Distributive Cooperative 3D Exploration under Range Communication Constraints

by

Ravi Kulan Rathnam

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

Approved Dissertation Committee

Prof. Dr. Andreas Birk

Prof. Dr. Kaustubh Pathak

Prof. Dr.-Ing. Dieter Kraus

Date of Defense: September 3, 2014
Declaration

I, Ravi Kulan Rathnam, hereby declare that I have written this PhD thesis independently, unless where clearly stated otherwise. I have used only the sources, the data and the support that I have clearly mentioned. This PhD thesis has not been submitted for conferral of degree elsewhere.

I confirm that no rights of third parties will be infringed by the publication of this thesis.

Bremen, April 9, 2015

Signature

________________________________________________________________________________________________________
Abstract

Exploration can be termed as the process of autonomously navigating in an environment in order to build a map. This is one of the core components for autonomous robots. Mobile robots are increasingly being used in areas where human interaction is not present or extremely limited. These include realms such as surfaces of planets and moons, underwater sites, nuclear power plants etc. In such situations, robots need to navigate the environment without human intervention. They need to decide the next goal points to reach in order to increase their knowledge of the environment, using just the information collected so far.

With great advances being made in the field of single robot mapping and exploration, the next step is to use multiple robots. The obvious advantage is the speed up of exploration time by distributing the exploration task among multiple robots. Furthermore, the use of multiple robots ensures that there is no single point of failure, increasing redundancy, allowing for increased robustness. Also, multiple robots may help in localisation by detecting each other, or the same feature from multiple locations.

Most of the work done till now has focused on 2D exploration. This thesis proposes an exploration method which performs true 3D exploration, with robots navigating in a 3D environment to produce a 3D map. Furthermore, the members of the robot team stay within communication range of each other, thereby always being able to communicate, and hence being able to cooperate in the exploration process.

Also, multi-robot exploration has a very large search space. This means that it is not possible to arrive at an optimal solution in a reasonable amount of time. This requires sampling the search space to reach a solution. The random nature of the algorithm often leads to suboptimal solutions. In this thesis, the random sampling is augmented by a heuristic which improves the exploration time by preventing unnecessary steps during the exploration process.

The algorithm is tested in a high-fidelity simulator which was developed during the course of the project which takes into account the dynamics of the robot, thereby allowing a seamless transition to real life systems.
List of Publications

The publications which form this thesis are shown below:


Additional works published by the author, which though not directly used in this thesis, which nonetheless contributed to the authors understanding of simulation, system architecture, mapping and autonomous navigation include:


Contents

1 Introduction 17
  1.1 Problem Formulation and Terminology .......................... 18
  1.2 Structure of Thesis ........................................... 19

2 Related Work 21
  2.1 Single Robot Exploration ....................................... 21
    2.1.1 Topological and Graph Based Exploration ................. 22
  2.2 Multi-Robot Exploration ........................................ 23
    2.2.1 Market Based Approaches .................................. 25
    2.2.2 Topological and Graph Based Exploration ................. 26
  2.3 3D Exploration ................................................. 26
  2.4 Conclusion ........................................................ 27

3 The Co³-AUVs Simulator 29
  3.1 Introduction ....................................................... 29
  3.2 Comparison of Simulator Engines ................................. 30
    3.2.1 Unified System for Automation and Robot Simulation (US-
         ARSim) ....................................................... 30
    3.2.2 Object-oriented Graphics Rendering Engine (OGRE) ...... 31
    3.2.3 Bullet Physics Engine ....................................... 31
  3.3 Architecture ....................................................... 31
    3.3.1 Physics-World ................................................ 33
    3.3.2 Physics-World Listeners ..................................... 34
    3.3.3 Sensor-Worlds ............................................... 34
    3.3.4 Sensor-World Listeners ..................................... 36
    3.3.5 Debug Display ............................................... 36
    3.3.6 Networking Module .......................................... 36
  3.4 Benchmarking Co³-AUVs simulator against
    USARSim .......................................................... 37
  3.5 Simulated Scenarios ............................................... 37
  3.6 Conclusion ........................................................ 38
  3.7 Acknowledgements ............................................... 42

4 System Architecture 43
10 Conclusion

10.1 Summary of Contributions ............................................. 97
10.2 Future Work ............................................................... 98
List of Figures

3.1 Comparison of ray traces per second between different simulator engines. 32
3.2 The structure of the Co³-AUVs Simulator. 33
3.3 Real and simulated Jacobs Muddy-Waters I AUV. 34
3.4 Two screenshots of the simulator, top one showing a visualisation of one of the scenarios with a Muddy-Waters-I AUV, and bottom one showing several debug and sensor visualisation windows. 35
3.5 Comparison of ray tracing capabilities between USARSim and the Co³-AUVs Simulator with respect to number of robots. 38
3.6 Wall2D map used for experiments with 2D navigation and exploration. The red rectangle shows the starting location. 39
3.7 Top down view of the BigLake scenario. The red rectangle represents the starting location. 40
3.8 The scenarios based on Monta da Guia, Faial Island. 41
4.1 Architecture of the system. The exploration algorithm is integrated into a fully autonomous control system which consists of sensor acquisition, mapping, navigation, and control. 45
5.1 Robots with a frontier between them, but at spatial penalty range from each other. Robots cannot move towards each other as they are repelled by the spatial penalty. 51
5.2 Average time taken for different robot teams with a sensor range of 15 meters in the Wall2D map. 56
5.3 Average time taken for different robot teams with a sensor range of 25 meters in the Wall2D map. 56
5.4 A complete map of the environment generated at the end of a run in the Wall2D map. Black represents occupied cells, white represents free cells, and grey represents unknown cells. 57
6.1 Average exploration time for centralised exploration with respect to communication range in the BigLake scenario. 61
6.2 Average exploration time for centralised exploration with respect to sensor range in the BigLake scenario. 62
6.3 Median, upper and lower quartiles for exploration time with respect to sensor range for distributive exploration in the Azores-Crater scenario.

6.4 Median, upper and lower quartiles for exploration time with respect to communication range for distributive exploration in the Azores-Crater scenario.

6.5 Average number of deadlocks with respect to communication range in the Azores-Crater scenario.

6.6 Median time spent per deadlock in the Azores-Crater scenario.

7.1 Median, upper and lower quartiles for total exploration time with respect to communication range using RNEPosInfl in the Azores-Crater scenario.

7.2 Situation during exploration when robots have to travel long distances to reach the next frontier in the Azores-Crater Scenario. The white lines represent the communication links between the robots (represented by axes). The frontier cells are colour coded by height with blue being the uppermost, and red being the lowermost.

7.3 Median number of deadlocks with respect to communication range in the Azores-Crater scenario using RNEPosInfl.

7.4 Median time per deadlock recovery with respect to communication range in the Azores-Crater scenario using RNEPosInfl.

7.5 Experiment with one distant frontier in the Azores-Crater scenario. The experiments start with a completely explored map except for one frontier (blue cells) which is placed far away from the starting location of the robots (represented by axes).

7.6 The average time to finish exploration for the distant frontier experiment for RNEOrig and RNEPosInfl in the Azores-Crater scenario. As can be seen, the suggested improvements of RNEPosInfl speed up exploration.

8.1 Frontier cells at the edge of the field of view of the sensor. The grey area represents the unknown area, and the blue cells represent the frontier cells. P1 is the current pose of the robot. If the robot moves towards the frontiers to P2, the number of frontier cells in its field of view decreases. Therefore, the robot will stay in its current position.

8.2 Results from using the utility without ray tracing (RNEPosInfl) on the Azores-Cliff scenario.

8.3 Results from using the utility with ray tracing (RNESens) in the Azores-Cliff scenario.

8.4 Median, upper and lower quartiles for exploration time with respect to communication range using RNESens with sensors with 120° field of view in the Azores-Cliff scenario.
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5</td>
<td>Median, upper and lower quartiles for travel time with respect to communication range using RNESens with sensors with $120^\circ$ field of view in the Azores-Cliff scenario.</td>
</tr>
<tr>
<td>9.1</td>
<td>Summary of MORPH project.</td>
</tr>
<tr>
<td>9.2</td>
<td>Sensor (S) viewing a local plane. The area between R and L represents the area in the field of view of the sensor.</td>
</tr>
<tr>
<td>9.3</td>
<td>Sudden changes in the local orientation of the cliff can cause the robot to change orientation drastically. Pose 1 is the current pose of the robot. Pose 2 is the proposed position of the robot with the same orientation. Pose 3 is the proposed pose after changing orientation.</td>
</tr>
<tr>
<td>9.4</td>
<td>Median, upper and lower quartile of travelling time with respect to communication range with $60^\circ$ field of view sensor in the Azores-Cliff scenario.</td>
</tr>
<tr>
<td>9.5</td>
<td>Median, upper and lower quartile of travelling time with respect to communication range with $120^\circ$ field of view sensor in the Azores-Cliff scenario.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Robots have been widely used in industry for manufacturing for several decades. As computer components and sensors get smaller, mobile robots are increasingly being used. These mobile robots are gradually starting to operate in areas inaccessible or inhospitable to humans. These include realms such as surfaces of planets and moons, underwater sites, nuclear power plants etc. Due to this, there is an increasing need to build systems which can operate autonomously over extended periods of time. One major requirement for any long term deployment is the ability to generate maps. Even in situations where mapping is not the primary goal of the mission, robots need to create maps in order to perform other tasks. For example, even in the case of a robot surveying a nuclear power plant where the goal is to detect leaks or discover intruders, the robot needs to create a map in order to calculate paths in the environment.

Although some form of a priori maps can be provided in scenarios with previous human presence, this is not possible in scenarios where robots are deployed in previously unknown or highly altered environments. These include, among others, search and rescue, planetary exploration and underwater exploration. In many of these cases, the robots cannot be manually operated because of high communication delay, and dangerous conditions for operators. In these cases, the robots need to build the map autonomously. The process of autonomously navigating in an environment in order to build a map can be termed as exploration.

