . The yellow edges indicate the two margins of the visibility region. The intersection of the two margins with the convex-hull boundary (blue line) introduces two new vertices (yellow circles) to the visibility graph. (b) These vertices are used to compute the shortest distance estimate to the goal from frontier f . The shortest path is illustrated in orange color.

$$\text{dist}(\hat{i}_1, g) = \min\{(\hat{\text{dist}}(c_1, g) + \text{eucl}(c_1, i_1)), (\hat{\text{dist}}(c_2, g) + \text{eucl}(c_2, i_1))\} \quad (6.5)$$

$$\text{dist}(\hat{i}_2, g) = \min\{(\hat{\text{dist}}(c_1, g) + \text{eucl}(c_1, i_2)), (\hat{\text{dist}}(c_2, g) + \text{eucl}(c_2, i_2))\} \quad (6.6)$$

Then, these two distance estimates can be used to compute the final estimate $\hat{\text{dist}}(f, g)$ as follows and an example is illustrated in Fig. 6.6(b).

$$\hat{\text{dist}}(f, g) = \min\{(\hat{\text{dist}}(i_1, g) + \text{eucl}(i_1, f)), (\hat{\text{dist}}(i_2, g) + \text{eucl}(i_2, f))\} \quad (6.7)$$

While $\text{eucl}(i_1, f)$ and $\text{eucl}(i_2, f)$ are not the true shortest distance values from f to the convex hull boundary through unknown cells, they help to provide a tight upper-bound on the shortest distance to the goal from f .

When the visibility region of the convex-hull's boundary from a frontier cell f is empty due to f being surrounded by mapped cells, a cell c on the convex-hull's boundary with the shortest distance to the frontier cell f is selected to connect the vertex of cell f , v_f , to the visibility graph as illustrated in Fig. 6.7. Since the distance between f and c is estimated using $\text{eucl}(f, c)$ where the straight line may go through mapped cells, the final shortest distance approximation $\hat{\text{dist}}(f, g)$ is not

6.3 Estimating shortest distances to goal through the unknown area

Algorithm 17 Compute $\hat{\text{dist}}(f, g)$ for each frontier cell f when g is outside $\mathcal{CH}(M)$

ComputeDistHat($G_{\text{vis}}(V^*), \Gamma$)

```

1: for all  $f \in \Gamma$  do
2:   if  $g$  is visible from  $f$  through unknown area then
3:      $\hat{\text{dist}}(f, g) \leftarrow \text{eucl}(f, g)$ 
4:   else
5:      $v_f \leftarrow$  the vertex associated with  $f$ 
6:     if  $f$  is on the convex-hull boundary then
7:       Generate temporary graph  $G' \leftarrow G_{\text{vis}}(V^* \cup \{v_f\})$  and update distances
8:     else
9:        $\mathcal{VR}(f) \leftarrow$  Visibility region of boundary of  $\mathcal{CH}(M)$  from  $f$ 
10:      if  $\mathcal{VR}$  is not empty then
11:        Generate temporary graph  $G' \leftarrow G_{\text{vis}}(V^* \cup \{v_f, v_{i1}, v_{i2}\})$  and update distances
12:      else
13:        Generate temporary graph  $G' \leftarrow G_{\text{vis}}(V^* \cup \{v_f, v_c\})$  and update distances
14:      end if
15:    end if
16:     $\hat{\text{dist}}(f, g) \leftarrow$  get shortest distance from  $v_f$  to  $v_g$  from  $G'$ 
17:  end if
18: end for

```

a tight approximation as in previous instance. However, the occurrence of such frontiers is limited during a directed exploration mission and does not affect the final result in a negative way. Alg. 17 summarizes the distance estimation through unknown cells for frontier cells given the visibility graph.

Figure 6.7: Computing shortest distance for frontier cells that are not visible to the convex-hull's boundary

6. DIRECTED EXPLORATION FOR GOAL ORIENTED NAVIGATION IN UNKNOWN ENVIRONMENTS

Figure 6.8: Illustration of the effect of task cancellation on early detection and recovery from dead-end situations. (a) Intermediate target point starts to send the robot deep in to a yet to be revealed dead-end. (b) Repetitive rechecking detects the dead-end, cancels the current target and assigns a new intermediate target. (c) Robot recovers from the dead-end and continues the motion towards the goal

6.4 Detecting and avoiding dead-ends

During goal oriented navigation, the robot could be heading towards a dead-end in the environment which is not yet revealed by the map. When it is revealed in the map, the dynamic planning algorithms are able to quickly re-plan and avoid the robot from going further in to the dead-end. This behavior is dependant on the update rate of both the map and the planner. Traditional exploration missions do not check the updated map until the robot has reached its assigned intermediate target position. Therefore, using this strategy for directed exploration towards goals could lead robots further into dead-end situations. In the previous chapter, an exploration task cancellation strategy was introduced by leveraging the fast frontier refreshing rates due to the proposed efficient incremental frontier extraction algorithms. It was shown that augmenting traditional exploration strategies with task cancellation steps avoids the robots going deep in to dead-end situations in environments. The parameter pair ($\gamma = 45^\circ, p_T = 0.3$) is selected based on the recommendations from the experimental results in previous chapter. Fig. 6.8 illustrates the use of this approach in quickly detecting and recovering from such dead-end situations during directed-exploration missions.

Alg. 18 summarizes the DEGON approach. The execution of the algorithm enters the while loop at line. 2 as long as the grid cell representing the goal g is not within the mapped cells of the occupancy grid map. If cell g is within the

6.4 Detecting and avoiding dead-ends

mapped cells, execution terminates the while loop and then a path can be directly planned to the goal and completes the navigation mission according to line. 17. While g is not visible to the robot, at each iteration of the while loop, the distance to all the mapped cells from the robot is computed using Dijkstra's shortest path algorithm in line. 4. This in turn gives the shortest distances to the frontiers Γ extracted in the next line. The convex hull of the mapped cells is generated in line. 6 and if g is outside the convex hull, then a visibility graph is generated to compute the distances from frontiers to the goal through unknown cells in the map using the function `ComputeDistHat`. If the goal cell g is within the convex hull, this indicates that cell g is within the occupancy grid map $\mathcal{M}$ and Dijkstra's shortest path algorithm can be directly used to compute the shortest distances from each frontier to the goal cell. These computed distances are then used in line. 13 to select the next intermediate target in the goal directed exploration approach. Then the robot is sent towards the selected intermediate target with the task cancellation strategy on. The execution completes this iteration of the while loop when the robot reaches the selected intermediate target or when the motion is cancelled.

Algorithm 18 Directed Exploration for Goal Oriented Navigation (DEGON)

Require: $\mathcal{M}$: the occupancy grid map

Require: g : the goal cell

```

1:  $M \leftarrow$  Mapped cells of  $\mathcal{M}$ 
2: while  $g$  is not within updated  $M$  do
3:    $r \leftarrow$  Robot's position cell
4:    $\text{dist}(r, *) \leftarrow \text{Dijkstra}(M, r)$ 
5:    $\Gamma \leftarrow$  Generate frontiers
6:   Compute  $\mathcal{CH}(M)$ 
7:   if  $g$  is outside  $\mathcal{CH}(M)$  then
8:     Compute  $G_{vis}(V^*)$ 
9:      $\hat{\text{dist}}(*, g) \leftarrow \text{ComputeDistHat}(G_{vis}(V^*), \Gamma)$ 
10:  else
11:     $\text{dist}(*, g) \leftarrow \text{Dijkstra}(M, g)$ 
12:  end if
13:   $f_c \leftarrow \arg \max_{f \in \Gamma} \{-(\text{dist}(r, f) + \hat{\text{dist}}(f, g))\}$ 
14:  Move robot to  $f_c$  with task cancellation until  $f_c$  is reached or motion is cancelled
15:   $M \leftarrow$  Mapped cells of  $\mathcal{M}$ 
16: end while
17: Plan a path to  $g$  using  $M$  and move robot

```

6. DIRECTED EXPLORATION FOR GOAL ORIENTED NAVIGATION IN UNKNOWN ENVIRONMENTS

6.5 Experimental evaluation

The proposed Directed Exploration for Goal Oriented Navigation (DEGON) approach is evaluated for its completeness and motion efficiency in multiple simulated environments. All experiments are conducted in Stage simulator with a simulated Pioneer2DX robot using Player drivers. The simulated robot is equipped with a laser range finder with 180° field of view and a maximum sensor range of 4m and has a maximum speed of 0.5m/s. Fig. 6.9 illustrates the environments used for experiments. In order to compare the completeness and efficiency of the DEGON method with dynamic algorithms, the same set of experiments are also conducted using a re-planning algorithm. A static path planning algorithm is executed at each 0.5m difference in robot's position to simulate the invocation of re-planning in dynamic planning algorithms and is referred to as RP. The same frequency based on 0.5m robot position change is used for invoking the task cancellation steps using the parameters ($\gamma = 45^\circ, p_T = 0.3$) for repetitive rechecking of dead-ends in the DEGON method. The generated paths are both qualitatively and quantitatively compared.

6.5.1 Completeness and efficiency of the generated paths

Figs. 6.11-6.13(b) show the qualitative comparison between the final traversed paths of the robot to reach its designated goal point in each environment for both methods. It can be seen that DEGON method is able to reach the goal point if the goal is reachable and also the traversed paths are similar to the paths generated by a dynamic planning algorithm.

In the outdoor environment, the DEGON approach is also able to move the robot through the narrow opening in the middle of the environment similar to the dynamic re-planning approach and illustrates that the method will always select the shortest reachable path towards the goal. Comparing the results in the maze environment, (6.11(a) and 6.11(b)), it can be seen that the DEGON method is equally successful in avoiding going deep in to dead-ends and is able to recover quickly and reach the goal point. Results from the office floor environment again illustrate that DEGON method does not deviate to explore nearby office rooms and

Figure 6.9: Environments used during simulated experiments

continues towards the goal point which is similar in behavior to the dynamic re-planning approach. The building entrance environment simulates a mixed scenario where the robot is expected to reach the entrance of the building in the middle of the environment and the road surrounding the building is littered with unstructured obstacles. It can be observed that in both methods, the robot continues to move in the shortest path before reaching a dead-end. At this point, both methods force the robot to turn back and reach the goal point using the alternate direction. The above qualitative comparisons illustrate that the proposed method is equally capable in finding the best path to reach a given goal in an initially unknown environment.

6. DIRECTED EXPLORATION FOR GOAL ORIENTED NAVIGATION IN UNKNOWN ENVIRONMENTS

Figure 6.10: Comparison between (a) DEGON and (b) RP results in outdoor environment

6.5.2 Quantitative comparison

Tab. 6.1 summarizes the comparison of the travel distances and the number of decision steps required to reach the final goal point for all the four environments. It can be seen that for both methods, the travel distances are relatively equal. However, there is a noticeable difference in the number of planning decision steps required to reach the goal point. The dynamic re-planning strategy (RP) invokes the re-planning procedure at each 0.5m change in robot's position and it can be seen that the number of decisions steps for RP is roughly equal to twice the length of the traversed paths. In comparison, DEGON method requires relatively lower number of decision steps. This is due to the proposed method making decisions only when it has reached current intermediate target or when it has detected a dead-end and the exploration task is cancelled. These results indicate that the proposed DEGON method is a valid alternative to dynamic path planning algorithms for navigating robots to goal points in unknown environments. Since DEGON method is able to seamlessly work with mapping systems that output varying sized maps with varying orientations (due to best particle changes in particle based mapping systems), it can be considered to be more adaptable to practical robotic systems.

Figure 6.11: Comparison between (a) DEGON and (b) RP results in maze environment

Environment	Travel Distance(m)		No. of Decision Steps	
	DEGON	RP	DEGON	RP
Outdoor	37.44	37.55	20	76
Maze	52.65	54.19	22	110
Office	25.16	25.30	13	46
Building Entrance	146.25	139.41	81	270

Table 6.1: Summary of travel distance and decision steps comparison

6.5.3 Real world validation

The proposed directed exploration strategy was validated in real world scenarios during TechX Challenge, robotics competition ¹. One phase of the competition scenario involved moving the robot to a specified goal through previously unknown environment with obstacles and is similar to the maze environment used in simulations. The navigation method was implemented using ROS software platform and was deployed on multiple robot platforms. Fig. 6.14 illustrates several sequential stages of a robot executing the directed exploration strategy to avoid obstacles and move towards the goal.

¹A robotics competition organized by Defence Science and Technology Agency (DSTA) of Singapore where a team of robots are tasked with autonomously navigating a largely unknown environment and search and engage a specified set of targets located in both indoor and outdoor settings.

6. DIRECTED EXPLORATION FOR GOAL ORIENTED NAVIGATION IN UNKNOWN ENVIRONMENTS

Figure 6.12: Comparison between (a) DEGON and (b) RP results in office environment

6.6 Summary

This chapter presented a Directed Exploration for Goal Oriented Navigation (DEGON) approach as an alternative to popular dynamic planning algorithms to navigate a robot to a goal in an unknown environment, as another application of the improvements made to frontier based exploration approaches through efficient incremental frontier generation. The proposed method also introduces a new approach to estimate the shortest distance between a frontier cell and the goal using a visibility graph generated through the unknown space. These distance estimates together with the distances to the frontier cells from robot are combined to generate a utility function that directs the robot's exploration towards the goal. Simulation experiments conducted at different environments indicate that the goal directed exploration combined with the dead-end detection based on fast frontier refresh rates performs comparably to dynamic planning algorithms on the distances travelled to reach the goal. The results also indicate that the DEGON method requires less than third of the decision steps required by dynamic planning algorithms to reach the goal. Given DEGON method's inherent ability to work with particle based mapping systems with outputs of varying map sizes and its comparable performances, it can be considered as a viable alternative for goal oriented navigation in unknown environments.