With great advances being made in the field of single robot mapping and exploration, the next step is to use multiple robots. The obvious advantage is the speed up of exploration time by distributing the exploration task among multiple robots. Furthermore, the use of multiple robots ensures that there is no single point of failure, increasing redundancy, allowing for increased robustness. Also, multiple robots may help in localisation by detecting each other, or the same feature from multiple locations.

However, the use of multiple robots also has a few drawbacks. Most importantly, the task of coordinating multiple robots efficiently is much more complex than controlling single robots. Care must be taken to avoid interference between robots, and that multiple robots do not explore the same area to avoid inefficiency. Also, there
needs to be coordination among the different robots. This is only possible if robots are able to communicate with each other in order to communicate the map, and decide the subsequent task assignments. Furthermore, communication feasibility is determined by various factors, such as distance to the robot, line of sight etc. The next section describes the precise problem statement for this thesis, and the specific terminology used.

### 1.1 Problem Formulation and Terminology

In this section, we define the terminology used throughout this thesis, and any assumptions made. First of all, it is assumed that each robot can localise itself perfectly in the environment. Unless stated otherwise, the robots use an omnidirectional sensor to produce a 3D occupancy grid of the environment. Initially, no prior information is known about the environment, and only information produced during exploration can be used. As the map is updated, each cell in the map is classified as one of the following during exploration:

- **Occupied** These are the cells in the environment that have obstacles which have been detected by the range sensor.
- **Free** These are the cells in the environment which have been seen by the range sensor of at least one of the robots and do not contain any obstacles.
- **Unknown** These are the cells about which no information is known because they have not been discovered by any of the range sensors. Initially all cells are marked as **Unknown**.
- **Frontier** Frontier cells are free cells which have at least one neighbouring unknown cell.

Furthermore, we define a strict communication limitation.

\[
\| p_i - p_j \| \leq R \quad \text{communication possible}
\]
\[
\| p_i - p_j \| > R \quad \text{no communication possible}
\]

where,

- \( p_i \) is the position of robot \( i \)
- \( p_j \) is the position of robot \( j \)
- \( R \) is the communication range.

Therefore, two robots can only communicate directly with each other when they are close enough to each other. However, two robots may communicate with each other through multi-hop communication by using multiple robots which are within communication range of each other. If we construct a communication graph of the robot team with each robot being a node in the graph, and an edge between two robots if they are within communication range of each other, then communication
is possible between any two members of the team if the graph has only one com-
ponent. The robot team can have a common map and the robots can be fully aware of each other’s positions only if multi-hop communication is possible between all robots. Therefore, an additional constraint that the robot team should not lose communicability during map building and exploration is placed. A communication constraint based purely on range is of course a simplification of the factors affecting the communication quality of the link. However, noise, multi-path effects, and various other sources influence the link quality as a function of distance in reality. Also, if it occurs that a link is not usable though it is still within the theoretical maximum range, the robots may return back to the last known good configuration, i.e. the previous set of poses where the network was still functioning, and resume from there.

Given all these constraints, the purpose of this thesis is to devise an algorithm to allow the robots to navigate the environment and reduce the amount of time spent to finish the exploration process. Exploration ends when there are no more unknown cells which can be reached by any of the robots.

1.2 Structure of Thesis

This thesis is organised as follows: Chapter 2 describes the general literature review for the exploration problem. Chapter 3 describes the simulator which was developed during the course of this project and is used to test the exploration system. Chapter 4 describes the architecture of the entire system. Chapter 5 describes how the algorithm developed by Rooker et al. [Rooker and Birk, 2007] was extended to be used in a continuous time scenario including distributing it across multiple robots. Chapter 6 extends the exploration algorithm to 3D. In Chapter 7, the algorithm is further improved by preventing inefficient motions. For Chapter 8 and Chapter 9, the sensor configuration is changed to allow for sensors with limited field of view. Chapter 9 adapts the exploration algorithm specifically to perform exploration in a cliff based scenario.
Chapter 2

Related Work

2.1 Single Robot Exploration

One of the earliest works on autonomous exploration to build a map was done by Yamauchi [Yamauchi, 1997] who introduced the concept of frontiers (boundary between free and unknown space). Frontier cells are grouped into frontier regions, and the robot travels to the nearest frontier region using a standard depth first search planning algorithm.

In Gonzalez-Banos et al. [González et al., 2000], a single robot computes the next-best view to construct a map as a 2D polygonal layout. Candidate points are sampled from the free space, and equally spaced rays are cast on the map, and the potential unknown area is calculated. This is used in a utility function which is also dependent on the path length to the goal point, and the goal point with the highest utility is used.

Grabowski et al. [Grabowski et al., 2003] try to get best viewing points for obstacles using an inverse sensor model. Multiple inverse sensor models are then combined to produce a region of interest map. A robot travelling to these areas can discern more information about the obstacles in the environment.

Sim et al. [Sim and Little, 2006] describe a system which produces a map using stereo imaging. In this method, the utility of the goal point is determined by performing ray traces from the goal point, and aggregating the utilities of the ray traced cells. The method aims to keep a set distance from unknown cells to aid in landmark based navigation. Upon reaching a goal point, the camera takes a 360 degree scan of the area to maximise coverage before computing the next goal.

Stachniss et al. [Stachniss et al., 2005] generate goal points on frontiers, and the information gain along the path to these frontier points are calculated by performing ray traces on the map. This information gain and the occupational probability of the cells in the path are used to calculate the expected utility, and the action with the highest utility is chosen.

Murali et al. [Murali and Birchfield, 2012] use a low resolution camera to construct a map of an indoor environment. The robot then explores a corridor by...
aligning itself to the center, and moves along till it reaches the end. At the end of the corridor, the robot turns towards the opening side of the corridor. However, the robot does not explore the rooms that appear on the side.

Freda et al. [Freda et al., 2008] present an exploration algorithm for a multi-bodied robot, equipped with multiple sensors. Frontier regions are found within sensor range, and then a search is performed in the configuration space to maximise the discovered area. Standard motion planning techniques such as Rapidly-Exploring Random Trees (RRTs) [Kufner and LaValle, 2001] are then used to plot a path in configuration space. When no more frontiers are visible from a particular configuration, the robot returns to its parent configuration, thereby effectively performing a depth-first search of the environment. The method is applied in 2D and 3D environments.

Calisi et al. [Calisi et al., 2007] combine standard frontier exploration with additional objectives such as victim or fire detection. Several methods to merge the goals from different modules are proposed, from weighted sums, to overwriting goals for higher priority modules.

Basilico et al. [Basilico and Amigoni, 2011] combine multiple scoring criteria for candidate goal points through the use of Pareto Frontiers. The different criteria are ordered and weights given according to a fuzzy measure which are then combined to find the “best” goal. The approach is then extended to multiple robots using the algorithm described in [Visser and Slamet, 2008].

2.1.1 Topological and Graph Based Exploration

Some exploration algorithms deal with performing exploration on topological graphs. The graphs are generated either by dropping markers in the environment, or by analysing the grid map using various methods to reduce it to a graph structure. Once the graphs are generated, these approaches take advantage of the simplified graph structure to solve the exploration problem.

Silver et al. [Silver et al., 2006] describe a method for exploring a subterranean mine. The robot constructs a topological map of the environment by constructing a Voronoi Graph of the occupancy map. Corridors are marked as edges and intersections are mapped as nodes. The exploration algorithm gives goals which make the robot traverse the Voronoi edges to reach the Voronoi nodes. The path planner then plans a path to avoid obstacles and reach the destination.

Kwon et al. [Kwon et al., 2006] produce a topological graph of an environment from an occupancy grid using a thinning algorithm. A position probability of the nodes is constructed, and the end nodes (nodes with only 1 edge) with low position probability close to the robot are chosen as next goal points. Exploration is said to be completed when all the end nodes have high position probabilities, i.e. when they are properly localised.

Batalin et al. [Batalin and Sukhatme, 2003] present a method in which the robot regularly drops markers in an area to form a graph. The robot is able to maintain communication with its closest marker allowing for localisation of the robot in the
The marker then describes the exploration state of its neighbouring edges, thereby allowing the robot to decide on its next possible move.

## 2.2 Multi-Robot Exploration

Several methods have been described to solve multi-robot exploration in two dimensions [Fox et al., 2006, de Hoog et al., 2009, Caiti et al., 2009, Sheng et al., 2006]. These methods to varying degrees take into account the multi-robot communication aspect, from only communicating if present [Fox et al., 2006], to best effort communication [de Hoog et al., 2009, Sheng et al., 2006], and to explicitly keeping all robots in communication range [Caiti et al., 2009, Rooker and Birk, 2007].

Concretely, Fox et al. [Fox et al., 2006] describe a method of mapping and exploration in which robots explore independently of each other, but coordinate when they are within communication range. When this happens, robots form small clusters with an assigned team leader which tries to find the optimal assignment of goals to the robots in the cluster. However, this assignment is not guaranteed to keep the robots in communication range.

Burgard et al. [Burgard et al., 2005] propose a centralised method to coordinate multiple robots by decreasing the utility of frontiers which are within the sensor range of a frontier assigned to another robot. The cost of a frontier cell is assigned to be the distance of the robot to the frontier, and utility is taken to be constant. Robots are assigned frontiers in order by picking the frontier with the highest utility. Utilities for the remaining frontiers are reduced based on the distance to the assigned frontier. In [Burgard et al., 2000], the utilities are reduced according to a probability distribution calculated from the histogram of sensor measurements, thereby adapting to the environment. Simmons et al. [Simmons et al., 2000] explicitly compute the sensor overlap by approximating the sensor field of view with a bounding box. This sensor overlap is used decrease the utility of frontiers which are close to each other.

Arkin and Diaz [Arkin and Diaz, 2002] present an algorithm for multi-robot exploration using line of sight communication constraints. In this method, robot take turns to move to newer areas till they can no longer “see” the other robots in the team. They then return to the area where the communication link is reestablished. In order to avoid continuous communication between the robots, only one robot is moving at any given time.

de Hoog et al. [de Hoog et al., 2009] assign roles to each robot in the team, either as explorers or relays. Explorers explore the environment and relays act as links between the explorers and the communication chain. Each explorer calculates the next goal for each explorer within the communication range. However, there is no emphasis on the robots staying within communication range. Communication requirements are managed by relays which try to keep the information available to each team member in sync. Hence, the communication chain is allowed to be broken.