Figure 6.13: Comparison between (a) DEGON and (b) RP results in building-entrance environment

Figure 6.14: Illustration of stages of a directed exploration based navigation using a tracked robot (Jaguar) in a real environment.

Chapter 7

Compact Exploration of Unbounded Environments

Trade-off between the information gain and travel cost has been extensively used as an evaluation criteria for selecting intermediate target points in robot exploration strategies where the primary goal is to reduce the mission completion time. Completion of exploration missions are ensured by the presence of closed boundary information of the environment to be explored. However, when boundaries of the environment to be explored are unknown or unspecified, exploration missions must focus on mapping in a balanced way in all directions surrounding a given location, to reveal as much usable information as possible. This chapter introduces a new utility function that can be used in exploration strategies to map the terrain in a balanced way in all directions to generate compact maps to reveal more usable information about the terrain surrounding a given location. The computation of the utility function is based on minimizing the variance of waiting times of frontier cells before they are being visited by the robots. The efficient frontier extraction and management strategy presented in earlier chapters is utilized to efficiently manage the waiting times of the frontier cells in the occupancy grid map during missions, hence the proposed compact exploration strategy is another example of using efficient frontier generation and management on improving exploration missions.

The chapter is organized as follows. The next section describes in detail the motivation and the background of compact exploration for autonomous mapping

7. COMPACT EXPLORATION OF UNBOUNDED ENVIRONMENTS

of unbounded environments. Then in Sec. 7.2, the utility function that replaces the traditional utility function is formulated and the details of the simplifications of the newly formulated utility function for efficient computation are given. Sec. 7.3 describes the use of the proposed utility function in both single and multi-robot exploration strategies to generate compact maps. Finally Sec. 7.4 details the experiments conducted to validate the proposed compact exploration strategies and concludes the chapter by analyzing the results from the experiments.

7.1 Motivation and background

Majority of existing exploration algorithms for both single and multi-robot missions select robot's next intermediate target point by employing a utility function that computes the utility value of each candidate target point by trading off the expected information to be gained at the point with the cost of reaching the target point [5, 11, 14, 20, 45]. This myopic nature of intermediate target point selection makes robots act greedily and to aggressively move away from the starting location [89]. When the boundary of the environment to be explored is pre-determined, the environment can be fully explored irrespective of the order in which regions are explored and numerous research has been conducted to improve the time efficiency of such missions [25, 26, 27, 58, 90]. However, when the environment to be explored is unbounded, employing a greedy exploration strategy could leave certain parts of the environment unexplored. This generally results in the generated map not revealing the topology of the environment accurately [89]. Fig. 7.1 illustrates an example situation where a robot employing the traditional utility function for exploration greedily moving away from the initial location thus not providing useful terrain information surrounding the initial location. Therefore exploring an unbounded environment introduces a new "efficiency" metric where the robot should seek to map all directions approximately equally.

Exploring unknown spaces where the focus is to generate information about environmental features such as obstacles has been examined in [30, 91, 92]. These approaches replace the traditional utility function with a function that favors navigation actions that sends robots to reveal more environmental features. While this approach adaptively decides the next best sensing location for feature mapping, the

Figure 7.1: Illustration of exploring an unbounded environment using a normal exploration strategy. As the mission evolves, the robots moves further away from the starting region (in the center of the map) in an elongated path

compact exploration in unbounded environments proposed in [89] specifically considers balanced mapping in equal directions from an initial location and in doing so reveals useful information about environmental features surrounding the initial location. Two strategies are proposed in this approach to generate compact maps. In both strategies, a new cost function is added to the existing evaluation function that is used to compare frontier points. These cost functions restrict robots from aggressively venturing into unknown territory thus providing a balanced mapping in all directions.

The utility function proposed in this chapter is also aimed at taking the latter approach to generate useful maps of unstructured environments by conducting compact exploration from an initial location. It differs from the approaches in [89] by utilizing the efficiently available frontier cell information from Chap. 4 to efficiently compute a new utility value, that completely replaces the traditional utility function.

7.2 Minimizing Waiting Time Variance based utility function

7.2.1 Formulation

A new attribute $w_i(t)$, *waiting time*, is introduced for each frontier cell i for time t . This is the time elapsed, in seconds, from which grid cell i is identified as a frontier to the current time step t . Old frontier cells have higher waiting times while newer frontier cells have lower waiting times. At each time step of the mission, waiting times of these frontier cells are incremented. The operation of waiting time incrementation can be simplified by storing for each frontier cell, the time step value at which the cell became a frontier cell. Then the waiting time of grid cell i can be computed as

$$w_i(t) = t - I[i]$$

where $I[i]$ stores the initial time step at which cell i was classified as a frontier cell. Therefore, computation of the waiting time for each frontier cell requires only the time step value during frontier cell initialization. Therefore, it requires differentiating between the newly extracted frontier cells in the current map update step from the existing frontier cells. The brute force approach using the traditional frontier extraction strategy would be to extract all the frontier cells and compare them with the frontier cells extracted in the previous update step to identify the new frontier cells at each map update step. This approach as mentioned in previous chapters is inefficient due to repetitive extraction of many unchanged frontier cells. The incremental efficient frontier extraction and management strategy proposed in Chap 4 can be directly used to extract only the newly generated frontiers and update their initialization value in I in an efficient manner.

Puig et. al [58] introduced a similar waiting time metric to quantify the balanced mapping of terrain and concluded that lower waiting time variance (WTV) indicates a more balanced distribution of waiting times across regions in the map, thus indicates that all regions have roughly similar arrival times by the robot. However, [58] used WTV only to evaluate the balancedness of the exploration missions and

7.2 Minimizing Waiting Time Variance based utility function

not to generate motion strategies for balanced exploration. Ensuring similar arrival times to frontier cells by the robot indicates that the robot(s) are restricted from continuing to explore in only one direction and must return to visit frontier cells in other directions, resulting in a balanced mapping of the terrain. This heuristic understanding of the balancing act of the robots' navigation is used in the selection of the WTV based utility function.

The next best intermediate target for the robot to visit is selected to be the frontier cell that maximizes the reduction of the waiting time variance in the robot's neighborhood. Therefore, the normalized reduction of the waiting time variance is used to compute the utility for the frontier cells as follows.

$$U_{WT}^m = \frac{\Delta\sigma_{WT}^2(t, t + \tau(m)) - \min_p \Delta\sigma_{WT}^2(t, t + \tau(p))}{\max_p \Delta\sigma_{WT}^2(t, t + \tau(p)) - \min_p \Delta\sigma_{WT}^2(t, t + \tau(p))} \quad (7.1)$$

$\Delta\sigma_{WT}^2(t, t + \tau(m))$ is the estimated reduction in waiting time variance if the robot moves from current location at time t to frontier cell m . Value $\tau(m)$ is the expected time required for the robot to reach frontier cell m from its current location. Throughout this chapter, it is assumed that the robot travels at a constant velocity, thus $\tau(m)$ is calculated by dividing the length of the path $l(m)$, generated by the path planner, by the constant velocity.

$\Delta\sigma_{WT}^2(t, t + \tau(m))$ is expanded as follows.

$$\Delta\sigma_{WT}^2(t, t + \tau(m)) = \sigma_{WT}^2(t) - \sigma_{WT}^2(t + \tau(m)) \quad (7.2)$$

$\sigma_{WT}^2(t)$ is the variance of the waiting times of the frontier cells in the neighborhood of the robot at time t and $\sigma_{WT}^2(t + \tau(m))$ is the estimated variance of the waiting time of the frontiers if the robot moves to cell m after spending $\tau(m)$ time. Γ_t is the set of neighboring frontier cells and $\Gamma_{t+\tau(m)}$ is the estimated set of neighboring frontier cells if the robot moves to cell m , consuming $\tau(m)$ time. $w_\mu(t)$ and $w_{t+\tau(m)}$ are the mean waiting times of the frontier cells in Γ_t and $\Gamma_{t+\tau(m)}$ respectively.

7. COMPACT EXPLORATION OF UNBOUNDED ENVIRONMENTS

$$\sigma_{WT}^2(t) = \frac{\sum_{i=1}^{|\Gamma_t|} (w_i(t) - w_\mu(t))^2}{|\Gamma_t|} \quad (7.3)$$

$$\sigma_{WT}^2(t + \tau(m)) = \frac{\sum_{i=1}^{|\Gamma_{t+\tau(m)}|} (w_i(t + \tau(m)) - w_\mu(t + \tau(m)))^2}{|\Gamma_{t+\tau(m)}|} \quad (7.4)$$

$\min_p \Delta \sigma_{WT}^2(t, t + \tau(p))$ is the minimum such reduction of the waiting time variance and $\max_p \Delta \sigma_{WT}^2(t, t + \tau(p))$ is the maximum possible reduction in waiting time variance and can be expanded as follows.

$$\min_p \Delta \sigma_{WT}^2(t, t + \tau(p)) = \sigma_{WT}^2(t) - \max_p \sigma_{WT}^2(t + \tau(p)) \quad (7.5)$$

$$\max_p \Delta \sigma_{WT}^2(t, t + \tau(p)) = \sigma_{WT}^2(t) - \min_p \sigma_{WT}^2(t + \tau(p)) \quad (7.6)$$

Using above expansions, the utility function in eq. 7.1 can be simplified as follows.

$$\begin{aligned} U_{WT}^m &= \frac{(\sigma_{WT}^2(t) - \sigma_{WT}^2(t + \tau(m))) - (\sigma_{WT}^2(t) - \max_p \sigma_{WT}^2(t + \tau(p)))}{(\sigma_{WT}^2(t) - \min_p \sigma_{WT}^2(t + \tau(p))) - (\sigma_{WT}^2(t) - \max_p \sigma_{WT}^2(t + \tau(p)))} \\ &= \frac{\max_p \sigma_{WT}^2(t + \tau(p)) - \sigma_{WT}^2(t + \tau(m))}{\max_p \sigma_{WT}^2(t + \tau(p)) - \min_p \sigma_{WT}^2(t + \tau(p))} \end{aligned} \quad (7.7)$$

It can be observed that the direct implementation of eq. 7.7 to compute the utility value for a single frontier cell, it requires iterating through all frontier cells in the neighborhood. Therefore the run time complexity of the intermediate target point selection strategy where all the neighboring frontier cells are evaluated, becomes $\mathcal{O}(k^2)$ where k is the total number of frontier cells in the robot's neighborhood. Next section illustrates the simplifications to this equation to further improve the computational complexity of computing the utilities for frontier cells in the robot's

neighborhood.

7.2.2 Efficient computation of the cell utility

It can be noticed that when a robot moves from its current position to a new frontier cell m , it generates a new set of frontier cells and cancels some of the existing frontier cells, while leaving some of the frontiers in the neighborhood intact. Let $\Gamma_{t+\tau(m)}$ be the estimated set of frontier cells after the robot's possible move to cell m . This set of frontier cells can be considered as a rearrangement of old and new frontier cells as follows.

$$\Gamma_{t+\tau(m)} = \{\Gamma_t \setminus \Gamma_{D,t+\tau(m)}\} \cup \Gamma_{O,t+\tau(m)} \quad (7.8)$$

$\Gamma_{D,t+\tau(m)}$ is defined as the set of cells that would lose the frontier status due to mapping by the robot and $\Gamma_{O,t+\tau(m)}$ is the set of new frontier cells that would be generated by the new observations up to cell m . This is illustrated graphically in Fig. 7.2

Figure 7.2: Dotted line connecting robot to point m is the planned path, and its length $l(m)$ is divided by the constant velocity v to get $\tau(m)$. Lines encircling the robot are frontiers. Thin line is the existing set of frontier cells Γ_t . Dotted line is the set of frontier cells, $\Gamma_{D,t+\tau(m)}$, that would be canceled if robot moves to cell m . Bold line is the estimated new set of frontier cell, $\Gamma_{O,t+\tau(m)}$, created if robot moves to m .

When the robot reaches m , a new set of frontier cells are generated. The waiting

7. COMPACT EXPLORATION OF UNBOUNDED ENVIRONMENTS

times of the newly generated frontier cells in set $\Gamma_{O,t+\tau(m)}$ is set to zero. However, waiting times of frontier cells that were not canceled, $\Gamma_t \setminus \Gamma_{D,t+\tau(m)}$, are incremented by the constant $\tau(m)$.

$\Gamma_{O,t+\tau(m)}$ is not an existing frontier cell set. This set is generated/estimated according to the available partial map of the environment at the current time step. Therefore the actual number of newly generated frontier cells cannot be correctly estimated. Hence, it is assumed that the number of newly generated frontier cells when robot reaches m is equal to the number of frontier cells that would be deleted (i.e. $|\Gamma_t| = |\Gamma_{t+\tau(m)}|$), which is approximately true for most instances. This assumption together with the knowledge of constant increment of waiting time of unchanged frontier cells provides an update equation that reduces the computational complexity of evaluating $\arg\max_m U_{WT}^m$ for intermediate target selection step.

The assumption of $|\Gamma_t| = |\Gamma_{t+\tau(m)}|$, makes the denominator of all σ_{WT}^2 values in eq. 7.7 to be equal, which further simplifies it to the calculation of sum of squared error values as shown below.

$$U_{WT}^m = \frac{\max_q S_{1,n+d}(t + \tau(q)) - S_{1,n+d}(t + \tau(m))}{\max_q S_{1,n+d}(t + \tau(q)) - \min_q S_{1,n+d}(t + \tau(q))} \quad (7.9)$$

$S_{1,n+d}(t + \tau(m))$ denote the sum of squared error term of $\sigma_{WT}^2(t + \tau(m))$ (i.e. numerator) given in eq. 7.4. Hence efficient calculation of the normalized waiting time variance gain term reduces to efficient calculation of this term. The notation used in this calculation is as follows.