In [Caiti et al., 2009], an area coverage method applied to underwater constraints
is proposed. The purpose of the coverage algorithm is to decrease the estimation error in the region. Robots are attracted towards areas of high uncertainty, but keep within a maximum distance from each other. Robots use fixed communication network topologies to communicate with each other, and the next goals of the team are calculated in a distributed fashion.

Lau et al. [Lau, 2003] describe an exploration algorithm using Social Potential Fields. The motion of the robot is determined by the net virtual force acting on it which attracts it to frontiers and repels it from obstacles and other robots. When the robots reach a local minima, they find the path to the nearest frontier and move towards it to get out of the local minima. However, this method does not take into account communication range for the team.

Wang et al. [Wang et al., 2011] divide the exploration area into predefined rectangular sub-areas with robots finishing a sub-area before going to the next sub-area. The robots perform standard frontier exploration in the sub-area by choosing the closest frontiers. After a robot finishes its sub-area, it chooses the next direction based on the closest frontiers and the presence of other robots and their sub-areas. However, communication is assumed to be global.

Wu and Zhang [Wu and Zhang, 2012] propose a method which attempts to find the minima of a scalar field by switching between individual exploration and periods of cooperation. The agents use noisy sensor data to assess the gradient of the scalar field. When the uncertainty increases considerably, the robots switch to a cooperative behaviour to reduce the uncertainty of the field. However, it is assumed that at this time, the robots can communicate to a central controller.

Puig et al. [Puig et al., 2011] attempt to reduce the variance in the exploration of unknown areas. This is useful in scenarios such as rescue robotics where it is desirable that regions are explored at similar speeds. This is done by dividing the unexplored area into regions according to the number of robots by K-means clustering. Each robot is then assigned to the optimal region using a Linear Programming method. Robots not close to their intended region are given priority in order to reach them as quickly as possible. The robots then explore the environment by avoiding obstacles and frontiers assigned to other robots.

Pei et al. [Pei et al., 2010] perform centralised exploration, but take into account the bandwidth of the robots. Frontier cells within a certain distance are chosen as goals, maximising the number of neighbouring unknown cells and hence maximising the number of reachable cells. The optimal relay positions are calculated ensuring that the maximum amount of information that needs to flow through a robot is not exceeded. Then robots are assigned to the respective goals using a linear programming method.

Kobayashi et al. [Kobayashi et al., 2003] use two robots to generate a belief measure map of the environment. The robots update the belief measure from the sensor information of other robots depending upon the distance to them. This allows for different belief maps, causing the robots to explore independently of each other.

Low et al. [Low et al., 2008] try to increase the certainty of the map by finding candidate observation locations. The task of environmental monitoring is performed
in a known environment by selecting paths between sampling “hotspots” which reduce the uncertainty in the observations.

Wu et al. [Wu et al., 2008] describe a multi-robot exploration algorithm based on the immune system. Unknown areas are likened to antigens and robots are compared to antibodies. The algorithm chooses the robots to explore a particular unknown space based on an immune model. The robots explore the area further keeping each other at a mid level distance between the sensor range and the maximum communication range.

Rooker et al. [Rooker and Birk, 2007] generate a set number of movements for each robot in the team. The combined movement for the team is then evaluated based on the distance of the robots to the frontiers and the communication availability of this “configuration”. The best configuration is then chosen as the desired set of goals.

Pal et al. [Pal et al., 2011] partition the unexplored cells using K-means clustering. The robots choose the best frontiers in their respective cluster based on the cost according to a modified A* algorithm which augments the A* cost with the robot movement direction, thereby favouring paths with fewer rotations. Also, since the number of frontiers can be quite large, calculating the A* cost to each frontier can be quite expensive. Therefore, the frontiers which have high estimated costs are removed from the set of possible frontiers.

Al Khawaldah and Livatino [Al khawaldah and Livatino, 2010] maintain line of sight between two robots while both robots explore the walls of an environment quickly. Once all the walls are explored, the two robots switch to frontier exploration using information based on the line of sight between the two robots.

### 2.2.1 Market Based Approaches

Market based approaches compare the problem of multi-robot exploration with a market economy. The goals are considered as resources and the robots bid for the tasks based on their costs for the robot. Generally, the robot that can perform the task with the least cost is allocated that particular task.

Zlot et al. [Zlot et al., 2002] use an auctioning system to perform multi-robot exploration. Robots construct a path from a set of goals. Before starting the next goal, the robot bids for the goal informing the other robots about the utility (number of unknown cells visible from the goal) and the cost (the time to travel to the goal). If another robot has a lower cost for the goal, the goal is handed to that robot. However, no emphasis is put on robots staying in communication range, and robots only communicate when possible.

In Sheng et al. [Sheng et al., 2006], each robot within a subnetwork bids for its next action. The robot with the best bid then executes its actions, and the auction is continued till each robot within the subnetwork is assigned its next goal. The robots are encouraged to stay close to each other by using a nearness measure, which decreases as robots get further away. However, if robots break away from the team, they do not attempt to return to the communication network.
Kumar et al. [Kumar et al., 2012] describe a method for area exploration based on flocking. In flocking techniques, the robots base their decisions only on neighbouring robots, and not all the robots in the team. The utility of a frontier is determined as a weighted sum of the distance to the frontier for the individual robot, the other robots in the team, and the number of unknown neighbours for the frontier. The robot with the highest bid is then assigned to the frontier.

2.2.2 Topological and Graph Based Exploration

As with single robot exploration, some multi-robot exploration algorithms simplify the environment to a graph structure and take advantage of the simplified graph structure.

Brass et al. [Brass et al., 2011] generate the graph by dropping RFID tags, and robots use the RFID information to prevent them from exploring the same area, making the exploration extremely efficient. However, there is no information being shared actively among the robots, and the result of exploration remains unknown till all robots return the starting location.

Hollinger et al. [Hollinger and Singh, 2012] divide a known environment into convex components which act as nodes in a graph. Each robot then plans paths for a set iterations ensuring that at the end of the path, the robots are within communication range of each other. However, the robots are allowed to break communication during the execution of the path. In the particular experiments presented, the task is not limited to exploration, but rather the task of finding a dynamic object in the graph.

Marjovi et al. [Marjovi et al., 2009] use a centralised map server to construct a topological graph of an environment. The cost of a frontier depends on the distance the robot has to travel to reach it, and the utility of a frontier is its distance from other robots. Hence, robots tend to chose frontiers away from other robots.

2.3 3D Exploration

Recently, there has been some interest in 3D exploration. Most of these methods describe the operation of a ground robot moving in a 2D environment but constructing a 3D map [Surmann et al., 2003, Fournier et al., 2007]. Even in cases in which robots are able to move freely in the 3D environment [Shade and Newman, 2011, Dornhege and Kleiner, 2011], the methods are not adopted towards efficient exploration of the environment with multiple robots.

Renzaglia et al. [Renzaglia et al., 2012] use Unmanned Aerial Vehicles (UAVs) to cover a terrain by optimising a combined objective functions that try to maximise the visible area and minimise the distance to the surveyed area.

Surmann et al. [Surmann et al., 2003] use a next best view algorithm to construct a map using multi-layer surfaces. The 3D map is represented as multiple 2D polygonal lines representing multiple levels in the map. To determine the next position,
the utility of the pose across the multiple levels is taken into account. However, the robots only move on a 2D-plane (the ground).

Shen et al. [Shen et al., 2012] use particles to represent free space. The density of free space particles in an area corresponds to the amount of exploration done in an area. They further propose an exploration algorithm using stochastic differential equations in which a single Micro UAV moves towards areas of low density, thereby moving towards unexplored areas of the environment.

Fournier et al. [Fournier et al., 2007] represent the environment as a 2.5D map, where a single ground robot moves towards frontiers whose utility is dependent on the distance between the frontier and obstacles, distance to the frontier and the robot, and the current robot direction.

Shade et al. [Shade and Newman, 2011] use a stereo camera to construct an octree to represent the 3D environment. Then a harmonic scalar function is used to construct a vector field whose gradient is perpendicular to the frontiers. This has the advantage that the robots approach the frontiers orthogonal to them, maximising visibility for a forward looking sensor.

Dornhege and Kleiner [Dornhege and Kleiner, 2011] use an octree to represent the 3D environment. Frontiers, i.e. borders between explored and unexplored space, and voids, i.e. confined spaces that are of interest for their application background of search and rescue, are identified. A next best view is calculated by ray tracing from all possible goal positions of a robot arm that carries the sensor, and the view with the best utility is chosen.

2.4 Conclusion

It can be seen that previous work on the subject of multi-robot exploration deals mainly with exploration in a two-dimensional area. Recently, there has been some work done in the field of exploration of a three dimensional environment. However, these have generally been limited to single land robots which are not able to navigate along all three dimensions. The purpose of this thesis is to deal with the problem of handling a complete team of robots for 3D exploration, which has not been tackled extensively.
Chapter 3

The Co$^3$-AUVs Simulator

3.1 Introduction

The use of simulations to aid robotic research has many potential benefits. Due to their wide availability and usability, many researchers use simulations to test their systems and algorithms, even without possessing the real hardware platforms. Some of the benefits of using simulators include but are not limited to:

1. Being able to test algorithms multiple times without having to worry about battery life
2. Being able to test systems without endangering robots, equipment or people
3. Being able to test different sensor configurations before putting the sensor on the robots

All these properties enable the reduction of costs, especially during the prototyping and testing phase. The advent of real-time simulator engines allows researchers to make a seamless transition between simulation and real world experiments.

This chapter describes the robotics simulator developed in the context of the “Cooperative Cognitive Control for Autonomous Underwater Vehicles (Co$^3$-AUVs)” project [Birk et al., 2011]. The general aim of the Co$^3$-AUVs project was to develop, implement and test advanced artificial cognitive systems for coordination and cooperative control of multiple AUVs.

One important topic in the project is 3D perception and mapping in the underwater domain. While 2D mapping is sufficient for open sea surface applications, there is a strong need for 3D models when it comes to more challenging - but increasingly important - application scenarios in harbours, marinas, at oil rigs, dams, i.e. at complex, typically man made structures as well as in challenging natural environments like reefs and cliffs.