The frontier cells in Γ_t are indexed from 1 to $n + d$, ($k = n + d$), where the last d indices represent frontier cells identified in $\Gamma_{D,t+\tau(m)}$ (i.e. $d = |\Gamma_{D,t+\tau(m)}|$). This means the frontier cells indexed from 1 to n represent frontier cells that would not change in between a robot's movement from current position to cell m . Similarly the frontier cells in $\Gamma_{t+\tau(m)}$ are indexed from 1 to $n + d$ where the first n are the unchanged frontier cells while last d belong to $\Gamma_{O,t+\tau(m)}$ based on the assumption that $|\Gamma_{D,t+\tau(m)}| = |\Gamma_{O,t+\tau(m)}|$.

In describing the new update equation, notation $T_{i,j}(t)$ is used to denote the sum of waiting times of the frontier cells indexed from i to j in the frontier cell set corresponding to time variable t . $T_{1,n+d}(t)$ denote the sum of waiting times of

7.2 Minimizing Waiting Time Variance based utility function

current frontier cells at time t . This value is pre-calculated for a single decision step and is stored as a constant and used in the utility computation for each frontier cell.

$$T_{1,n+d}(t) = \sum_{i=1}^{n+d} w_i(t) \quad (7.10)$$

$T_{1,n+d}(t + \tau(m))$ denote the projected sum of waiting times of frontier cells if robot would travel to cell m . Since waiting times of frontier cells indexed from $n+1$ to $n+d$ are zero,

$$\begin{aligned} T_{1,n+d}(t + \tau(m)) &= T_{1,n}(t + \tau(m)) \\ &= T_{1,n}(t) + n\tau(m) \end{aligned} \quad (7.11)$$

With the above definitions, $S_{1,n+d}(t + \tau(m))$ can be expanded as,

$$\begin{aligned} S_{1,n+d}(t + \tau(m)) &= \sum_{i=1}^{n+d} \left(w_i(t + \tau(m)) - \frac{T_{1,n+d}(t + \tau(m))}{(n+d)} \right)^2 \\ &= \sum_{i=1}^n \left(w_i(t + \tau(m)) - \frac{T_{1,n+d}(t + \tau(m))}{(n+d)} \right)^2 \\ &\quad + d \left(\frac{T_{1,n+d}(t + \tau(m))}{(n+d)} \right)^2 \end{aligned} \quad (7.12)$$

$T_{1,n+d}(t + \tau(m))$ can be rewritten as $T_{1,n+d}(t) - T_{n+1,n+d}(t) + n\tau(m)$ and can be calculated in $\mathcal{O}(d)$ time for each m given $T_{1,n+d}(t)$ is pre-calculated at the start of the decision step.

The first term of eq. 7.12 denoted as V_1 can be expanded as follows.

7. COMPACT EXPLORATION OF UNBOUNDED ENVIRONMENTS

$$\begin{aligned}
V_1 &= \sum_{i=1}^n \left(w_i(t) - \frac{T_{1,n+d}(t) - T_{n+1,n+d}(t) + n\tau(m)}{(n+d)} + \tau(m) \right)^2 \\
&= \sum_{i=1}^n \left(\left(w_i(t) - \frac{T_{1,n+d}(t)}{(n+d)} \right) + \frac{T_{n+1,n+d}(t)}{(n+d)} \right. \\
&\quad \left. + \left(1 - \frac{n}{n+d} \right) \tau(m) \right)^2 \\
&= \sum_{i=1}^n \left(w_i(t) - \frac{T_{1,n+d}(t)}{(n+d)} \right)^2 + n \left(\frac{T_{n+1,n+d}(t)}{(n+d)} \right)^2 \\
&\quad + n \left(1 - \frac{n}{n+d} \right)^2 \tau(m)^2 + V_2 + V_3 \sum_{i=1}^n \left(w_i(t) - \frac{T_{1,n+d}(t)}{(n+d)} \right) \tag{7.13}
\end{aligned}$$

$$V_2 = 2n \left(\frac{T_{n+1,n+d}(t)}{(n+d)} \right) \left(1 - \frac{n}{n+d} \right) \tau(m) \tag{7.14}$$

$$V_3 = 2 \left(\frac{T_{n+1,n+d}(t)}{(n+d)} + \left(1 - \frac{n}{n+d} \right) \tau(m) \right) \tag{7.15}$$

V_2 , V_3 and second and third terms of eq. 7.13 can be calculated in $\mathcal{O}(d)$. The first summation term of eq. 7.13 can be simplified as follows,

$$\begin{aligned}
\sum_{i=1}^n \left(w_i(t) - \frac{T_{1,n+d}(t)}{(n+d)} \right)^2 &= \sum_{i=1}^{n+d} \left(w_i(t) - \frac{T_{1,n+d}(t)}{(n+d)} \right)^2 - \sum_{i=n+1}^{n+d} \left(w_i(t) - \frac{T_{1,n+d}(t)}{(n+d)} \right)^2 \\
&= S_{1,n+d}(t) - \sum_{i=n+1}^{n+d} \left(w_i(t) - \frac{T_{1,n+d}(t)}{(n+d)} \right)^2 \tag{7.16}
\end{aligned}$$

It can be observed that this term can also be calculated with $\mathcal{O}(d)$ time complexity for each m given $S_{1,n+d}(t)$ at the start of decision step, which indeed is available.

The last term of eq. 7.13 denoted as V_4 is further simplified as,

$$\begin{aligned}
V_4 &= V_3 \left(\sum_{i=1}^n \left(w_i(t) - \frac{T_{1,n}(t)}{n} \right) + n \left(\frac{T_{1,n}(t)}{n} - \frac{T_{1,n+d}(t)}{(n+d)} \right) \right) \\
&= nV_3 \left(\frac{T_{1,n}(t)}{n} - \frac{T_{1,n+d}(t)}{(n+d)} \right) \\
&= V_3 \left(\frac{dT_{1,n+d}(t) - (n+d)T_{n+1,n+d}(t)}{(n+d)} \right) \tag{7.17}
\end{aligned}$$

It is evident that V_4 also can be calculated in $\mathcal{O}(d)$ time for each m since $T_{1,n+d}(t)$ is calculated only once at the start of the decision step.

Above simplifications indicate that $S_{1,n+d}(t + \tau(m))$ term can be calculated in $\mathcal{O}(d)$ time for each grid cell m . Hence the computation of utility value for a single cell m can be done with $\mathcal{O}(d)$ complexity. Since the decision step iterates over $k = n + d$ different frontier cells to generate utilities for all neighboring grid cell, the computational complexity of computing the utility function for all cells m is $\mathcal{O}((n+d)d)$ which is lower than $\mathcal{O}((n+d)^2)$ since $d \ll n$, hence improves the overall efficiency of the decision process of the exploration mission. Next section describes the use of this formulated utility function to derive compact exploration algorithms.

7.3 Compact exploration algorithms

7.3.1 Single robot compact exploration

Compact exploration strategy for a single robot can be derived directly from the traditional single robot frontier based exploration strategy [13] by simply replacing the traditional utility function with the function introduced in the previous section. Alg. 19 illustrates the major steps of the compact exploration strategy.

In traditional exploration missions, the boundary of the environment to be explored is explicitly or implicitly specified. This results in the termination condition being the complete mapping of the area within the specified boundary. However, in compact exploration, the boundary information is not available. Therefore, an artificial termination condition must be enforced such as the time elapsed from the start of the mission or the battery power remaining in the robot. These termination

7. COMPACT EXPLORATION OF UNBOUNDED ENVIRONMENTS

Algorithm 19 Single Robot Compact Exploration

```

1: while termination condition is not met do
2:   Incrementally extract new frontier cells and record time-step value in  $I[c]$  for each new cell  $c$ 
3:   Extract the set of neighboring frontier cells  $\Gamma_t$  from frontier database
4:   Update waiting times  $w_c(t)$  for each cell  $c \in \Gamma_t$ 
5:   Compute  $T_{1,n+d}(t)$  for  $\Gamma_t$ 
6:   Compute  $S_{1,n+d}(t)$  for  $\Gamma_t$ 
7:   for each  $m \in \Gamma_t$  do
8:      $V_2 = 2n \left( \frac{T_{n+1,n+d}(t)}{(n+d)} \right) \left( 1 - \frac{n}{n+d} \right) \tau(m)$ 
9:      $V_3 = 2 \left( \frac{T_{n+1,n+d}(t)}{(n+d)} + \left( 1 - \frac{n}{n+d} \right) \tau(m) \right)$ 
10:     $V_4 = V_3 \left( \frac{dT_{1,n+d}(t) - (n+d)T_{n+1,n+d}(t)}{n+d} \right)$ 
11:     $V_1 = S_{1,n+d}(t) - \sum_{i=n+1}^{n+d} \left( w_i(t) - \frac{T_{1,n+d}(t)}{n+d} \right)^2 + n \left( \frac{T_{n+1,n+d}(t)}{n+d} \right)^2 + n \left( 1 - \frac{n}{n+d} \right)^2 \tau(m)^2 +$ 
12:       $V_2 + V_4$ 
13:     $S_{1,n+d}(t + \tau(m)) = V_1 + d(T_{1,n+d}(t) - T_{n+1,n+d}(t) + n\tau(m))$ 
14:  end for
15:   $MaxS = \arg \max_q S_{1,n+d}(t + \tau(q))$ 
16:   $MinS = \arg \min_q S_{1,n+d}(t + \tau(q))$ 
17:  for each frontier cell  $m$  do
18:    if  $m \in \Gamma_t$  then
19:       $U_{WT}^m = \frac{MaxS - S_{1,n+d}(t + \tau(m))}{MaxS - MinS}$ 
20:    else
21:       $U_{WT}^m = -\text{normalized travel cost}$ 
22:    end if
23:  end for
24:   $m^* = \arg \max_m U_{WT}^m$ 
25:  Move robot to new intermediate target point  $m^*$ 
26:  Update map
27: end while

```

conditions are mission and robot type dependent and can be selected during the mission planning stage.

Figs. 7.3 and 7.4 illustrate two, three consecutive step sequences of intermediate target point selection during a compact exploration mission using the proposed utility function. It can be observed that the proposed utility function biases the robot's motion to a roughly circular motion surrounding the starting position resulting in a compact map. In [89] one of the proposed methods is based on forcing the robot's motion to be circular around a given starting position. However, in this proposed utility function, this motion is not a fixed motion and will be adjusted based on the revealed topology of the environment as illustrated by the results of exploring an environment with large blocked area, depicted in Fig. 7.9(b).

Figure 7.3: Left column illustrates the selection of the intermediate target point from the available utility information. The darker and larger pixels indicate higher utility while lighter and smaller pixels indicate low utility for spatially corresponding frontier cells. The selected frontier cell is indicated using the red square. The corresponding occupancy grid maps and the traversed paths of the robot are given in the right column of figures.

7. COMPACT EXPLORATION OF UNBOUNDED ENVIRONMENTS

Figure 7.4: Illustration of intermediate target point selection and the corresponding maps with traversed paths (in red) during another three consecutive decision steps.

7.3.2 Multi robot compact exploration

The utility function proposed can be used by various coordination strategies in multi-robot exploration missions. The algorithms 20 and 21 illustrate two approaches for coordinating multiple robots to conduct compact exploration using the proposed utility function.

7.3.2.1 Utility discounting approach

The exploration strategy given in alg. 20 uses the utility discount mechanism proposed in [5] to coordinate the robots to explore different parts of the environment. It uses the same intermediate target point selection strategy in alg. 19 based on the proposed utility. However, after each selection of an intermediate target, the utility values of all the map cells are updated using the operation in line 12. This is done with the aid of the array J which is initialized to 1.0 at the start of the mission. Values in J are updated, when the grid cells are assigned to robots as intermediate targets, based on observability of grid cells by robots assigned to these intermediate targets. $P(\text{dist}(m^*, m))$ is the probability of robot observing cell m from its new location m^* . Multiplying the utility value for each neighboring grid cell with its corresponding $J(m)$ variable discounts utility values in the neighborhood of the robot that is moving to point m^* thus preventing other robots from selecting points in cell m^* 's neighborhood.

Algorithm 20 Multi Robot Compact Exploration

```

1: Set  $J(m) = 1.0$  for all frontier cells  $m$ 
2: while termination condition is not met do
3:   Update map
4:   Incrementally extract new frontier cells and record time-step value in  $I[c]$  for each new cell  $c$ 
5:   Extract the set of neighboring frontier cells  $\Gamma_t$  from frontier database
6:   Update waiting times  $w_c(t)$  for each cell  $c \in \Gamma_t$ 
7:   Compute  $T_{1,n+d}(t)$  for  $\Gamma_t$ 
8:   Compute  $S_{1,n+d}(t)$  for  $\Gamma_t$ 
9:   Compute  $U_{WT}^m$  for each frontier cell  $m$ 
10:  for each robot  $r$  waiting for next intermediate target do
11:     $m^* = \arg \max J(m)U_{WT}^m$ 
12:     $J(m) = J(m) \times (1 - P(\text{dist}(m^*, m)))$  for each grid cell  $m$ 
13:    Send  $m^*$  as robot  $r$ 's next intermediate target
14:  end for
15: end while

```

7. COMPACT EXPLORATION OF UNBOUNDED ENVIRONMENTS

7.3.2.2 Combining utility discounting with a high level robot re-positioning

The same utility function is integrated to the coordinated multi-robot exploration strategy proposed in [32] to generate another multi-robot compact exploration strategy. This strategy contains a high level re-positioning decision for the robots at certain decision intervals to ensure that the robots continue to map different sections of the map. The high level re-positioning strategy [32] was proposed by the author of this thesis as an extension of the traditional coordinated multi-robot exploration algorithms for outdoor unstructured environments where region assignments for robots cannot be conducted using environmental features. The re-positioning strategy is summarized in the Appendix B. Alg. 21 is similar to Alg. 20 and differs only by the introduction of the re-positioning step in lines 11-13.

Algorithm 21 Multi Robot Compact Exploration w/ Re-positioning

```
1: Set  $J(m) = 1.0$  for all frontier cells  $m$ 
2: while termination condition is not met do
3:   Update map
4:   Incrementally extract new frontier cells and record time-step value in  $I[c]$  for each new cell  $c$ 
5:   Extract the set of neighboring frontier cells  $\Gamma_t$  from frontier database
6:   Update waiting times  $w_c(t)$  for each cell  $c \in \Gamma_t$ 
7:   Compute  $T_{1,n+d}(t)$  for  $\Gamma_t$ 
8:   Compute  $S_{1,n+d}(t)$  for  $\Gamma_t$ 
9:   Compute  $U_{WT}^m$  for each frontier cell  $m$ 
10:  for each robot  $r$  waiting for next intermediate target do
11:    if robot  $r$  needs to be spatially re-positioned then
12:       $m^* =$  best grid cell reported by re-positioning strategy
13:    else
14:       $m^* = \arg \max_m J(m)U_{WT}^m$ 
15:       $J(m) = J(m) \times (1 - P(\text{dist}(m^*, m)))$  for each grid cell  $m$ 
16:    end if
17:    Send  $m^*$  as robot  $r$ 's next intermediate target
18:  end for
19: end while
```

7.4 Experimental evaluation

The use of the proposed utility function is evaluated on three different simulated environments for both single robot and multi robot compact exploration missions. Simulation experiments are conducted using the stage simulator with Player drivers and Orca components as described in [93]. Robots used for all experiments were

equipped with a laser range sensor with a Field of View of 180° and a range of 5 m. The velocity of robots were set to a constant speed of 0.5 m/s. Three identical robots were used for the multi-robot exploration experiments. The three environments used for experiments are depicted in Fig. 7.5.

Figure 7.5: Unstructured environments used for simulated experiments (a) Low obstacle density environment (b) High obstacle density environment (c) Environment with a barricaded region

For each environment type and exploration type (single or multi robot) ten simulations are conducted and the mean and standard deviation values of the metrics to be compared (described below) are computed. Results of single robot exploration are compared with a non-compact exploration algorithm presented in [14] while results for multi-robot exploration are compared with [5]. Both these non-compact algorithms are referred to as normal exploration algorithms from here on-wards.

7. COMPACT EXPLORATION OF UNBOUNDED ENVIRONMENTS

Two metrics are used to measure the compactness of the maps being generated. One is the compactness of the Oriented Bounding Box (OBB) [74] containing the current map. The compactness of the OBB is defined as the ratio between the width and height of the bounding box. The width is always selected to be along the major axis of the OBB while height is selected to be along the minor axis of the OBB.

$$\text{Compactness of Bounding Box} = \frac{\text{length of minor axis of OBB}}{\text{length of major axis of OBB}}$$

A higher value indicates that the map containing explored and unexplored cells is compact. However it is possible to have a higher value for OBB compactness without the explored area actually being compact. Hence a second metric, the ratio of explored cells in the OBB to the total number of cells in the OBB, is used to further elucidate the comparison.

$$\text{Explored Ratio} = \frac{\# \text{ of mapped cells within OBB}}{\# \text{ of total cells within OBB}}$$

7.4.1 Compact Ratio and Explored Ratio of the OBBs of map

The effectiveness of the proposed exploration algorithms in generating compact maps is evaluated using the two metrics defined previously. Therefore, it is important to understand how the compact ratio and the explored ratio metrics evolve during an exploration mission and is detailed below.

7.4.1.1 Comparison of Single Robot Exploration strategies

Figs. 7.6, 7.7 and 7.8 compare the progression of the compact ratio of the bounding box and the exploration ratio of the bounding box for single robot compact and normal exploration strategies for the three environments.

It can be observed from Figs. 7.6(a), 7.7(a) and 7.8(a) that the OBBs generated for maps created from compact strategy has higher compact ratio for all environments compared to the normal strategy. Fig. 7.9 illustrates the maps generated and the paths traveled by the robot during exploration by the compact strategy. Comparing the Fig. 7.9(a) to Fig. 7.1 illustrates that the normal strategy leads the

7.4 Experimental evaluation

Figure 7.6: Comparison of (a) compactness ratio of bounding box and (b) explored ratio in bounding box for the single robot missions in the low obstacle density environment

Figure 7.7: Comparison of (a) compactness ratio of bounding box and (b) explored ratio in bounding box for the single robot missions in the high obstacle density environment

robot in an elongated path resulting in an OBB which has a larger width compared to its height, hence the lower compact ratio. The compact strategy leads robot in a circular path resulting in a map that is balanced in all directions thus resulting in OBBs which resemble squares, hence the higher compact ratio. Observation of the traversed path of the robot in the barricaded environment in Fig. 7.9(b) reveals that the utility function is able to direct the robot in the most efficient path to conduct the compact exploration. In here, when the robot encounters the large obstacle, it turns back instead of moving along the obstacle to reach the other side of the mapped area, which would have been the behavior if a fixed circular motion is used.

The comparisons of the explored ratios in Fig. 7.6(b), 7.7(b) and 7.8(b) do not

7. COMPACT EXPLORATION OF UNBOUNDED ENVIRONMENTS

Figure 7.8: Comparison of (a) compactness ratio of bounding box and (b) explored ratio in bounding box for the single robot missions in the environment with large barricaded region

Figure 7.9: Illustration of the generated map and the path of the robot using the compact exploration strategy in (a) low obstacle environment and (b) environment with the barricaded region. In (b), when the robot encounters the large obstacle (at the top of the mapped area), it turns back and starts exploring in the opposite circular direction.

provide a clear indication of the compact maps generated by the proposed method. This is mainly due to the way the OBBs are generated and should be understood in conjunction with the compact ratios. An OBB for a map is the box with the smallest area that can bound the map. Hence for an elongated map generated by normal strategy as in Fig. 7.1, the OBB closely resembles the actual map. This

results in a higher explored ratio as illustrated by the explored ratio values of initial phases of the exploration missions for all environments. However, as the mission evolves, the robot continues to map other areas and may deviate from its elongated mapped region, due to the presence of obstacles, and this results in making the OBBs relatively squarer compared to earlier OBBs. The slowing of the reduction of the compact ratios of OBBs for the normal strategy indicates this phenomena. As the OBBs transition from rectangular shapes to squarer shapes, the number of unmapped cells within the OBBs increase in normal strategy thus results in low explored ratios. This phenomena is pronounced in the barricaded environment where a large obstacle forces the normal strategy to deviate from elongated (i.e. roughly linear) exploratory motion, more frequently, thus results in increasing the compact ratio and decreasing the explored ratio of the OBBs.

7.4.1.2 Comparison of Multi-Robot Exploration Strategies

The compact ratios and explored ratios of OBBs of maps for the two proposed multi-robot exploration strategies are compared with the normal strategy and the results are given in Figs. 7.10-7.16.

Figs. 7.10-7.12 compare compactness ratios and explored ratios of OBBs of maps between the normal and utility discount based multi robot strategies.

Figure 7.10: Comparison of (a) compactness ratio of bounding box and (b) explored ratio in bounding box for the utility discounting based multi robot missions in low obstacle density environment

It can be observed that of the two ratio metrics, the explored ratio metric provide the best indication of the compactness of the maps generated by the utility discount

7. COMPACT EXPLORATION OF UNBOUNDED ENVIRONMENTS

Figure 7.11: Comparison of (a) compactness ratio of bounding box and (b) explored ratio in bounding box for the utility discounting based multi robot missions in high obstacle density environment

Figure 7.12: Comparison of (a) compactness ratio of bounding box and (b) explored ratio in bounding box for the utility discounting based multi robot missions in the environment with a barricaded region

based compact multi robot exploration strategy. As illustrated in Fig. 7.13, the normal multi robot strategy coordinates robots to map different regions of the environment. This results in the OBBs of the maps becoming more compact similar to the compact multi robot strategy. However, in the normal multi robot exploration strategy the individual robots do not follow a compact mapping strategy, therefore they leave large unmapped regions inside the OBBs resulting in a lower explored ratio compared to the compact multi robot strategy. In contrast, the compact multi robot exploration strategy generates a compact map of the environment while coordinating its exploratory motion to avoid multiple robots mapping the same region thus resulting in a higher explored ratio.

Figure 7.13: Qualitative comparison of the generated maps and the traversed paths of the robots using (a) Normal and (b) Compact exploration strategies

Figs. 7.14-7.16 compare compactness ratios and explored ratios of OBBs of maps between the normal multi-robot strategy and compact multi-robot exploration with re-positioning strategy and results are comparable to the compact multi-robot strategy without the re-positioning strategy. The high level robot re-positioning operation moves robots in such a way that the robots are distributed in a balanced way along the different frontiers thus does not negatively affect the overall compact exploration scheme.

Figure 7.14: Comparison of (a) compactness ratio of bounding box and (b) explored ratio in bounding box for the utility discounting with re-positioning based multi robot missions in low obstacle density environment

7. COMPACT EXPLORATION OF UNBOUNDED ENVIRONMENTS

Figure 7.15: Comparison of (a) compactness ratio of bounding box and (b) explored ratio in bounding box for the utility discounting with re-positioning based multi robot missions in high obstacle density environment

Figure 7.16: Comparison of (a) compactness ratio of bounding box and (b) explored ratio in bounding box for the utility discounting with re-positioning based multi robot missions in the environment with a barricaded region

7.4.2 Compactness of the map

Results for the compactness ratio and explored ratio for the OBBs of maps reported in the previous section illustrated that the compactness ratio of OBBs can be used to measure the level of compact exploration during single robot missions and explored ratio can be used for multi robot missions. Since both of these metrics are normalized, they are multiplied together to generate a single metric that measures the compactness of the maps generated during exploration missions for all different exploration missions.

$$\text{Compactness of Map} = \text{Compact Ratio of OBB} \times \text{Explored Ratio in OBB}$$

Figs. 7.18-7.19 compare the compactness measure of the maps for all experiments conducted and again illustrate that the proposed utility function allows selection of intermediate target points during exploration missions that results in compact maps.

Figure 7.17: Comparison of compactness of maps generated during exploration of the low obstacle density environment with compact (a) 1 robot (b) coordinated 3 robot and (c) coordinated 3 robots with re-positioning strategies

Figure 7.18: Comparison of compactness of maps generated during exploration of the high obstacle density environment with compact (a) 1 robot (b) coordinated 3 robot and (c) coordinated 3 robots with re-positioning strategies

7.5 Summary

This chapter detailed the formulation of a utility function that can be used to replace the traditional gain-cost trade-off based utility to devise exploration strategies that generate compact maps of the environment. The utility function was formalized

7. COMPACT EXPLORATION OF UNBOUNDED ENVIRONMENTS

Figure 7.19: Comparison of compactness of maps generated during exploration of the environment with barricaded region with compact (a) 1 robot (b) coordinated 3 robot and (c) coordinated 3 robots with re-positioning strategies

as the normalized reduction in waiting time variance of neighboring frontier cells. For each frontier cell, waiting time parameter is introduced and its maintenance is derived from the efficient frontier management strategy proposed in the earlier chapters of the thesis. The utility function was integrated to both single and multi-robot exploration strategies and the resulting maps were compared with the maps generated from traditional frontier based explorations. The results from the experiments indicated that the proposed utility function aids in selecting intermediate target points, during exploration missions, that result in compact maps.

Chapter 8

Conclusion and Recommendations

8.1 Conclusion

Based on the premise of *increasing the re-planning capability results in an increase of exploration efficiency*, an efficient frontier cell extraction and management approach to support fast re-planning of exploration actions is presented in this thesis. The approach is based on incrementally processing only the locally updated map data instead of processing the entire occupancy grid map. Comparisons of the proposed approach with the traditional frontier extraction approach reveal that the proposed method is able to generate frontier information more efficiently thus facilitates a high frontier refreshing rate. Comparisons with other state of the art incremental approaches for frontier generation also reveal that the proposed method is more efficient and is superior in the accuracy of frontier cell detection with a consistent detection accuracy of over 95%. In addition, a byproduct of the proposed SRFD strategy is the efficient computation of the reachability map of the 2D environment. This resultant reachability map can be used during later planning stages of an autonomous navigation mission, further improving the computational complexity of the robot's navigation component.

Frequent exploration step re-planning facilitated by the proposed efficient frontier extraction algorithms is evaluated by proposing a new task cancellation strategy, based on a probabilistic decision step. The task cancellation step decides whether the further motion along the planned path to its current intermediate target is re-

8. CONCLUSION AND RECOMMENDATIONS

dundant or not. If the remaining motion is deemed redundant, it is cancelled and a new intermediate target is selected resulting in an exploration task re-planning. Experiments carried out in both real and simulated environments reveal that the proposed re-planning capability further improves the frontier based exploration strategy by avoiding redundant motions of the robot. The results also indicate that the travel distance improvements are more pronounced in environments where robots encounter a lot of dead-end situations, such as indoor environments and disaster hit areas with lots of rubble, indicating that augmenting exploration missions with task cancellation would facilitate efficient initial exploration in search and rescue missions conducted in such environments.