For the purpose of exploration, it is necessary to have a simulator which can simulate the dynamics and behaviour of multiple AUVs operating in a complex environment in real time. The robots have to be able to construct a 3D map using
range sensors available to it. Therefore, it is necessary to have a simulator which can perform high speed ray traces in a simulated environment. High-fidelity simulation of 3D sonar is tremendously complex [Murino and Trucco, 2000, Lorenson and Kraus, 2009] and hence not suited for realistic environment scenarios that are desired for the Co3-AUVs Simulator. And even with simplified modelling, it is very important to achieve a high amount of ray tracing operations. This was hence an important design criterion for our simulator.

3.2 Comparison of Simulator Engines

As discussed above, for our purposes, the ray tracing performance of the simulator was very important to the design of the simulator.

In this section, the ray tracing performance of different engines is evaluated. For comparison purposes, a heightfield based on data from Chesapeake Bay, Maryland was used. The heightfield was then converted to a format recognized by each simulator engine. Then, a 3D range sensor was simulated, which performed ray traces to the terrain. The different simulator engines tested are described below. All tests were performed on Intel(R) Core(TM) i7 CPU 920 @ 2.67GHz, 3 GB RAM, under Ubuntu 9.10 32-bit (Bullet and OGRE) and Microsoft Windows Vista 32-bit (USARSim).

3.2.1 Unified System for Automation and Robot Simulation (USARSim)

USARSim is a simulation system using the Unreal 3 game engine [USARsim, 2006]. It is widely used as a robot simulator, especially in the RoboCup Rescue Virtual Competition [Carpin et al., 2007, Wang et al., 2005, Balakirsky et al., 2006, Carpin et al., 2006]. The game engine uses NVIDIA PhysX as the physics engine. The Unreal Engine is closed source, but Epic Games (the parent company) has released a scripting API which can be used to extend the system. However, the performance of programs using the scripting system is significantly worse than using the native API.

A grid of range sensors of size 55 x 55 was simulated. Each range sensor is set to perform a single ray trace along one direction. From past experience, it has been proven that this method yields faster results than using a single range sensor that performs 55 x 55 ray traces. This setup minimises the use of UnrealScript and allows the Unreal engine to schedule the work in its simulated threads. Each range sensor performed 5 ray traces per second. Hence, USARSim is able to perform more than 15,000 ray traces per second on this heightmap.

However, the Unreal engine can perform higher number of ray traces with larger number of sensors, each performing a smaller number of ray traces. This is because in the former case, all the sensors data needs to be synchronised in one location. However, by increasing the number of sensors, the total number of points where the
data is synchronised is increased, creating a less restrictive bottleneck. Using this method, USARSim is able to simulate up to 9 30x30 sensors performing at the rate of 5 Hz. Hence, USARSim is able to perform 40,000 ray traces per second when used in this way.

It has to be noted that computing ray trace performance in UnrealScript is a bit inaccurate because we are limited to the resolution of the time step of the game engine.

3.2.2 Object-oriented Graphics Rendering Engine (OGRE)

OGRE [OGRE, 2010] is an open source graphics rendering engine which can use either OpenGL (in Linux and Mac OS X), or Microsoft Direct3D (in Microsoft Windows). It has also widely been used in commercial titles such as TorchLight, and Venetica. Being purely a rendering engine, it does not perform any physics processing. Therefore, the physics calculations need to be performed by using other libraries. There are APIs which combine OGRE with a physics engine. These include OgreBullet [OgreBullet, 2010] (using Bullet physics library), and OgreODE [OgreODE, 2010] (using the Open Dynamics Engine).

OGRE does have basic ray casting capabilities, mainly for viewport-to-scene mouse trace object intersections. Thus, the basic ray casting on OGRE is only accurate to a bounding box level. More accurate ray casts require accessing the vertex information manually. However, ray casting to a terrain is accurate, with OGRE being able to perform 15,000 ray casts per second on terrains. Unfortunately, ray trace operation in OGRE is not reentrant and hence cannot be distributed across many threads.

3.2.3 Bullet Physics Engine

Bullet [Bullet, 2010] is an open source physics engine, used in several commercial applications including Grand Theft Auto 4 and LightWave 3D CORE. It is ranked as the third physics library by Game Developers Magazine (after PhysX and Havok Physics), but highest open source physics library. It provides features such as dynamics and collision detection. In addition to rigid body physics, it also has support for soft body dynamics.

Using Bullet, the ray traces are significantly faster than either OGRE or USARSim, being able to perform 70,000 ray traces per second. This performance was measured with a single threaded application. As will be seen, it is possible to perform significantly higher number of ray traces by using multiple threads.

3.3 Architecture

Due to the ray tracing performances described in the previous section, the Co³-AUVs Simulator uses Bullet as the physics engine, and OGRE as the rendering engine. The
Figure 3.1: Comparison of ray traces per second between different simulator engines.
The Physics-World is the core physics simulation module. This is implemented using solely the Bullet Physics Engine. It contains the world including static and dynamic elements (such as vehicles). Whenever any change occurs in the Physics-World, they are propagated to the outside using the callbacks on the registered listeners. These changes may be as a consequence of external modifications (such as adding objects...
to, or removing objects from the world), or movement (caused by the dynamics of the physics simulation), or physics interactions (such as actuators or collisions). Keeping the physics simulation separated from the other components ensures that the dynamics functions quickly, and is unaffected by other components, ensuring simulation quality.

### 3.3.2 Physics-World Listeners

As stated above, the changes in the Physics-World are propagated using Physics-World Listeners. Physics-World Listeners are those components which need to be informed about changes in the Physics-World. These changes include movements, addition of new objects, collisions etc. Physics-World Listeners include sensors, as well as any visualization tools that are needed to view the world. Physics-World Listeners can use the information provided by the Physics-World in a variety of ways. For example, a logger would store this information in a file, whereas a more involved visualisation tool would load different meshes for the objects and allow users to view the world. This approach can also be used to distribute the physics across other simulators by transferring the relevant information. These could be other simulators which use more advanced dynamical models specifically catering to a specific vehicle. Sensors are also implemented as Physics-World Listeners, which is described in the next section.

### 3.3.3 Sensor-Worlds

Sensor-Worlds are modules which contain the sensors. They are implemented as Physics-World Listeners as they require updates about the world. Sensor-Worlds may use this information in different ways. For example, Odometry Sensor-Worlds, need only information about the movement of robots and store the values. In contrast, Range Sensor-Worlds which need to perform ray traces, load objects into the
Figure 3.4: Two screenshots of the simulator, top one showing a visualisation of one of the scenarios with a Muddy-Waters-I AUV, and bottom one showing several debug and sensor visualisation windows.
world, and perform scans. The Sensor-Worlds ensure that the sensor updates are performed at regular intervals. Also, in order to achieve real time updates, it is possible to use multiple Sensor-Worlds of the same type. Implementing Sensor-Worlds as a separate module allows collecting sensor data without affecting the dynamics of the simulator. It is worth mentioning that due to our architecture, we have implemented configurable sensor schemes like odometry sensors, 2D and 3D range scanners and camera sensors.

3.3.4 Sensor-World Listeners

Sensor-World Listeners are analogous to Physics-World Listeners, but are meant for transferring sensor data. When sensor readings are updated, the data is propagated using the registered Sensor-World Listeners. These include any components that require sensor data, including networking components, as well as any visualization applications.

3.3.5 Debug Display

The Debug Display is the visualisation tool for the simulator. It is used to visualise the world, as well as visualising sensor data as it becomes available. For this purpose, it is implemented as both a Physics-World Listener and a Sensor-World Listener. The visualisation of the world currently uses the OGRE library. When the module receives updates from the Physics-World, it loads the required mesh (in case of objects being added), and moves the mesh in the world (in the case of movements), thereby allowing the visualisation of the scenario in real time. This functionality is shown in Figure 3.4. In addition to this, it is also used to view sensor data. When data is received from any of the sensors such as odometry, 2D and 3D range scanners, the corresponding sensor data is shown in the according widgets. This functionality is also shown in Figure 3.4.

3.3.6 Networking Module

The networking module performs two distinct tasks in the simulator. Firstly, it provides an interface to input commands into the simulator. This includes being able to spawn and unspawn robots in the simulator, as well as controlling already spawned robots. Secondly, it provides a method to allow users to collect sensor data. For this purpose, it is implemented as a Sensor-World Listener. Whenever new sensor data arrives at the networking module, it serialises the data and sends it across the network to any clients connected to it. The use of a network interface to communicate with the simulator allows for client software to be written in any programming language and running on a different machine using any operating system (not necessarily the one used by the simulator machine). Also, errors in the client software do not interfere with the simulation. Furthermore, the network interface allows the simulation to be run across multiple machines if required.
3.4 Benchmarking $Co^3$-AUVs simulator against USARSim

In addition to the ray tracing comparisons described above, the overall system was benchmarked against USARSim, which has been used in the context of marine robotics as basis for MarineSIM [Senarathne et al., 2010]. For this purpose, a range sensor performing 30 x 30 scans with a frequency of 5 Hz was created in both simulators. Then, vehicles were spawned with the sensor and moved in the world. The average time between the range sensor scans as number of robots (and hence number of sensors) increased was recorded.

The results of this experiment are seen in Figure 3.5. As can be seen, when the number of robots increases to more than 9, the time difference between scans in USARSim increases. At this point, the simulator is not able to schedule the sensors to scan at the required rate of 5 Hz.

In comparison, in the $Co^3$-AUVs Simulator, the time difference between scans is unvaried even with 24 robots added. This is made possible with the presence of 8 Range Sensor-Worlds which are each running in their own thread. In this way, the sensors perform scans without affecting each other or the dynamics of the system. As the number of sensors increase, additional Range Sensor-Worlds can be added to maintain the required frequency.

3.5 Simulated Scenarios

During the course of the thesis, several environments were constructed to test the exploration algorithm. Since the experiments simulate exploration done by AUVs, only the area which is under water (represented in blue in the corresponding figures) is covered by the robots. Below are the descriptions of the simulated scenarios:

1. **Wall2D**: This map was created by extruding a 2D pattern from a flat surface, and hence has the same structure at all depths. The robots explore an area of $40m \times 40m$. Hence, this map is used exclusively for experimenting with 2D exploration and navigation. This map is shown in Figure 3.6.

2. **BigLake**: This is an artificially generated scenario on a map of size $100m \times 60m \times 30m$. It represents a 3D underwater volume with multiple islands. A top down view of this map can be seen in Figure 3.7.

3. **Azores-Crater**: This data set is based on a real world scenario depicting Monte da Guia, Faial Island in the Azores. The real satellite image and the simulated map can be seen in Figures 3.8a and Figure 3.8b respectively. The central part of this area is used for exploration covering an area of $225m \times 175m \times 12.5m$. 

37
Figure 3.5: Comparison of ray tracing capabilities between USARSim and the Co$^3$-AUVs Simulator with respect to number of robots.

4. **Azores-Cliff**: This data set is also based on the Monte da Guia dataset. However, the area that is explored is the cliff on the eastern slopes of the island. The area to be explored for this scenario are shown in Figure 3.8c. This map covers an area of 140m $\times$ 150m $\times$ 14m.

<table>
<thead>
<tr>
<th>Scenario Name</th>
<th>Dimensions in meters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall2D</td>
<td>40 $\times$ 40</td>
</tr>
<tr>
<td>BigLake</td>
<td>100 $\times$ 60 $\times$ 30</td>
</tr>
<tr>
<td>Azores-Crater</td>
<td>225 $\times$ 175 $\times$ 12.5</td>
</tr>
<tr>
<td>Azores-Cliff</td>
<td>140 $\times$ 150 $\times$ 14</td>
</tr>
</tbody>
</table>

### 3.6 Conclusion

The chapter describes the implementation of a modular underwater simulator with several sensors, including a 3D range sensor. The simulator is implemented using the Bullet Physics Library as the physics engine and OGRE (Object-Oriented Graphics Rendering Engine) as the rendering engine. The simulator performs significantly
Figure 3.6: *Wall2D* map used for experiments with 2D navigation and exploration. The red rectangle shows the starting location.
Figure 3.7: Top down view of the BigLake scenario. The red rectangle represents the starting location.
Figure 3.8: The scenarios based on Monta da Guia, Faial Island.
better than commonly used robotics simulators with respect to ray tracing operations which are crucial for 3D perception and mapping experiments. A further benefit of the simulator is that it is completely based on open source components. It is able to calculate both the dynamics and the sensor data in real time. Due to its highly modular architecture, the different components can be changed easily, and be distributed across many machines. This makes it a very useful tool for researchers working in the area or underwater AUV research. This simulator is used throughout this thesis to test the proposed exploration algorithm.

3.7 Acknowledgements

I would like to thank all the people involved in the development of the simulator. I would like to thank Alexandru Ichim for his work on the Range Sensor-Worlds and initial work on the simulator, Max Pfingsthorn for his work on the development of the simulator scenarios, Dr. Kaustubh Pathak for developing the dynamic model for the vehicle.
Chapter 4

System Architecture

The exploration algorithm is integrated into a fully autonomous control system which consists of sensor acquisition, mapping, navigation, and control. The system is implemented using the Robot Operating System (ROS), which allows for a modular architecture. Since the focus of the thesis is on exploration, the other components are simplified. Due to the modular architecture present in ROS, the other components can be replaced with more refined variants allowing the exploration module to be integrated into a real system. The different modules are shown in Figure 4.1 and are described below:

**Exploration Module:** The Exploration Module performs the cooperative exploration presented. It uses the map provided by the Mapping Module to determine the utilities of the generated configurations.

**Mapping Module:** The Mapping Module assumes perfect localisation, and creates a 3D occupancy grid with a resolution of 0.5 meters. The robots can use the pose graph data structure for the map which allows the different mapping modules to transfer the map efficiently. Briefly, when a map update occurs, the robot sends a pose graph node with the location and sensor data associated with this node to the network. However, since the map being produced has a coarse resolution (0.5 meters), it is not necessary to transfer the entire scan, but only those beams which cause a change in the map. Furthermore, this data can be further compressed by standard compression algorithms. More detailed information about efficient distributed mapping with pose graphs including its use under communications constraints can be found in [Pfingsthorn and Birk, 2008]. This data can be used by the Mapping Module which assimilates them into the current map, using the standard Bresenham’s ray tracing algorithm in 3D.

**Frontier Exploration Module:** The Frontier Exploration Module uses the map produced by the Mapping Module to produce a cost map. Firstly, it grows the obstacles in the map to prevent the robot from getting too close to the obstacles. All frontier cells are found, and unified into contiguous frontiers.
All frontiers smaller than a certain size are removed. A wave propagation algorithm is then used to generate the cost map. The wave propagation does not pass through obstacles. Therefore, the value of a cell shows the shortest distance to the closest frontier.

**Global Planner:** When the Exploration Module generates goal configurations, the goals for each of the robots is transferred to the global planner of the robot. The Global Planner then finds the shortest path based on the map from the Mapping Module (with obstacles grown), to the goal using an A* search on the occupancy grid. The generated waypoints are then sent to the Local Planner.

**Local Planner:** The Local Planner guides the robot to the next goal point. It takes the target position from the Global Planner and gives corresponding motor commands to the robot, thereby reaching the required goal. It assumes the hydrodynamics of the Jacobs Muddy-Waters-I AUV (See Figure 3.3). As with the real AUV, the robot has 8 motors allowing it to perform all translations, but it cannot roll or pitch. To reach the next waypoint, the robot orients itself towards the goal on the yaw axis, and then performs the required translation.
Figure 4.1: Architecture of the system. The exploration algorithm is integrated into a fully autonomous control system which consists of sensor acquisition, mapping, navigation, and control.
Chapter 5

The Exploration Algorithm

5.1 Discrete Time Communicative Exploration

This thesis extends the Communicative Exploration algorithm described by Rooker et al. [Rooker and Birk, 2007]. The main goal of this algorithm is to extend frontier based exploration (see [Yamauchi, 1997]) by communication constraints. Concretely, the algorithm keeps each of the robots within the maximum communication range of at least one other robot in the team. Through the use of multi-hop communication, it is hence guaranteed that there is always a fully connected network between the robots. This fully connected network can then be used to acquire a common map, and to coordinate the robot team.

At each step, given the common map and the current poses of the robots in the team, the algorithm tries to calculate the next set of movements for all the robots. The number of possible moves grows exponentially to the number of robots in the team. Therefore, it is necessary to use a heuristic solution which tries to solve the problem by best effort.

Calculation of the Utility

The most important step in the Communicative Exploration algorithm presented by Rooker et al. [Rooker and Birk, 2007] is the calculation of the utility. The exploration always aims to maximise the utility at each step. The utility of the configuration, i.e. the set of poses of the robots in the team, is dependent on the following factors.

1. **Loss of Communication**: If the configuration causes the robots to no longer maintain communication with each other, the utility is set to $-\infty$.

2. **The Distance to Frontiers**: The most important factor in the utility of the configuration is the distance to the frontier. When the frontiers are calculated using a standard frontier extraction algorithm, a Manhattan distance map of the shortest path to the closest frontier is generated. The map is then used to
calculate the Manhattan Utility $U_{MD}(p)$ for a position $p$.

$$u_{MD}(p) = \begin{cases} 
-\infty & \text{if } p \text{ lies inside an obstacle or in unknown space} \\
-md(p) & \text{otherwise}
\end{cases}$$

where $md(p)$ is the Manhattan distance of the shortest path to the frontiers. The total Manhattan Utility $U_{MD}$ is then calculated as the sum of the individual $u_{MD}(p)$.

$$U_{MD}(C) = \sum_{i=1}^{N} u_{MD}(p_i)$$

where $C = \{p_1, p_2, \ldots, p_N\}$

$p_i$ is the position of robot $i$. Therefore, the total utility $U_{CE}$ for a configuration $C$ is calculated as

$$U_{CE}(C) = \begin{cases} 
-\infty & \text{if loss of communication} \\
U_{MD}(C) & \text{otherwise}
\end{cases}$$

Communicative Exploration Algorithm

The communicative algorithm is shown in Algorithm 1. At each iteration, the position of all the robots are put in the current configuration $C$. Then a set of $\kappa$ unique configuration changes are generated. A configuration change for a system of $N$ robots is defined as follows:

$$cfg = \{m_1, m_2, \ldots, m_N\}$$

where $m_i$ is a possible movement of the robot $i$. The following movements are possible

$$m \in \{NW, N, NE, W, R, E, SW, S, SE\}$$

where NW stands for Northwest, N for North, S for South, E for East, W for West, R for Remain etc. Once the configuration changes are generated, they are added to the current configuration to give the set of possible future configurations. Then the utility is calculated as described above, and the configuration with the best utility is chosen. As mentioned earlier, the number of possible configurations increases exponentially with the number of robots and therefore it is not possible to find the optimal solution with a limited amount of time. However, the likelihood of finding the optimal solution, or at least a very good solution, increases with the
increase in the number of samples. This is an important feature of any probabilistic approach.

Algorithm 1: One iteration of the Communicative Exploration Algorithm

```plaintext
Generate the configurations
i = 1
cfgs = {}
while i <= \kappa do
  Generate a configuration change for each robot
  for j = 1 to N do
    random select d \in \{N, NE, E, SE, S, SW, W, NW, R\}
    cfg[j] = d
  end
  if cfg \notin cfgs then
    cfgs[i] = cfg
    i +=
  end
end
Find the configuration with the maximum utility
bestcfg = cfgs[1]
bestUtil = get utility of cfg[1]
for i = 1 to \kappa do
  util = get utility of cfg[i]
  if util > bestUtil then
    bestUtil = util
    bestcfg = cfg[i]
  end
end
give goal points to each robot based on bestcfg
```

Deadlock Recovery

Since the algorithm always tries to find the best configuration in the neighbourhood of the current configuration, it acts similar to a gradient ascent algorithm. Similar to gradient ascent, the robot team can get stuck in local maxima. This generally occurs when the robots are exploring different frontiers and are at the limit of their communication range, and any further movement towards the frontier would cause a loss in communication. When no progress has been made by any of the robots, a deadlock is said to have occurred. This situation is detected, and a simple deadlock recovery procedure is initiated. One of the robots is randomly selected as a rendezvous point, and the team tries to converge at this point. During deadlock recovery, since no new areas are discovered and all goal points are known, the robots
are allowed to go out of communication range for short periods of time. Once the deadlock has been recovered, the robots start the exploration process again. At this point in time, the robots have travelled a fair distance away from their previous goal frontiers, thereby ensuring that they are closer to different frontiers. Also, the robots are closer to each other, allowing them free motion to their new closest frontiers without going out of communication range of each other.