This re-planning capability of the frontier based exploration strategy due to the frequent frontier refreshing is employed to develop a frontier exploration based goal directed navigation strategy. Frontier based exploration is directed towards the goal by computing the utility of each frontier cell as the shortest distance estimate to the goal from the robot through the frontier cell. The shortest distance to the goal from frontier cell through the unknown area is estimated using a visibility graph generated on the unknown area. It was shown that this approach negates the need to possess a large grid for planning and only requires the currently available mapped space. Results from experiments conducted in multiple simulated environments revealed that the directed exploration navigation strategy is identical in travel distance performance to traditional dynamic planning algorithms and is superior in the number of re-planning steps required to reach the goal. Therefore, the proposed navigation strategy is a viable alternative to dynamic planning algorithms.

This thesis also tackles the efficiency of exploration missions conducted in unbounded environments. It was argued that when exploring unbounded environments where the boundaries are not known, the efficiency of exploration must be measured by some means other than the travel distance or the time to complete the mission. The ability to generate usable map information in a balanced nature in all directions (i.e. generating compact maps) is proposed as one such criteria. A minimization of waiting time variance of frontier cells based utility is introduced and it was illustrated that the use of this utility on exploration algorithms results in compact maps of the environment in both single and multi-robot missions. The waiting times of frontier cells are efficiently updated using the efficient frontier management approach

presented in the earlier chapters of the thesis. All of the above reported results indicate that the use of efficient frontier extraction and management algorithms yield more efficient frontier based exploration and navigation missions in both bounded and unbounded environments.

8.2 Recommendations for future research

As with any research, there is room for more extensions and development of the proposed work. Some of these are identified and outlined below.

- As mentioned previously, the resulting reachability information of the grid cells in the environment can be later used to aid more autonomous navigation missions. However, the proposed approach is developed only for 2D occupancy grid maps. With the increasing adoption of low cost quadcopters with limited computing capabilities in urban/built environments where they require navigation in full 3D environments, extending the efficient incremental reachability computation to 3D occupancy grid maps[71] will reduce the computational burden of these robots. The resulting computational gains can be used for other navigational or mission computations, ultimately improving the overall mission performance.
- It was mentioned that combining frontier cells according to their spatial relationships and using these combined information during intermediate target selection steps can improve the exploration efficiency. The proposed SRFD method stores and allows access to frontier information as contours. Frontier contours inherently combine the spatial relationships among nearby frontier cells. Therefore, efficient access to frontier contours without the need to re-compute them at each map update step can be used to develop more efficient frontier based exploration missions and warrants more examination.
- Since frontier contours evolve over time and expand outwards from the robot, generation of the frontier contours at high frequency using the SRFD method can be considered as propagating a wavefront across the map and can be used to efficiently approximate the geodesic distances between the robot and

8. CONCLUSION AND RECOMMENDATIONS

frontiers and between different frontiers. In exploration algorithms that look ahead more than one sensing step [46], computing the distances between different frontiers is one of the major performance bottlenecks. Hence SRFD based efficient distance approximations could be used to further improve the efficiency of such exploration missions.

- The compact exploration strategy proposed in this thesis is purely based on the design of the utility function. While the minimizing waiting time variance based utility function drives robots in a compact exploratory motion, the presence of large objects relative to the sensor range of the robot forces the robot to travel through already explored regions for longer periods, resulting in reduction of the expansion rate of the map. Integrating this strategy with a better path planning algorithm that balances the navigation through explored and unexplored regions in the map could yield more efficient compact exploration strategies.
- The compactness value proposed to measure the efficiency of exploration missions in unbounded environments is based on the Oriented Bounding Box (OBBs) of the mapped area. Selecting the next intermediate target based on the improvements to the compactness of the OBB of the mapped area could be considered as an alternative to the waiting time variance based approach. As part of the boundary of the mapped area is given by the frontier cells, efficient computation of the OBBs using the efficiently available frontier information warrants examination and could result in an alternative utility for compact exploration missions.

Appendix A

Oriented Bounding Box based Frontier Detector (OFD)

This chapter summarizes an earlier approach by the author for efficient detection and management of frontier cells[31]. The approach described in this chapter, OFD, is one of the frontier extraction algorithms employed during the comparison of computational efficiency and detection accuracy of the proposed SRFD method in Chap. 4.

A.1 Updating a frontier contour between two map update steps

OFD is also based on incremental update algorithms. Therefore, sensor updates between two map update steps are considered for each incremental step of the algorithm. Three types of frontier-contour updates can be recognized between two map update steps (i.e. update window). The first type of update occurs when a frontier-contour is completely replaced by a new one due to robot movement directly towards the contour. These types of updates usually occur when a robot traverses through a corridor or tunnel like environment. The second type of frontier-contour update results in deleting a portion of the original contour from one corner and connecting a new frontier-contour to the it. In the final update type, the original

A. ORIENTED BOUNDING BOX BASED FRONTIER DETECTOR (OFD)

frontier is split in to two contours initially by deleting the sub-contour at the middle. Since robot is merely extending the frontier boundary, the new frontier-contour gets attached to both contours generated by the previous splitting process. The last two update types usually occur in environments which are relatively large in comparison to the robot’s sensor range such as in large rooms and outdoors.

It can be noted that map updates between two map update steps modify only a portion of the map and only a subset of the frontier-contours. Therefore, in order to minimize the processing required to generate frontier-contours at each update step, only the cells that changed their statuses from/to frontier/non-frontier from the last map update step need to be identified. Once identified, these changes can be used to update a frontier-contour database from the last update step, to generate the accurate database of frontier-contours for the current update step.

A.2 Tracking cell status changes through an update window

Monitoring the change of a single cell’s *occ* does not provide complete information about a cell becoming a new frontier or losing its frontier status. For example, a cell could change its *occ* from UNKNOWN to FREE upon a map update. But whether it becomes a frontier cell or not is also dependant on the values of its neighboring cells. Therefore, three new classes are introduced to indicate the status of the cells. Grid cells belonging to *Probable Frontier* (PF) class are cells that have a collection of free and unknown cells in its 8-connected neighborhood (i.e. $\mathcal{N}_8(c)$ for cell c). Neighborhoods of the cells belonging to *Unexplored* class (U) are completely categorized as unknown while neighborhoods of cells belonging to *Explored* class (E) are a combination of only occupied and free cells. When a single cell’s categorization changes, it affects statuses of the 9 neighborhoods it is part of. After each sensor update, *occ* of some cells would change which will result in change of statuses of some cells. Any cell that changes its status from U or E to PF is a candidate for a new frontier cell while any cell that changes state from PF to U or E is a candidate that would lose its frontier status after the map update.

A.2 Tracking cell status changes through an update window

To track changes of cell statuses from/to frontier status between two map update steps, two sets are introduced together with two additional variables for each grid cell. The two sets named *New Frontier Set (NFS)* and *To-Be Deleted Set (TDS)* store the grid cells that probably became frontiers and lost frontier status respectively during a single update window. The additional variables are denoted as $C_{o,i}$ and $C_{c,i}$ respectively for each cell i . $C_{o,i}$ stores the *old* class that was associated with the grid cell at the conclusion of the previous map update step. And $C_{c,i}$ stores the *current* class associated with the cell after the latest sensor update on the map. Note that with each *new* sensor update on the map, $C_{c,i}$ is updated with the new classification $C_{n,i}$ of cell i . Using these variables and the two sets, status of a grid cell can be tracked from one map update step to another through a series of map updates. At the next map update step, the two sets NFS and TDS contain the final set of cells that need to be considered for updating the frontier database. The four rules used to update NFS and TDS are described below.

Rule 1 If $C_{o,i} = PF$ and $C_{c,i} = PF$ and $C_{n,i}$ is (U or E), then add cell i to TDS.

Rule 2 If $C_{o,i} = (U \text{ or } E)$ and $C_{c,i} = PF$ and $C_{n,i}$ is (U or E), then delete cell i from NFS.

Rule 3 If $C_{o,i} = PF$ and $C_{c,i} = (U \text{ or } E)$ and $C_{n,i} = PF$, then delete cell i from TDS.

Rule 4 If $C_{o,i} = (U \text{ or } E)$ and $C_{c,i} = (U \text{ or } E)$ and $C_{n,i} = PF$, then add cell i to NFS.

Rule 1, is applied when a cell that was marked as a probable frontier at the previous map update step (i.e $C_{o,i} = PF$) gets categorized as U or E for the first time. Hence, at the next step, it has to be considered for deletion from the frontier database, thus is added to TDS. According to Rule 2, the cell has been a non frontier at the previous map update step but has become a PF during one of the map updates, therefore has been added to NFS. Since the new sensor update reclassifies it as a non PF (i.e. $C_{n,i} = U \text{ or } E$), cell has to be deleted from the the NFS. Similarly, in Rule 3, cell has been a probable frontier at the previous map update step and has been classified as non PF during one of the sensor updates after the map update step. Therefore it was added to the TDS to be considered for deletion from frontier database. Now with the new sensor update, it has been reclassified

A. ORIENTED BOUNDING BOX BASED FRONTIER DETECTOR (OFD)

as a PF which requires the cell to be deleted from the TDS. Both Rule 3 and 4 are introduced as safe guards against sudden *occ* changes that occur with robot's pose updates. These usually occur due to sensor noise or mapping errors induced due to localization uncertainties. Rule 4 is used when a cell has been classified as a non frontier in the previous map update step and has not changed its status during previous sensor updates. With the new sensor update, it has been classified as a PF thus should be added to the NFS, to be considered for a new frontier cell.

A.3 Oriented bounding boxes (OBBs) for frontier-contour generation

The set NFS contains all the new cells that are classified as PF. Each PF cell's 3x3 neighborhood has a non-zero number of free and unknown cells. A frontier cell is a free cell which has an unknown cell as an 8-connected neighbor. Therefore, if $\mathcal{F}_k$ is the new set of frontier cells appeared after the map update step k . $\mathcal{F}_k \subset NFS$. The set NFS can be partitioned into a collection of connected sets $\mathcal{C}_{NFS} = \cup_j \mathcal{C}_{NFS,j}$. Hence it follows that $\mathcal{C}_{w,i} \subset \mathcal{C}_{NFS,j}$ where $\mathcal{C}_{k,i}$ is a new frontier-contour i generated at step k . Therefore, frontier-contour generation algorithm can be executed on individual connected sets of NFS. Since frontier-contours are extracted using a combination of segmentation and contour extraction algorithms present in Image Processing literature[36], a rectangular bounding box is created for each $\mathcal{C}_{NFS,j}$. To reduce the computational load for each $\mathcal{C}_{NFS,j}$, an Oriented Bounding Box (OBB) [74] is generated. An OBB is a rectangle covering the entire set of cells concerned, with the smallest area. Cells in map coordinate frame M can be mapped to OBB coordinate frame O using,

$$X^O = \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} (X^M - \mu) = V X^M \quad (\text{A.1})$$

where V_1 and V_2 are the major and minor axes of the OBB respectively. Principal Component Analysis can be used to generate these two vectors and the origin of the OBB coordinate frame is placed at μ which is the mean vector of the set of cells

concerned. The reverse mapping from OBB coordinates to original map coordinate frame can be done by

$$X^M = V^{-1}X^O + \mu \quad (\text{A.2})$$

Therefore, frontier-contours are generated using individual OBBs and are then remapped to the original coordinate frame. Fig. A.1 illustrates an example of OBBs generated for four connected cells sets in NFS.

Figure A.1: Example OBBs : Dark points are the PF cells which encompass frontier cells. Actual frontiers can be extracted by processing individual OBBs

A.4 Implementation

A.4.1 Frontier Database

The data structure that is used to store the frontier-contours must have an efficient set of update operations since multiple addition and deletion operations take place during a single update window and must be executed at each map update step. In addition, the data structure must provide exploration component with efficient access to all the stored frontiers. The same database setup proposed in [29] is used with a modification to the way the index is arranged to access frontiers. For each grid cell in the map, an associated frontier-contour ID is stored in an array F and this ID can be accessed in $\mathcal{O}(1)$ given the cell. Unlike in [29], where the mapping is from an ID to a set of cells, mapping is made from an ID to a single cell. To access

A. ORIENTED BOUNDING BOX BASED FRONTIER DETECTOR (OFD)

all the cells associated with the ID, a flood-fill[36] operation has to be carried out with the mapped cell as the seed point.

According to Sec. A.1, database must support three types of modifications to existing frontier-contours. Complete deletion of contours result in removing a mapping and can be done in $\mathcal{O}(n)$. Extending a frontier contour requires removal of an existing mapping which requires $\mathcal{O}(n)$ and addition of a new mapping which has a complexity of $\mathcal{O}(n)$. This is in contrast to the $\mathcal{O}(n \lg n)$ complexity involved in frontier-contour splitting and merging operations on the map-set arrangement in [29].

A.4.2 Algorithms

At a new map update step, first the cells that lose their frontier status are deleted from the frontier database. This is done by calling `DELETECANCELLEDFRONTIERS`, given in Alg. 22. Once cancelled frontiers are deleted, newly generated frontiers must be added to the database. This is done by calling `ADDANDMERGEFRONTIERS` function described in Alg. 23. At the end of this step, frontier database has been updated to the latest available frontier-contours in the map. Finally, $C_{o,i}$ value for each modified cell i is replaced with the associated $C_{c,i}$ to initialize tracking of modified cells for the next update window.