5.2 Modification to the Utility Function

Several modifications need to be made in order to apply the communicative exploration algorithm to realistic scenarios [Rathnam and Birk, 2011]. In the algorithm presented above, the robots are modelled as point robots, and hence there is nothing to prevent the robots from colliding with each other. It is possible to solve this by using an obstacle avoidance module to the robots. However, this module might be working counter to the goals of the exploration module. Therefore, it is better to take into account the closeness of the robots in the exploration module. In order to prevent robots coming too close to each other, a spatial penalty is added to the utility. This also prevents the robots from going towards the same frontier.

The pairwise spatial penalty between two robots $i$ and $j$, $SP_{i,j}$ is defined as

$$SP_{i,j} = \begin{cases} P_{\text{max}}(1 - \frac{\text{dist}(i,j)}{D_{\text{Spatial}}}) & \text{if } \text{dist}(i,j) \leq D_{\text{Spatial}} \\ 0 & \text{otherwise} \end{cases}$$

where $P_{\text{max}}$ is the maximum spatial penalty to apply, $\text{dist}(i,j)$ is the distance between robots $i$ and $j$, $D_{\text{Spatial}}$ is the distance till which to apply the spatial penalty.

The total spatial penalty $SP$ is then defined as

$$SP = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} SP_{i,j}$$

where $N$ is the number of robots.

Therefore, the total utility $U_{RNE}$ (Random Neighbourhood Exploration) for a configuration $C$ is calculated as

$$U_{RNE}(C) = \begin{cases} -\infty & \text{if loss of communication} \\ U_{MD}(C) - SP(C) & \text{otherwise} \end{cases}$$

In order to ensure that the robot team does not get stuck in a local maximum as a result of the Spatial Penalty, the spatial penalty should only be applied for distances at which robots sense the same area. If this is not the case, the $U_{MD}$ causes the robots to move towards each other, whereas $SP$ causes the robots to move away from each other. If this occurs at a distance where the goal frontier is not seen
Figure 5.1: Robots with a frontier between them, but at spatial penalty range from each other. Robots cannot move towards each other as they are repelled by the spatial penalty.

by either robot, the robots are stuck at a local maximum with no configuration available which improves upon the current utility. Such a situation is shown in Figure 5.1. In the figure, if the robots move towards the frontier, the spatial penalty starts to take effect causing the utility to decrease. If the robots move away from each other to decrease the spatial penalty, they move away from the frontier causing the utility to again decrease, thereby causing a local maximum at this configuration. Therefore, $D_{Spatial}$ has to be set to less than $2 \times SensorRange$ in order to prevent such a situation from occurring.

5.3 Implementation into a Continuous Time System

A major issue in implementing the Communicative Exploration algorithm in a robotic system is that the real motions of the robots take time, and the robot must take steps to avoid obstacles not present on the constructed map. It is hence necessary to extend the abstract algorithm from [Rooker and Birk, 2007] with the according means to handle the dynamics of a real multi-robot system. Therefore, the exploration module is implemented here as a state machine. These states are
required for both the centralized and the distributed version of the algorithm when it is to be used on real robots with non-trivial dynamics.

The following states are introduced:

1. *CalculateNextChange*: In this state, the exploration module tries to calculate the next actions of the robot team. Once the best subsequent action is calculated, the individual goals are sent to the navigation modules of each robot. Then, the module goes into the *MovingTowardsGoal* state. Also, in this state, the module checks whether the robot team is now deadlocked. In case of a deadlock, the module goes into the Deadlock state.

2. *MovingTowardsGoal*: This state is reached when the goal points have been set on the robots, and the robots are travelling towards their respective goals. In this state, the module checks whether all robots have reached their intended goal. Once the goals are reached, the module goes into the *CalculateNextChange* state.

3. *Deadlock*: This state is reached when the robot team is deadlocked, i.e. the greedy heuristic led to a local optimum. The robot then attempts to recover from the deadlock by randomly selecting a robot to be the rendezvous point. This goal point is then sent to the navigation modules of the robots, and the robots go into the *DeadlockRecovery* state.

4. *DeadlockRecovery*: In this state, the module waits until the robots reach their deadlock recovery goal. The goal is said to be reached when the robot is within a set distance from it. Once all robots recover from the deadlock, the module goes into the *CalculateNextChange* state.

### 5.4 Distributive Exploration

#### 5.4.1 Motivation

As we increase the number of robots in the system, the number of possible configuration changes increases exponentially. The combinatorial explosion in the number of possible configuration changes can at least be mitigated by the linearly increasing amount of available computation power for the distributed version while the centralized version has to cope with a very limited fixed amount of computation power of a single robot.

Furthermore, there is the crucial benefit of increased robustness. This is of particular interest for underwater multi-robot-teams, which tend to be rather small in number and where the possible performance gains for large scale teams are hence not so important. Underwater robotics is challenging and very error-prone for a variety of reasons including the harshness of the environment and the limitation in available sensor technology. It is hence quite common that a vehicle has to abort
its mission or that it is even completely lost. Increased mission robustness through a team of robots is hence desirable but it can only be achieved if it does not have the Achilles heel of a central coordinator.

The increased robustness and the performance gain in the exploration are bought at the cost of an increased amount of data that is transmitted for the purpose of distributive exploration algorithm. In the case of the centralized version of the module, only the configuration change for this robot needs to be transmitted to each robot at each iteration, whereas in the distributive case, the entire configuration change needs to be transmitted.

5.4.2 Extension to Distributive Exploration

Due to the increased computational power available in a team of robots, the generation of configurations can be distributed across the multiple robots. Therefore, the distributed modules together can generate linearly increasing number of configurations with respect to the number of robots. Once this is done, the robots can bid on the next configuration change with each other.

Therefore, for the distributed version of the exploration module, the states are as follows:

1. **CalculateNextChange**: This state performs the same actions as in the centralized version. However, after calculating the best configuration change, this configuration change is not sent to the navigation modules, but the best change and the utility is broadcast to all other exploration modules. Then, the module goes into the WaitingForConfiguration state.

2. **WaitingForConfiguration**: In this state, the exploration module waits for the results of the best change from other explorers. Once it receives the response from all other explorers, it chooses the best configuration change based on the utility sent. In case of a tie (configurations with the same utility), the configuration change sent by the robot with the lowest ID is chosen. The intended goal point is then sent to each robot. Also, it is possible that a robot receives more than one ConfigurationChange message from another robot. Therefore, in this state, the robot only checks the message number that it is responding to. After this, the module goes into the MovingTowardsGoal state.

3. **Deadlock**: As in the CalculateNextChange state, the goal is not sent to the navigation modules, but to the exploration module of the other explorers. The module then goes into the WaitingForDeadlock state. Also, another condition for deadlock detection is added. It is possible that one of the robots in the team detects a deadlock before other members of the team. In this situation, the other robots may be waiting for a configuration change message from this robot. Therefore, if a robot receives a deadlock message from any of the other robots, the robot goes into the Deadlock state.
4. *WaitingForDeadlock*: In this state, the module waits for the deadlock goals from all other explorers. The robot goal sent by the robot with the lowest ID is chosen and sent to each robot. After this, the module goes into the *DeadlockRecovery* state, which is the same as for the centralized version.

## 5.5 Message Format

The Configuration change message has the following fields:

- **origin**: This is the id of the robot from which the message was sent. This should be maximum 1 byte long. This field is required for the distributive version of the algorithm in order to break ties.

- **MessageNumber**: This can be seen as the id of the message. When a robot receives multiple messages from a robot, then it only takes into account the message with the correct message number for this iteration. The message number is not unique to the entire run, and can be repeating. It only needs to be unique to the possible messages received in one iteration. Therefore, this field only takes up one byte.

- **Utility**: This is the utility of the configuration change proposed as a float.

- **ConfigurationChange**: This is the actual configuration change which has the best utility. This field is 1 byte per robot in the team.

Therefore, the total size of the message for a team of $N$ robots is $N + 6$ bytes.

For the centralised algorithm, the exploration module has to send the direction for each robot, except for the robot on which the exploration module is running.

## 5.6 Results

The aforementioned algorithm was tested on the *Wall2D* map (see Figure 3.6). Figure 5.2 and Figure 5.3 show the average amount of time (over 10 runs) it takes to completely explore the environment with a team of robots, once with each sonar having a range of 15 meters and once with sonars with a range of 25 meters. Figure 5.4 shows a final map once the environment is completely explored.

As can be seen, the algorithm typically performs slightly better in terms of exploration time in the distributed version than in the centralized version. However, the performance difference is not very big for relatively small numbers of robots in the team. The reason for this becomes clear when we look at the performance difference as we increase the number of samples in the centralized version. As can be seen, the performance does not increase significantly as we increase the number of samples used for the heuristic optimization. However, the advantage of this fact
is that we can generate a smaller number of samples for comparable results, freeing up the robot processor for other possible tasks.

The effect becomes even more apparent when the smallest possible team-size, i.e. two robots is considered. This is, so to say, the worst case for the distributed version because the total number of possible combinations of configuration changes is only 81. It is hence quite feasible to check all of them in a single robot, hence a centralized version profits from the lack of the overhead compared to the distributed one.

Also, the amount of data that needs to be transferred for the purpose of exploration is extremely small. There are 9 possible directions, which can be coded in 4 bits, the minimum number of data bytes that needs to be transmitted per iteration is \[
\lceil \frac{(N - 1)}{2} \rceil\text{ bytes for } N\text{ robots.}
\]

In the case of distributed exploration, the exploration module on each robots needs to send its best configuration change candidate to every other exploration module at each iteration. Therefore, the total data bytes being received per iteration by an explorer is \((N-1) \cdot (N+6)\) bytes. In a system with 5 robots, the total amount of data being transferred would be 44 bytes. This is almost negligible compared to the amount of data required to update the map with a cooperative pose-graph SLAM, which requires the map updates and the estimated spatial transformations for each robot. Given a 90 degrees field of view and 0.5 degree resolution for the sonar, the amount of data required is \(((180 \cdot 2 + 24)) \cdot N\) for \(N\) robots, namely 2 bytes per range and 9 floats for representing the poses in the pose-graph edge and the related uncertainties in each pose-graph edge. This corresponds to 1920 bytes for 5 robots.

However, it should be noted that in the 2D case, since there are only 9 possible directions, it is possible to transfer the motion commands for 2 robots in 1 byte, thereby decreasing the amount of data being transferred by a half.

5.7 Conclusion

In this chapter, a distributed version of the communicative exploration algorithm, which extends the frontier-based exploration algorithm with a strict obedience of communication constraints is presented. The major advantage of using a distributed approach in contrast to a centralized version is that the robot team is more robust and more resistant to failure, which is especially a significant advantage for underwater robotics missions. Furthermore, the distributed version can benefit from increased available computation power in larger teams leading to more efficient execution.

In addition to the extension in form of a distributed version, the communicative exploration algorithm is in turn embedded here in a complete robot system with proper mapping and motion control. This required an extension of the abstract algorithm in the form of a state machine to handle the underlying dynamics of the vehicles. The system is tested in a high-fidelity marine robotics simulator.
Figure 5.2: Average time taken for different robot teams with a sensor range of 15 meters in the Wall2D map.

Figure 5.3: Average time taken for different robot teams with a sensor range of 25 meters in the Wall2D map.
Figure 5.4: A complete map of the environment generated at the end of a run in the *Wall2D* map. Black represents occupied cells, white represents free cells, and grey represents unknown cells.
Chapter 6

Extension to 3D

6.1 Motivation

2D exploration has been widely studied due to its usefulness in most man made environments. Even buildings with multiple floors can be seen as multiple 2D exploration problems. While the inherent 3D nature of unstructured environments is taken into account by land robots, the fact that land robots can only navigate the ground makes them not wholly capable in performing true 3D exploration. In underwater scenarios, natural environments, and also important man-made environments such as harbours and ports have 3D structures. Also, submersible robots or vehicles are required and able to navigate in all three dimensions in order to accurately study the environment, i.e. teams must be able to dive and coordinate across all depths. This gives rise to the problem of true 3D exploration, i.e. what steps do the robots have to take in order to fully explore the 3D structure or environment.

6.2 Extension to 3D

The different components of the system can be readily adapted to the three dimensional nature of the problem by replacing the 2D grid structures with 3D grids.

The Communicative Exploration algorithm is extended into 3D by modifying the generation of the configurations [Rathnam and Birk, 2012]. Instead of the 9 possible directions on a plane, each robot can move into the 27 adjacent cells on the grid. Therefore, the configuration change is now defined as

$$cfg = \{m_1, m_2, \ldots m_N\}$$

where $m_i$ is a possible movement of the robot $i$. The following movements are possible

$$m \in \{U, R, D\} \times \{N, R, S\} \times \{E, R, W\}$$
where U is up (+z), D is down (−z), N is north (+x), S is south (−x), E is east (+y), W is west (−y), R is remain, which corresponds to no change on the axis. In the case of 3D exploration, since there are 27 possible directions, they can be coded in 5 bits. It is possible to fit the directions for 3 robots into 2 bytes, saving some more bandwidth, thereby decreasing the amount of data being transferred by a third. Therefore, the total amount of data being transferred for the configuration change is $\lceil \frac{2}{3}(N - 1) \rceil$ bytes.

### 6.3 Results for Centralised 3D Exploration

The following experiments were conducted in the *BigLake* scenario (Figure 3.7), which is of size 100m×60m×30m. The robots are started in a central location in the environment and the effects of sensor range with an omnidirectional sensor and communication range on the exploration algorithm are observed. The effect of sensor range is investigated by alternating between discrete values of 15, 20, 25 and 30 meters. The effect of communication range is analysed by alternating the maximum communication range between 20, 40, 60 and 750 meters. Since the world has a maximum dimension of 100m×60m×30m, at the communication range of 750 meters, the robots can be considered to always be in communication range of each other. Each experiment was repeated 10 times and results from multiple runs are averaged.

As can be seen from Figure 6.2, the average time taken decreases with increase in sensor range. In the case of three robots with a communication range of 40 meters, the average time goes from 925 seconds at a sensor range of 15 meters, to 616 seconds for a sensor range of 30 meters. This is expected as the robots can sense a greater part of the environment from their position, and hence need to navigate less to observe the entire environment. It is interesting to note that the performance does not increase continuously, but rather tapers off. The main reason for this is that at higher ranges, the area that the robot can clear is dependent on the obstacles blocking the sensor. Therefore, an increase in the sensor range does not affect the performance greatly after this point.

Also from Figure 6.1, we can see that the performance of the algorithm decreases by restricting the communication range. In the case of 2 robots with a sensor range of 20 meters, the average exploration time changes from 650 seconds at a communication range of 60 meters, to almost 1000 seconds at a communication range of 20 meters. The reason for this is that at lower communication ranges, the robots spend more time deadlocked and recovering from deadlocks. In this situation, the robots explore for a smaller proportion of the time. Also, after the deadlock recovery, it is possible that the utility function guides some of the robots towards the same frontier. In this situation, the robots end up exploring the same area, thereby increasing the time they need to explore the entire environment. Furthermore, in this case, the advantage of having more robots is lost. Also, with large communication ranges, the robots tend to explore different parts of the environment, thereby decreasing...
the total overlap, maximising the performance of the algorithm. As with the change in sensor range, the performance of the algorithm does not always increase with increase in communication range. The reason for this is that given the size of the environment, large communication ranges acts similar to an infinite communication range. The robot team is always able to find a configuration better than the current one without getting deadlocked. Therefore, the communication range only restricts the exploration slightly, and hence no time is wasted being deadlocked.

Furthermore, it should be noted that the total amount of data transferred for the purposes of exploration is very small. The configuration change message consists of the ID of the robot, the configuration change and the utility. Therefore, the configuration change message is $N + 3$ bytes long. This is almost negligible compared to the amount of data required to update the map with a cooperative pose-graph SLAM, which requires the map updates and the estimated spatial transformations for each robot.

![Graph showing average exploration time for centralised exploration with respect to communication range in the BigLake scenario.](image)

Figure 6.1: Average exploration time for centralised exploration with respect to communication range in the *BigLake* scenario.
Figure 6.2: Average exploration time for centralised exploration with respect to sensor range in the BigLake scenario.
6.4 Results for Distributive Exploration

The performance of distributive 3D exploration [Rathnam and Birk, 2013a] was tested in the Azores-Crater scenario which simulates Monte da Guia, Faial Island in the Azores (see Figure 3.8b). The environment has dimensions $225m \times 175m \times 12.5m$ and the robots are started in a corner of the map. The effects of communication and sensor range on the exploration algorithm are investigated. The sensor range was increased from 10 meters to 20 meters at a communication range of 40 meters. The communication range was increased from 20m to 80m while keeping the maximum sensor range at 10m. Experiments with each set of parameters were run 10 times and the results are aggregated.

6.4.1 Effect of Sensor Range on Distributive 3D Exploration

As expected, the performance of the exploration increases with increase in sensor range. For example, in the case of 1 robot, the median exploration time decreases from 1970 seconds with a sensor range of 10 meters to 1050 seconds in the case of 20 meters. In the case of 6 robots, the median exploration time decreases from 1100 seconds to 450 seconds for the same change. This is expected, because with...
higher sensor ranges, robots need to navigate less of the environment in order to explore the same volume as robots with lower sensor ranges. It is seen that the effect of increasing the sensor range is greater for a higher number of robots. This is because, the higher number of robots are able to take full advantage of the increased sensor range by exploring the entire environment without going into deadlocks. In the experiments conducted, at a sensor range of 20 meters, no deadlocks occurred in any of the runs in the case of 5 or 6 robots, with only one deadlock occurring in the case of 4 robots.

6.4.2 Effect of Communication Range on Distributive 3D Exploration

As expected, in general, the median exploration time decreases with increase in communication range. In the case of 3 robots, the median exploration time decreases from 1250 seconds at a communication range of 20 meters to 1170 seconds at a communication range of 80 meters. This same trend is seen for robot teams of different sizes. However, it is interesting to see that the improvement in performance is not continuous. As the communication range is increased from 20 meters, the performance initially decreases. In the case of 6 robots, the median exploration time...
Figure 6.5: Average number of deadlocks with respect to communication range in the Azores-Crater scenario
time increases from 1130 seconds at a communication range of 20 meters, to 1230 seconds at a communication range of 30 meters. The same is seen in the case of 3 robots. In this case, the median exploration time increases from 1250 seconds for a communication range of 20 meters to almost 1490 seconds in the case of 40 meters. This is because these communication ranges are still too small compared to the environment, thereby not decreasing the number of deadlocks that occur. In the case of 2 robots, the average number of deadlocks remains stable at close to 5 deadlocks per run for these communication ranges (see Figure 6.5). However, since the robots have to travel greater distances to recover from the deadlocks, the total exploration time does not decrease. In this case, the average amount of time spent during deadlock recovery increases from 45 seconds for a communication range of 20 meters, to almost 100 seconds for a communication range of 40 meters. In the case of 6 robots at a communication range of 20 meters, the robots spend 33 seconds per deadlock on average. However, this increases to 76 seconds in the case of a communication range of 30 meters, i.e. the average time spent per deadlock more than doubles in both these cases (see Figure 6.6).