During the deletion process in Alg. 22, each cell c in TDS is checked for being stored as a frontier cell in the array F . If $F[c]$ is valid, cell c is actually a frontier and can be deleted. However, frontiers stored in the database are sometimes approximations of actual frontiers due to the nearest neighbor mapping used for OBB, hence may be missed by the TDS. Therefore to minimize the number of missed deletions, the set TDS is dilated[36] at the start of the deletion algorithm. Then individual deleted contours are identified. For each deleted frontier-contour segment, a maximum of two unique adjacent non-deleted frontier cells are searched in the database. If none is found, this is associated with an entire frontier-contour being removed. Hence the database mapping associated with the contour is removed in line 21. One adjacent cell is found when the contour-segment is deleted from only one corner of an existing frontier-contour. Then the remaining segment of the contour is remapped

Algorithm 22 DELETECANCELLEDFRONTIERS

```

1:  $TDS \leftarrow DILATE(TDS)$  from  $TDS$ 
2:  $Q_f \leftarrow$  All deleted frontier contours  $f_d$ 
3:  $F[c] \leftarrow NIL$  for all valid cells  $c \in f_d$  for all  $f_d \in Q_f$ 
4: for each  $f_d$  do
5:    $C \leftarrow$  Find non deleted points adjacent to  $f_d$  {Maximum cardinality of  $C$  is two}
6:   if  $|C| = 1$  then
7:     Remove Database mapping for  $F[c \in C]$ 
8:     Add Database mapping  $MAP(\mathcal{C}_{ID}, c)$ 
9:     Mark frontier cells connected to  $c$  with  $\mathcal{C}_{ID}$ 
10:     $\mathcal{C}_{ID} \leftarrow \mathcal{C}_{ID} + 1$ 
11:   else if  $|C| = 2$  then
12:     for each  $c \in C$  do
13:       if  $F[c] = ID(f_d)$  then
14:         Remove Database mapping for  $F[c]$ 
15:         Add Database mapping  $MAP(\mathcal{C}_{ID}, c)$ 
16:         Mark frontier cells connected to  $c$  with  $\mathcal{C}_{ID}$ 
17:          $\mathcal{C}_{ID} \leftarrow \mathcal{C}_{ID} + 1$ 
18:       end if
19:     end for
20:   else
21:     Remove Database mapping for  $ID(f_d)$ 
22:   end if
23: end for

```

with the new ID as in lines 7-10. When two cells are found, this indicates the deletion of a frontier segment from the center of an existing contour. Before remapping each remaining part of the original contour, a check is made to ensure that it was not remapped by another splitting operation. If not remapped, it is remapped to a new contour ID in lines 14-17. Multiple splittings of the same contour result in some sub-contours identified in line 5 have already been remapped hence, the additional check removes an unnecessary remapping. $\mathcal{C}_{ID}$ is a global variable that is used as a counter to generate unique frontier-contour IDs.

After deleting all the cancelled frontier-contours, new frontier-contours are generated and added to the database using the algorithm 23. As mentioned in A.3, a separate OBB is generated for each connected cell set $\mathcal{C}_{NFS,j}$ and frontier contour generation is carried out on each OBB. The resultant contours are remapped to the original coordinate frame. As remapping may cancel connectedness of some of the contours, additional cells are marked as frontier cells to ensure each contour is a connected set of cells as in line 6. Then for each new contour f , all the adjacent contours are recorded. If such contours are found, these are existing frontiers of which the new frontier is an extension. Hence all of these contours are merged into a single frontier-contour and a new mapping is added (lines 8 - 17). If no adjacent

A. ORIENTED BOUNDING BOX BASED FRONTIER DETECTOR (OFD)

Algorithm 23 ADDANDMERGEFRONTIERS

```

1:  $\mathcal{C}_{NFS} \leftarrow$  Find connected cell sets in NFS
2: for each  $C_s \in \mathcal{C}_{NFS}$  do
3:    $B_s \leftarrow$  Find OBB for  $C_s$ 
4:    $\mathcal{F} \leftarrow$  Generate frontier contours from  $B_s$  {contours are converted to Map coordinates}
5:   for each  $f \in \mathcal{F}$  do
6:     Make  $f$  is connected
7:      $\mathcal{J}_N \leftarrow$  Record all frontiers that are adjacent to  $f$ 
8:     if  $|\mathcal{J}_N| > 0$  then
9:       for each  $id \in \mathcal{J}$  do
10:         $c \leftarrow$  Get mapping for  $id$  from Database
11:         $F[c'] \leftarrow \mathcal{C}_{ID}$  for all frontier cells  $c'$  in the same contour as  $c$ 
12:        Remove Database mapping for  $id$ 
13:      end for
14:    end if
15:     $F[c_i] \leftarrow \mathcal{C}_{ID}$  for all  $c_i \in f$ 
16:    Add Database mapping  $\text{MAP}(\mathcal{C}_{ID}, c_o)$  { $c_o$  is any cell in  $f$ }
17:     $\mathcal{C}_{ID} \leftarrow \mathcal{C}_{ID} + 1$ 
18:  end for
19: end for

```

contour is found, this indicates a new frontier-contour generation step, which follows after a complete deletion of an existing frontier, and the this new contour is added in lines 15 - 17.

Appendix B

Re-positioning of robots for efficient multi-robot exploration

This chapter summarizes the robot re-positioning strategy used during evaluations of the utility function proposed in Chap. 7, for multi robot scenarios. This is a research carried out by the author to improve the efficiency of multi-robot exploration missions in unstructured environments where boosting or lessening the utilities of frontier cells according to semantic labels of map features cannot be conducted due to the unstructured nature of the environment[32]. This work proposes to use a two level strategy where the traditional low-level exploration of each robot is interrupted momentarily to re-position the robots in such a way they are distributed evenly along the available frontiers.

B.1 Re-positioning robots for coordinated exploration

In unstructured environments, explicitly assigning regions for different robots is impossible due to the lack of identifiable semantic labels. Therefore, instead of considering the global distribution of robots as an explicit region assignment problem followed by exploration, it can be considered as a momentary re-positioning of robots to regions without fixed boundaries, preceded and followed by exploration.

B. RE-POSITIONING OF ROBOTS FOR EFFICIENT MULTI-ROBOT EXPLORATION

The re-positioning of the robots to regions can be modeled to provide a balanced distribution of the robots along frontier regions. Since the re-positioning approach does not explicitly assign robots any boundaries, robots can employ a localized coordination strategy, such as the one presented in [5, 44], to carry out the exploration task without any virtual boundary restrictions once the robots are re-positioned. This hierarchical approach ensures a global distribution of robots through the re-positioning strategy and a localized distribution through classical coordination strategies.

The re-positioning of the robots is achieved by generating a configuration X^* of robots which maximizes the measure $\mathbf{J}(\mathbf{X})$ that represents the balanced distribution of the robots along the frontiers as in eq. B.1. A configuration X is a vector of robot positions.

$$X^* = \arg \max_{X \in \mathcal{X}} \mathbf{J}(X) \quad (\text{B.1})$$

The search space $\mathcal{X}$ can be discretized and team optimality can be relaxed to generate a computationally tractable solution that re-positions individual robots. The new criteria is given in eq. B.2 where robot j is re-positioned to a region v^* and its associated target point x_{j,v^*} given the target points of the rest of the robots, denoted by X_{-j} .

$$(v^*, x_{j,v^*}) = \arg \max_{v \in \Upsilon, x_{j,v} \in X_{j,v}} \mathbf{J}(v, x_{j,v} | X_{-j}) \quad (\text{B.2})$$

Υ is the set of regions that robot j is capable of traversing. Each such region v has an associated target point $x_{j,v}$ for robot j and this is selected to be the closest candidate target point in v for robot j . Once robot j is re-positioned with respect to the rest of the robots, it can continue the exploration task using the classical localized coordination strategy.

B.2 Discretization of map space

Re-positioning robots to different frontier regions requires the higher level strategy to have an abstract representation of the map. Since it is assumed that all robots have the same sensor span, a simple block based decomposition of the map can be

B.2 Discretization of map space

used to abstract away the lower level occupancy grid values and provide a high level representation of different regions of the environment. In this implementation, map space is decomposed into equal sized square blocks to generate a higher level region based representation. The length of the square block is selected to be twice the size of the maximum sensor range (i.e. $2R_{max}$) of the robots. Smaller block sizes (e.g R_{max}) do not provide the necessary separation and this can lead to increased sensor overlaps which requires a specific discounting mechanism. While block sizes which are larger than $2R_{max}$ provide better separation of space, they are incapable of efficiently representing the frontier boundaries due to extreme level of abstraction. Due to above reasons, a medium sized block of the size of $2R_{max}$ is chosen that provides a better separation of map space while providing a reasonable abstraction for the frontier regions.

Each block in the decomposition has an associated *explored ratio* (r). This ratio r is the ratio of grid cells in the block marked as explored to the total number of grid cells contained in the block. The grid cells trapped inside obstacle boundaries are also treated as explored. This ratio can then be used for a coarse categorization of different blocks as in eq. B.3. A block with r value which is larger than 0 and less than 1 with tolerance ϵ contains both explored and unexplored grid cells thus very likely to have a traversable boundary between the explored and unexplored (i.e. frontier). This categorization provides an efficient access into frontier regions through *Possible Frontier* blocks.

$$\text{Block Type} = \begin{cases} \text{Explored} & \text{if } 1 - r < \epsilon \\ \text{Unexplored} & \text{if } 0 < r < \epsilon \\ \text{Possible Frontier} & \text{else} \end{cases} \quad (\text{B.3})$$

ϵ is a tolerance for the ratio and is experimentally selected to be 0.05. Since the categorization of Possible Frontier Blocks is made purely based on the r value, an actual frontier extraction operation is performed on these Possible Frontier Blocks to extract the blocks with frontiers, identified Frontier Blocks.

B. RE-POSITIONING OF ROBOTS FOR EFFICIENT MULTI-ROBOT EXPLORATION

B.3 Calculation of the utility for a frontier block

The above discretization allows the robots to treat regions as blocks and each robot makes its re-positioning decision by considering the utility of re-positioning the robot to these frontier blocks. This utility for a frontier block v is calculated as follows.

$$J(v, x_{j,v} | X_{-j}) = \omega_1 I(v) - \omega_2 \text{Cost}(x_{j,v}) - \omega_3 \xi(x_{j,v}) \quad (\text{B.4})$$

$I(v)$ is the information gain expected by the robot by visiting the frontier block v and $\text{Cost}(x_{j,v})$ is the cost associated with traveling to the target point $x_{j,v}$ associated with v . These two components provide the necessary trade-off between the information gain and the travel cost similar to the utility function defined in [5]. The global distribution of the robots in the environment is controlled by the *influence*, $\xi(x_{j,v})$, on the target point $x_{j,v}$ by the rest of the robots and this is explained in detail in sec. B.3.2. ω_i values, where $i = \{1, 2, 3\}$, are positive weights that can be varied to change the importance of the components that make up the utility value.

B.3.1 Information gain

The information gain for a frontier block can be defined as the fraction of unexplored area as in eq. B.5.

$$I(v) = 1 - r(v) \quad (\text{B.5})$$

However, the value calculated in eq. B.5 does not correctly represent the actual information gain for a robot traveling to a region marked as a frontier block in certain situations. For example, consider the situation depicted in fig. B.1.

Figure B.1: Information Gain of a Frontier Block

B.3 Calculation of the utility for a frontier block

In this situation, the frontier block at middle left is almost explored (White area represents already explored space). But the block to the east of this block is completely unexplored. Therefore, influence of the neighboring blocks on the information gain of the robot also need to be considered. Therefore, the new information gain of the block is defined as the summation between information gain of the block and the average information gain of the neighborhood of block v as given in eq. B.6.

$$I(v) = \begin{cases} I'(v) & \text{if } I'(v) < 1 \\ 1 & \text{if } I'(v) \geq 1 \end{cases} \quad (\text{B.6})$$

$$I'(v) = 1 - r(v) + \frac{\sum_{n \in \eta} (1 - r(n))}{|\eta|} \quad (\text{B.7})$$

$$\eta = \{n : \text{Neighbor of block } v \wedge (1 - r(n) > \epsilon)\} \quad (\text{B.8})$$

However, note that unlike in [20], the information gain regions are not discounted. Discounting is used in [20] to avoid multiple robots from reaching the same neighborhood. However, discounting over immediate neighborhoods does not facilitate a satisfactory global distribution of robots over all the frontier boundaries. Therefore, a separate measure called *Influence Factor* is included in the utility to facilitate a more balanced distribution of the robots and is explained next.

B.3.2 Influence factor

Influence Factor is a heuristic used to provide a measure of the balanced distribution of the robots through the frontier regions. The concept of influence by robots on candidate target points in frontier blocks is similar to the idea of potential fields [94] in robot motion planning. It is considered that each robot has a certain influence on the nearby frontier blocks. A higher influence on a frontier block by a robot indicates a high possibility of this frontier block being visited by the robot in the future and vice versa. Considering $\mathbf{R}$ be the set of all the robots, the influence by a robot $i \in \mathbf{R} \setminus \{j\}$ on a frontier block v and its associated candidate target point $x_{j,v}$ is dependant on the maximum sensor range of i ($R_{max,i}$) and the travel distance $d(x_i, x_{j,v})$ and is given by eq. B.9.

B. RE-POSITIONING OF ROBOTS FOR EFFICIENT MULTI-ROBOT EXPLORATION

$$\xi_i(x_v) = \begin{cases} 1 & \text{if } d(x_i, x_v) \leq R_{max,i} \\ e^{-\frac{d(x_i, x_v)}{\lambda R_{max,i}}} & \text{else} \end{cases} \quad (\text{B.9})$$

Note that the influence by a robot on a candidate point is calculated based on its current target point x_i and not on its current position. Since robots are on the move, it is important to capture the dynamic nature of the exploration process when influences on frontier blocks are calculated. This is illustrated in fig. B.2. Since the influence value on a candidate target point is a distance based measure, target points p_2, p_3 have higher influence values while points p_5, p_6 have lower influence values induced by the robot.

Figure B.2: Influence on frontiers by a robot moving to a particular target point

The influence on a particular frontier block v and its associated candidate target point $x_{j,v}$ is the summation of the influences on the block by all robots in the environment except the robot j , which is engaged in the decision making process. (eq. B.10). Since a higher influence value reduces the utility, the re-positioning strategy favors frontier blocks which are less influenced by other robots and, hence re-positions the robot j to have a more balanced distribution of the robots along frontiers.

$$\xi(x_v) = \sum_{i \in \mathbf{R} \setminus j} \xi_i(x_v) \quad (\text{B.10})$$

B.4 Strategy level decider

Once a robot reaches its current target point, the robot can continue the exploration using its localized coordination strategy or re-position itself with respect to other

robots. Prolonged use of localized coordination strategy can induce inefficient local grouping of robots. Therefore a limit, κ , is introduced for the total number of continuous invocations of localized coordination strategies (k_i for robot i), in between two re-positioning commands. Once this threshold is reached, the robot will re-position itself in the environment which provides a better global distribution of the robots. Longer distances to candidate target points induce larger cost values in the reward function. Larger cost values degenerate the lower level coordination strategy [5] to a closest frontier coordination strategy. This reduces the effectiveness of the localized coordination strategy over large distances. Therefore, when the distance to the next target point exceeds a particular threshold D , the higher level strategy is selected to ensure a better distribution of the robots. These rules are summarized in eq. B.11

$$L = \begin{cases} \text{Lower Level} & \text{if } k_i < \kappa, \text{ and } d(x_i, p_i) < D \\ \text{Higher Level} & \text{if } k_i \geq \kappa \text{ or } d(x_i, p_i) \geq D \end{cases} \quad (\text{B.11})$$

Any traditional exploration strategy can be used as the low level strategy of robots. In this thesis, the compact exploration strategy is used as the low level exploration strategy.

Appendix C

Author's Publications

Journal papers

- P.G.C.N. Senarathne and Danwei Wang, “Incremental Algorithms for Safe and Reachable Frontier Detection for Robot Exploration”, Robotics and Autonomous Systems (In Press).

Conference papers

- P.G.C.N. Senarathne and Danwei Wang. “Frontier based Exploration with Task Cancellation,” in Proceedings of IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), 2014.
- P.G.C.N. Senarathne and Danwei Wang. “Directed Exploration for Goal Oriented Navigation in Unknown Environments”, in Proceedings of IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), 2014.
- P.G.C.N. Senarathne, Danwei Wang, Zhuping Wang, and Qijun 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, pp. 114-119. IEEE, 2013.

C. AUTHOR'S PUBLICATIONS

- P.G.C.N. Senarathne, Danwei Wang, and Han Wang. “A new gain function for compact exploration.” In Control Automation Robotics & Vision (ICARCV), 2012 12th International Conference on, pp. 650-655. IEEE, 2012.
- P.G.C.N. Senarathne and Danwei Wang, “A two-level approach for multi-robot coordinated exploration of unstructured environments”, in Proceedings of the 27th Annual ACM Symposium on Applied Computing. ACM, 2012, pp. 274-279.
- P.G.C.N. Senarathne, W. S. Wijesoma, K. W. Lee, B. Kalyan, M. Moratuwage, N. M. Patrikalakis, and F. S. Hover, Marinesim: Robot simulation for marine environments, in OCEANS 2010 IEEE-Sydney. IEEE, 2010, pp. 15.

References

- [1] G. Dissanayake, J. Paxman, J. V. Miro, O. Thane, and H.-T. Thi, “Robotics for urban search and rescue,” in *Industrial and Information Systems, First International Conference on*. IEEE, 2006, pp. 294–298. [1](#)
- [2] D. Calisi, A. Farinelli, L. Iocchi, and D. Nardi, “Multi-objective exploration and search for autonomous rescue robots,” *Journal of Field Robotics*, vol. 24, no. 8-9, pp. 763–777, 2007. [1](#)
- [3] C. Weisbin and G. Rodriguez, “NASA robotics research for planetary surface exploration,” *Robotics & Automation Magazine, IEEE*, vol. 7, no. 4, pp. 25–34, 2000. [1](#)
- [4] A. Bennett and J. Leonard, “A behavior-based approach to adaptive feature detection and following with autonomous underwater vehicles,” *Oceanic Engineering, IEEE Journal of*, vol. 25, no. 2, pp. 213–226, April 2000. [1](#)
- [5] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, “Collaborative Multi-Robot Exploration,” in *Proceedings IEEE International Conference on Robotics and Automation*, vol. 1, 2000, pp. 476–481. [2](#), [12](#), [24](#), [25](#), [28](#), [148](#), [161](#), [163](#), [186](#), [188](#), [191](#)
- [6] T. Edlinger and E. von Puttkamer, “Exploration of an indoor-environment by an autonomous mobile robot,” in *Intelligent Robots and Systems '94. 'Advanced Robotic Systems and the Real World', IROS '94. Proceedings of the IEEE/RSJ/GI International Conference on*, vol. 2, sep 1994, pp. 1278–1284 vol.2. [2](#)

REFERENCES

- [7] G. Oriolo, M. Vendittelli, L. Freda, and G. Troso, “The SRT method: randomized strategies for exploration,” in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 5, april-1 may 2004, pp. 4688–4694 Vol.5. [2](#)
- [8] A. Franchi, L. Freda, G. Oriolo, and M. Vendittelli, “The Sensor-based Random Graph Method for Cooperative Robot Exploration,” *Mechatronics, IEEE/ASME Transactions on*, vol. 14, no. 2, pp. 163–175, april 2009. [2](#)
- [9] B. Kuipers and Y.-T. Byun, “A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations,” *JOURNAL OF ROBOTICS AND AUTONOMOUS SYSTEMS*, pp. 47–63, 1991. [2](#)
- [10] S. S. Ge, Q. Zhang, A. T. Abraham, and B. Rebsamen, “Simultaneous Path Planning and Topological Mapping (SP2ATM) for environment exploration and goal oriented navigation,” *Robotics and Autonomous Systems*, vol. 59, no. 3-4, pp. 228–242, 2011. [2](#)
- [11] H. H. Gonzalez-Banos and J.-C. Latombe, “Navigation Strategies for Exploring Indoor Environments,” *The International Journal of Robotics Research*, vol. 21, no. 10-11, pp. 829–848, Oct. 2002. [2](#), [23](#), [24](#), [26](#), [106](#), [148](#)
- [12] A. Elfes, “Using occupancy grids for mobile robot perception and navigation,” *Computer*, vol. 22, no. 6, pp. 46–57, 1989. [2](#), [11](#), [128](#)
- [13] B. Yamauchi, “A frontier-based approach for autonomous exploration,” in *Proceedings., 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Jul. 1997, pp. 146–151. [2](#), [11](#), [21](#), [25](#), [110](#), [157](#)
- [14] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT Press, 2005. [2](#), [11](#), [21](#), [24](#), [65](#), [95](#), [148](#), [163](#)
- [15] M. Baglietto, M. Paolucci, L. Scardovi, and R. Zoppoli, “Information-based multi-agent exploration,” in *Robot Motion and Control, 2002. RoMoCo'02. Proceedings of the Third International Workshop on*. IEEE, 2002, pp. 173–179. [2](#), [22](#)

-
- [16] A. Makarenko, S. Williams, F. Bourgault, and H. Durrant-Whyte, “An experiment in integrated exploration,” in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 1, 2002, pp. 534–539 vol.1. [2](#), [22](#), [29](#)
 - [17] F. Bourgault, A. A. Makarenko, S. B. Williams, B. Grocholsky, and H. F. Durrant-Whyte, “Information Based Adaptive Robotic Exploration,” 2002, pp. 540–545. [2](#), [22](#)
 - [18] F. Amigoni, V. Caglioti, and U. Galtarossa, “A Mobile Robot Mapping System with an Information-Based Exploration Strategy,” in *International Conference on Informatics in Control, Automation and Robotics*, 2004, pp. 71–78. [2](#), [22](#)
 - [19] C. Stachniss, G. Grisetti, and W. Burgard, “Information Gain-based Exploration Using Rao-Blackwellized Particle Filters,” in *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005. [2](#), [22](#), [29](#)
 - [20] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, and et al., “Coordination for multi-robot exploration and mapping,” in *Proceedings of the Seventeenth National Conference on Artificial Intelligence, AAAI*, 2000. [2](#), [23](#), [24](#), [25](#), [27](#), [28](#), [148](#), [189](#)
 - [21] D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, and B. Stewart, “Distributed multi-robot exploration and mapping,” *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1325–1339, Jul. 2006. [2](#), [26](#), [28](#)
 - [22] J. Faigl and M. Kulich, “On determination of goal candidates in frontier-based multi-robot exploration,” in *Mobile Robots (ECMR), 2013 European Conference on*. IEEE, 2013, pp. 210–215. [3](#), [26](#)
 - [23] A. Mobarhani, S. Nazari, A. H. Tamjidi, and H. D. Taghirad, “Histogram based frontier exploration,” in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 1128–1133. [3](#), [26](#)
 - [24] R. Zlot, A. T. Stentz, M. B. Dias, and S. Thayer, “Multi-robot exploration controlled by a market economy,” in *Proceedings IEEE International Conference on Robotics and Automation*, vol. 3, 2002, pp. 3016–3023. [3](#), [26](#), [28](#)

REFERENCES

- [25] C. Stachniss, O. Mozos, and W. Burgard, “Speeding-up multi-robot exploration by considering semantic place information,” in *Proceedings IEEE International Conference on Robotics and Automation*, May 2006, pp. 1692–1697. [3](#), [27](#), [148](#)
- [26] K. M. Wurm, C. Stachniss, and W. Burgard, “Coordinated Multi-Robot Exploration using a Segmentation of the Environment,” in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2008, pp. 1160–1165. [3](#), [27](#), [28](#), [148](#)
- [27] D. Holz, N. Basilico, F. Amigoni, and S. Behnke, “Evaluating the Efficiency of Frontier-based Exploration Strategies,” in *Proceedings of Joint 41st International Symposium on Robotics and 6th German Conference on Robotics*, Jun. 2010. [3](#), [110](#), [148](#)
- [28] F. Amigoni, A. Q. Li, and D. Holz, “Evaluating the impact of perception and decision timing on autonomous robotic exploration,” in *Mobile Robots (ECMR), 2013 European Conference on*. IEEE, 2013, pp. 68–73. [3](#), [4](#), [107](#)
- [29] M. Keidar and G. A. Kaminka, “Robot exploration with fast frontier detection: theory and experiments,” in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, ser. AAMAS ’12. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 113–120. [3](#), [30](#), [110](#), [181](#), [182](#)
- [30] R. Grabowski, P. Khosla, and H. Choset, “Autonomous exploration via regions of interest,” in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 2. IEEE, 2003, pp. 1691–1696. [4](#), [148](#)
- [31] 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. [9](#), [32](#), [177](#)
- [32] P. Senarathne and D. Wang, “A two-level approach for multi-robot coordinated exploration of unstructured environments,” in *Proceedings of the 27th Annual*

REFERENCES

- ACM Symposium on Applied Computing*. ACM, 2012, pp. 274–279. 9, 26, 162, 185
- [33] J.-C. Latombe, *Robot Motion Planning*, ser. The Springer International Series in Engineering and Computer Science. Springer US, 1996, vol. 124. 13
- [34] G. Grisetti, C. Stachniss, and W. Burgard, “GMapping.” [Online]. Available: <http://openslam.org/gmapping.html> 16, 65, 95, 129
- [35] S. Kohlbrecher and J. Meyer. Hector SLAM. [Online]. Available: http://wiki.ros.org/hector_slam 16
- [36] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. 19, 80, 180, 182
- [37] G. Borgefors, “Distance transformations in digital images,” *Computer vision, graphics, and image processing*, vol. 34, no. 3, pp. 344–371, 1986. 19
- [38] C. Y. Lee, “An algorithm for path connections and its applications,” *Electronic Computers, IRE Transactions on*, no. 3, pp. 346–365, 1961. 19
- [39] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *et al.*, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 2. 19, 128
- [40] B. J. Julian, S. Karaman, and D. Rus, “On mutual information-based control of range sensing robots for mapping applications,” in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 5156–5163. 23
- [41] A. Visser, Xingrui-Ji, M. van Ittersum, L. González Jaime, and L. Stancu, “Beyond Frontier Exploration,” in *RoboCup 2007: Robot Soccer World Cup XI*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, vol. 5001, pp. 113–123. 23, 95
- [42] F. Amigoni, “Experimental evaluation of some exploration strategies for mobile robots,” in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, may 2008, pp. 2818–2823. 24, 25

REFERENCES

- [43] D. Joho, C. Stachniss, P. Pfaff, and W. Burgard, “Autonomous exploration for 3D map learning,” in *Autonome Mobile Systeme 2007*. Springer, 2007, pp. 22–28. [24](#), [25](#)
- [44] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, “Coordinated Multi-Robot Exploration,” *IEEE Transactions on Robotics*, pp. 376–386, 2005. [24](#), [25](#), [28](#), [186](#)
- [45] A. Visser and B. A. Slamet, “Balancing the information gain against the movement cost for multi-robot frontier exploration,” in *European Robotics Symposium 2008*. Springer, 2008, pp. 43–52. [25](#), [110](#), [148](#)
- [46] M. Kulich, J. Faigl, and L. Preucil, “On distance utility in the exploration task,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 4455–4460. [25](#), [28](#), [176](#)
- [47] A. Doniec, N. Bouraqadi, M. Defoort, V. T. Le, and S. Stinckwich, “Distributed Constraint Reasoning Applied to Multi-robot Exploration,” *Tools with Artificial Intelligence, IEEE International Conference on*, vol. 0, pp. 159–166, 2009. [25](#), [28](#)
- [48] X. Ma, Q. Zhang, and Y. Li, “Genetic Algorithm-based Multi-robot Cooperative Exploration,” in *Control and Automation, 2007. ICCA 2007. IEEE International Conference on*, 30 2007–june 1 2007, pp. 1018–1023. [25](#)
- [49] R. Rocha, F. Ferreira, and J. Dias, “Multi-robot complete exploration using hill climbing and topological recovery,” in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, 2008, pp. 1884–1889. [25](#)
- [50] V. Jisha and D. Ghose, “Frontier based goal seeking for robots in unknown environments,” *Journal of Intelligent & Robotic Systems*, vol. 67, no. 3-4, pp. 229–254, 2012. [25](#), [129](#), [130](#)
- [51] M. Juliá, A. Gil, L. Paya, and O. Reinoso, “Potential field based integrated exploration for multi-robot teams,” in *Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics*, 2008. [25](#), [29](#)

REFERENCES

- [52] A. Solanas and M. Garcia, “Coordinated multi-robot exploration through unsupervised clustering of unknown space,” in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, sept.-2 oct. 2004, pp. 717–721 vol.1. [26](#)
- [53] S. Thrun, “Learning metric-topological maps for indoor mobile robot navigation,” *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998. [27](#)
- [54] H. Choset and J. Burdick, “Sensor based planning. I. The generalized Voronoi graph,” in *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, vol. 2, may 1995, pp. 1649–1655 vol.2. [27](#)
- [55] E. Tsardoulas, A. Serafi, M. Panourgia, A. Papazoglou, and L. Petrou, “Construction of minimized topological graphs on occupancy grid maps based on gvd and sensor coverage information,” *Journal of Intelligent & Robotic Systems*, pp. 1–18, 2013. [27](#)
- [56] Z. Liu and G. von Wichert, “Extracting semantic indoor maps from occupancy grids,” *Robotics and Autonomous Systems*, vol. 62, no. 5, pp. 663–674, 2014. [27](#)
- [57] M. Berhault, H. Huang, P. Keskinocak, S. Koenig, W. Elmaghraby, P. Griffin, and A. Kleywegt, “Robot exploration with combinatorial auctions,” in *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 2, oct. 2003, pp. 1957–1962 vol.2. [28](#)
- [58] D. Puig, M. Garcia, and L. Wu, “A new global optimization strategy for coordinated multi-robot exploration: Development and comparative evaluation,” *Robotics and Autonomous Systems*, vol. 59, no. 9, pp. 635–653, 2011. [28](#), [148](#), [150](#)
- [59] M. N. Rooker and A. Birk, “Multi-robot exploration under the constraints of wireless networking,” *Control Engineering Practice*, vol. 15, no. 4, pp. 435–445, 2007. [28](#)
- [60] W. Sheng, Q. Yang, J. Tan, and N. Xi, “Distributed multi-robot coordination in area exploration,” *Robotics and Autonomous Systems*, vol. 54, no. 12, pp. 945–955, 2006. [28](#)

REFERENCES

- [61] J. D. Hoog, S. Cameron, and A. Visser, “Autonomous multi-robot exploration in communication-limited environments,” in *Proceedings of the 11th Conference Towards Autonomous Robotic Systems (Taros 2010)*, 2010, pp. 68–75. [28](#)
- [62] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: part i,” *Robotics Automation Magazine, IEEE*, vol. 13, no. 2, pp. 99–110, june 2006. [29](#)
- [63] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *Robotics, IEEE Transactions on*, vol. 23, no. 1, pp. 34–46, 2007. [29](#)
- [64] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, O. von Stryk, U. Klingauf, K. Petersen, A. Kleiner, O. von Stryk, S. Kohlbrecher *et al.*, “Hector open source modules for autonomous mapping and navigation with rescue robots,” in *Proceedings of 17th RoboCup international symposium*, 2013. [29](#)
- [65] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, “An evaluation of the rgb-d slam system,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1691–1696. [29](#)
- [66] M. Juliá, Ó. Reinoso, A. Gil, M. Ballesta, and L. Payá, “A hybrid solution to the multi-robot integrated exploration problem,” *Engineering Applications of Artificial Intelligence*, vol. 23, no. 4, pp. 473–486, 2010. [29](#)
- [67] L. Freda, F. Loiudice, and G. Oriolo, “A randomized method for integrated exploration,” in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 2457–2464. [29](#)
- [68] C. Stachniss, D. Hahnel, and W. Burgard, “Exploration with active loop-closing for FastSLAM,” in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 2, sept.-2 oct. 2004, pp. 1505–1510 vol.2. [29](#)
- [69] H. Choset, S. Walker, K. Eiamsa-Ard, and J. Burdick, “Sensor-Based Exploration: Incremental Construction of the Hierarchical Generalized Voronoi

REFERENCES

- Graph,” *The International Journal of Robotics Research*, vol. 19, no. 2, pp. 126–148, Feb. 2000. [29](#)
- [70] N. Kalra, D. Ferguson, and A. Stentz, “Incremental reconstruction of generalized Voronoi diagrams on grids,” *Robotics and Autonomous Systems*, vol. 57, no. 2, pp. 123–128, 2009. [29](#), [36](#)
- [71] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “Octomap: An efficient probabilistic 3d mapping framework based on octrees,” *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013. [29](#), [175](#)
- [72] B. Lau, C. Sprunk, and W. Burgard, “Efficient grid-based spatial representations for robot navigation in dynamic environments,” *Robotics and Autonomous Systems*, vol. 61, no. 10, pp. 1116–1130, 2013. [29](#)
- [73] 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, 2013. [31](#), [110](#)
- [74] S. Gottschalk, M. C. Lin, and D. Manocha, “OBBTree: a hierarchical structure for rapid interference detection,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH ’96. New York, NY, USA: ACM, 1996, pp. 171–180. [32](#), [164](#), [180](#)
- [75] J. Holm, K. De Lichtenberg, and M. Thorup, “Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity,” *Journal of the ACM (JACM)*, vol. 48, no. 4, pp. 723–760, 2001. [43](#)
- [76] Z. Galil and G. F. Italiano, “Data structures and algorithms for disjoint set union problems,” *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 319–344, 1991. [43](#)
- [77] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: an open-source Robot Operating System,” in *ICRA Workshop on Open Source Software*, 2009. [65](#)

REFERENCES

- [78] M. De Berg, M. Van Kreveld, M. Overmars, and Otfried Cheong Schwarzkopf, *Computational geometry*. Springer, 2000. 85, 88
- [79] H. Samet, “The Quadtree and Related Hierarchical Data Structures,” *ACM Computing Surveys*, vol. 16, no. 2, pp. 187–260, Jun. 1984. 85, 86
- [80] W. Pugh, “Skip Lists: A Probabilistic Alternative to Balanced Trees,” *Commun. ACM*, vol. 33, no. 6, pp. 668–676, Jun. 1990. 86, 87
- [81] J. L. Bentley, “Solutions to Klee’s rectangle problems,” Technical report, Carnegie-Mellon Univ., Pittsburgh, PA, Tech. Rep., 1977. 88
- [82] A. Howard and N. Roy, “The robotics data set repository (radish),” 2003. [Online]. Available: <http://radish.sourceforge.net/> 95
- [83] R. E. Korf, “Real-time heuristic search,” *Artificial intelligence*, vol. 42, no. 2, pp. 189–211, 1990. 128
- [84] S. Koenig, M. Likhachev, and D. Furcy, “Lifelong planning A,” *Artificial Intelligence*, vol. 155, no. 1, pp. 93–146, 2004. 128
- [85] A. Stentz, “Optimal and efficient path planning for partially-known environments,” in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. IEEE, 1994, pp. 3310–3317. 128
- [86] S. Koenig and M. Likhachev, “D* Lite.” in *AAAI/IAAI*, 2002, pp. 476–483. 128
- [87] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit *et al.*, “Fastslam: A factored solution to the simultaneous localization and mapping problem,” in *AAAI/IAAI*, 2002, pp. 593–598. 129
- [88] J. Sklansky, “Finding the convex hull of a simple polygon,” *Pattern Recognition Letters*, vol. 1, no. 2, pp. 79–83, 1982. 133
- [89] M. Nieuwenhuisen, D. Schulz, and S. Behnke, “Exploration Strategies for Building Compact Maps in Unbounded Environments,” in *Intelligent Robotics and Applications*, ser. Lecture Notes in Computer Science, S. Jeschke, H. Liu, and

REFERENCES

- D. Schilberg, Eds. Springer Berlin / Heidelberg, 2011, vol. 7101, pp. 33–43. [148](#), [149](#), [158](#)
- [90] L. Freda and G. Oriolo, “Frontier-Based Probabilistic Strategies for Sensor-Based Exploration,” in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, april 2005, pp. 3881–3887. [148](#)
- [91] Y. Girdhar, D. Whitney, and G. Dudek, “Curiosity based exploration for learning terrain models,” *arXiv preprint arXiv:1310.6767*, 2013. [148](#)
- [92] Y. Girdhar and G. Dudek, “Exploring underwater environments with curiosity,” in *Computer and Robot Vision (CRV), 2014 Canadian Conference on*. IEEE, 2014, pp. 104–110. [148](#)
- [93] P. Namal Senarathne, W. S. Wijesoma, K. W. Lee, B. Kalyan, M. Moratuwage, N. M. Patrikalakis, and F. S. Hover, “Marinesim: Robot simulation for marine environments,” in *OCEANS 2010 IEEE-Sydney*. IEEE, 2010, pp. 1–5. [162](#)
- [94] D. E. Koditschek, *Robot planning and control via potential functions*. Cambridge, MA, USA: MIT Press, 1989, pp. 349–367. [189](#)

Glossary

$A(\mathcal{Y}_j^{i,l})$	Set of safe grid cells represented by $\mathcal{Y}_j^{i,l}$
$A'(C)$	The set of grid cells in $\mathcal{S}_k$ associated with the connected component $C \in \mathcal{C}_k$
$\mathcal{B}_k$	The set of mapped-free-space-boundary cells
$\mathcal{BB}(U_k)$	Bounding box of the set U_k
$\mathcal{B}_k^{del}$	The set of cells that have lost boundary feature
$\mathcal{B}_k^{new}$	Newly generated boundary cells
$\mathcal{C}_k$	The set of connected components of G'_k
$C_{\mathcal{B},k}$	The set of boundary contours stored in the database
C_k^*	Connected component reachable by the robot
$C_{\mathcal{B},k}^{del}$	The set of contours extracted from $\mathcal{B}_k^{del}$
$C_{\mathcal{B},k}^{new}$	Newly extracted boundary contours
c_k^*	Robot's current position grid cell
E'_k	The set of edges of G'_k
E_k	Set of edges in graph G_k
$\text{eucl}(c_1, c_2)$	Euclidean distance between the cells c_1 and c_2
$\mathcal{F}_k$	The set of free cells

GLOSSARY

G'_k	Graph representation of the connectivity information of the safe cells $\mathcal{S}_k$
G_k	Safe-patch Graph
$\mathcal{J}_k$	The set of inflated obstacle cells
$\mathcal{M}_k$	The set of grid cells of the occupancy grid map after the k^{th} update step
$\mathcal{N}^\varphi$	Neighboring nodes of $\varphi \in V_k$
$\mathcal{N}_4(c)$	4-connected neighboring cells of cell c
$\mathcal{N}_8(c)$	8-connected neighboring cells of cell c
$\mathcal{O}_k$	The obstacle cell set
$\text{occ}_k(\mathbf{c})$	The occupancy class of grid cell. OBS for obstacle, FREE for free cell and UNKNOWN for unmapped
$\mathcal{O}_k^{del}$	Deleted obstacle cells
$\mathcal{O}_k^{new}$	Newly detected obstacle cells
$\mathcal{P}_k$	The set of grid cells representing the robot's trajectory up to step k
$\mathcal{Q}_k$	The set of all $\mathcal{Y}_j^{i,l}$
R_{max}	Maximum sensor range of the robot in grid cells
$\mathcal{S}_k$	The set of safe cells for robot
$\mathcal{T}_k$	Forrest of spanning trees of G_k
$\dot{\mathcal{T}}_k$	Forrest of trees after removing node set Υ_k^s from and adding node set $\hat{\Upsilon}_k$ to $\mathcal{T}_{k-1}$
$\ddot{\mathcal{T}}_k$	Forrest of trees after inserting the node set Υ'_k to $\dot{\mathcal{T}}_k$
U_k	The set of cells who's <i>occ</i> value is updated during step k
V'_k	The set of nodes of G'_k

V_k	Set of nodes in graph G_k
$\mathcal{V}(c)$	Maps safe grid cell c to a node in V_k
$\mathcal{V}'(c)$	Maps safe grid cell c to its associated node in G'_k
$\mathcal{X}_j^i$	Nodes added to G'_k at step j that belong to $C^i \in \mathcal{C}_k$
$Y(\varphi)$	Maps a node φ in V_k to an element in set Q_k
$\mathcal{Y}_j^{i,l}$	The l^{th} connected component of set $\mathcal{X}_j^i$
Γ_k	The set of frontier cells
$\Gamma_k(c)$	The set of neighboring frontier cells surrounding cell c
$\Lambda(\varphi)$	The adjacent nodes of φ that are neither child nor parent nodes of φ
ν	The root node of the spanning tree representing reachable connected component
Φ_k	The set of edges connecting new nodes in Υ'_k to existing trees
Υ'_k	The set of nodes added due to new safe cells
$\hat{\Upsilon}_k$	The set of nodes added due to splitting of nodes in Υ_k^s
Υ_k^s	The set of split nodes
Ξ_k	Partition of inflated obstacle boundary cells generated at step k according to membership of safe-patches represented by V_{k-1}
$\aleph_k$	Set of root nodes of sub-trees that got disconnected during safe-patch splitting