6.4.3 Effect of Number of Robots on Distributive 3D Exploration

As can be seen from Figure 6.3, the median exploration time decreases with increase in number of robots at all sensor ranges. At a sensor range of 10 meters, the median exploration time drops from 1970 seconds in the case of 1 robot to 1100 seconds in the case of 6 robots. At a sensor range of 20 meters, the median exploration time drops from 1000 seconds when 1 robot being used, to 450 seconds when 6 robots are used. As the number of robots increase, the robots are able to explore areas independently of each other. This allows for different areas of the map to be explored simultaneously, allowing for faster exploration.

It is interesting to see that the performance does not increase continuously with the increase in the number of robots, but rather tapers off as more robots are added. For example, with a sensor range of 15 meters, increasing from 1 robot to 2 robots, decreases the median exploration time by close to 30 percent. Subsequently, increasing the number of robots to 3, only causes a gain of 200 seconds, or 20 percent. Adding a further 3 robots, only decreases the median exploration time by only 200 seconds. The same effect can be seen when we look at a higher sensor range of 20 meters. An initial increase in the number of robots by 1 causes a great increase in performance with the median exploration time dropping from over 1000 seconds to 650 seconds. A subsequent increase of 4 more robots only decreases the median exploration time to 450 seconds, a decrease of 30 percent for four additional robots. The reason for this is that even though the robots are able to explore independently of each other, it is possible that robots which are close to each other explore the same frontiers. Also, it may occur that one of the robots is behind another robot with respect to the frontiers. As a result, this robot explores less than the robot in
Figure 6.6: Median time spent per deadlock in the Azores-Crater scenario.
front of it, causing a decrease in the benefit of the additional robot.

Another disadvantage of increasing the number of robots is that the robot team takes longer to recover from deadlocks when they occur. Deadlocks occur when robots travelling to their closest frontiers break the communication chain. This generally happens in a communication network where each robot is connected to only one other robot forming a long chain. In this situation, when one of the robots is chosen as a rendezvous point, robots travel proportional to their distance in the chain to the desired robot, resulting in some robots travelling far greater distances than others to recover from the deadlock. This is clearly seen when the median times for deadlock recovery are compared (Figure 6.6). As can be seen, median time per deadlock increases both with increase in number of robots and increase in communication range. For example, at a communication range of 30 meters, the median time per deadlock recovery increases from 16 seconds in the case 2 robots to 76 seconds in the case of 6 robots. Also, when we look at the case of 3 robots, the median time for deadlock recovery increases from 21 seconds at a communication range of 20 meters to 78 seconds at a communication range of 80 meters. Despite the increase in time per deadlock recovery, the performance of the exploration increases with increasing number of robots. This is because the increase in deadlock recovery time is offset by the fact that, as the number of robots increase, the robot team is able to form larger chains and has more flexibility of movement, resulting in a decrease in the number of deadlocks. Therefore, despite the increase in complexity as a result of increasing the size of the robot team, the algorithm is able to compensate by utilising the extra robots considerably.

6.5 Conclusion

In this chapter, the cooperative exploration algorithm is extended to 3D. A detailed study of the effect of sensor range, communication range, and number of robots is performed. It is seen that the additional robots invariably reduce the total exploration time in all experiments. This is definitely a requirement for multi-robot exploration, despite the increase in complexity. It is also seen that in general, increase in communication range and sensor range decreases exploration time. Increasing communication range allows robots more freedom to navigate allowing them to explore the environment much faster. Also, the extension to 3D allows for more realistic scenarios with regard to AUV navigation.

However, the approach has a few drawbacks which need to be addressed. It is seen that the amount of time robots have to spend during deadlock recovery increases greatly with respect to communication range and number of robots. This is looked into in the next chapter.
Chapter 7

Improvements to the Communicative Exploration Algorithm

7.1 Description

As described in the previous chapter, when the number of robots is increased, the number of possible configurations increases exponentially. However, the number of generated configurations only increases linearly. Also, the chances of finding the optimal configuration decreases considerably. It also becomes increasingly unlikely that a configuration change is generated in which all robots move closer to the frontiers. However, the robot team moves to a new configuration when it is better than the current configuration. This can happen if more than half the robots move towards frontiers, while the other robots move away from frontiers. This results in individual robots making unnecessary movements. Since the team only moves when all robots have reached their new positions, this increases the total time that the exploration takes.

Therefore, to prevent robots from moving away from frontiers, an additional check is done in each exploration module [Rathnam and Birk, 2013b]. After the best configuration is chosen by an exploration module, for each robot, it is checked whether the movement has a positive influence on the utility function. If not, this robot is ordered to remain stationary. After this test, the algorithm functions as previously, sending the best calculated configuration to the other exploration modules. This part of the algorithm is described in Algorithm 2.

As explained previously, as the communication range increases, the distances that the robots need to travel to recover from deadlocks also increases significantly, since the robots have to reach within a certain distance of the goal robot. Also, when the robots come close to each other, the team tends to travel towards the same frontier, thereby decreasing the use of having multiple robots. To address this problem, the algorithm is further changed such that a robot is said to recover
Algorithm 2: Exploration with Positive Influence Test

\[
\begin{align*}
\text{bestChange} &= \text{get Best Change from Communicative Exploration} \\
\text{bestUtil} &= \text{getUtility(bestChange)} \\
\text{for } i = 1 \text{ to numberOfRobots do} \\
&\quad \quad \text{nextChange} = \text{bestChange} \\
&\quad \quad \text{nextChange}[i] = \text{NoMove} \\
&\quad \quad \text{util} = \text{getUtility(nextChange)} \\
&\quad \quad \text{if } \text{util} > \text{bestUtil} \text{ then} \\
&\quad \quad \quad \text{bestChange} = \text{nextChange} \\
&\quad \quad \quad \text{bestUtil} = \text{util} \\
\text{end}
\end{align*}
\]

from a deadlock when it reaches within a percentage of the communication range (in this case 50 percent) of the recovered robot team. This ensures that the robots have to travel a shorter distance to recover from deadlocks. Also, robots do not get too close to the same frontiers, allowing more independent exploration. From this point, the exploration algorithm with the aforementioned changes will be referred to as RNEPosInfl and the original algorithm as RNEOrig.

7.2 Results

The following experiments were conducted on a data set based on the Azores-Crater scenario (see Figure 3.8b) which depicts Monte da Guia, Faial Island in the Azores. The central part of this area is used for exploration covering a span of 225m \times 175m \times 12.5m. The robots are started in a corner of the map at the beginning of the exploration process. Each experiment is repeated 10 times and the results are aggregated.

Figure 6.4 shows the total exploration time without the improvements (RNE-Orig) as the number of robots and communication ranges are changed. Figure 7.1 shows the total exploration time with the proposed changes (RNEPosInfl). As can be seen, the exploration algorithm consistently performs better with the improvements.

As can be seen in both cases, the median exploration time decreases with increase in number of robots at all communication ranges. At a communication range of 20 meters, the median exploration time decreases from close to 2000 seconds in the case of 1 robot, to approximately 1060 seconds in the case of 6 robots. At a communication range of 30 meters, the median exploration time for RNEPosInfl drops from 1435 seconds in the case of two robots, to 1000 seconds in the case of 6 robots. For RNEOrig, the performance increase is slightly less, with the median exploration time dropping to 1230 in the case of 6 robots. At higher communication ranges, the drop is even higher. At a communication range of 80 meters with 6 robots, the
Figure 7.1: Median, upper and lower quartiles for total exploration time with respect to communication range using RNEPosInfl in the Azores-Crater scenario.

Figure 7.2: Situation during exploration when robots have to travel long distances to reach the next frontier in the Azores-Crater Scenario. The white lines represent the communication links between the robots (represented by axes). The frontier cells are colour coded by height with blue being the uppermost, and red being the lowermost.
median exploration time drops even further to close to 760 seconds in the case of RNEOrig, and 600 seconds for RNEPosInfl. As the number of robots increase, the robots are able to explore areas independent of each other. This allows for different areas of the map to be explored simultaneously. It can be seen that in general, the increase in performance is greater at higher communication ranges. This can be attributed to the fact that at higher communication ranges, less deadlocks occur and robots have more freedom to explore the environment. Furthermore, at low communication ranges, the number of deadlocks is extremely high. In the case of 2 robots at 20 meters, more than 8 deadlocks occur on average before the robots are able to finish exploration (see Figure 7.3). As mentioned, during deadlock recovery, the robot team travels through known areas, and hence very little exploration happens during this phase. Also, after deadlock recovery, the robots are positioned somewhat closer to each other. Hence, it is possible that a few robots may be positioned behind other robots with respect to frontiers. This means that these robots will not explore any areas not explored by other robots, thereby decreasing the benefit of adding additional robots. Additionally, as a result of deadlock recovery, the robots leave some frontiers to explore subsequently. This results in fractionalisation of the frontiers, requiring the robots to travel large distances to reach these frontiers.

![Number of Deadlocks with respect to Communication Range](image)

Figure 7.3: Median number of deadlocks with respect to communication range in the Azores-Crater scenario using RNEPosInfl.
Figure 7.4: Median time per deadlock recovery with respect to communication range in the *Azores-Crater* scenario using *RNEPosInfl*. 
As mentioned previously, RNEOrig suffers from a few drawbacks. Namely, the time spent recovering from deadlocks increases considerably with respect to number of robots and communication range. Also, the time taken to reach distant frontiers is also very high due to the fact that the robots make a few unnecessary movements. Some of these issues are addressed by RNEPosInfl, as can be seen in Figure 7.4. In the case of 2 robots, the median amount of time spent on deadlock changes from 63 seconds to only 86 seconds, an increase of only 33 percent as opposed to 100 percent for RNEOrig. The same effect is seen with 6 robots. The median time per deadlock increases from 31 seconds in the case of a communication range of 20 meters, to 45 seconds in the case of a 40 meter communication range, a change of only 45 percent. This is achieved because the distance that the robots have to travel to recover from deadlocks is significantly lower than in RNEOrig. Furthermore, when the robot team has to travel large distances to reach the next frontier, the greedy approach performs significantly better than RNEOrig. This is seen by the fact that the median exploration time does not increase as much in the case of RNEPosInfl, as opposed to RNEOrig.

The performance of the exploration increases as the communication range increases further. This can especially be seen at higher communication ranges of 60 meters and 80 meters, the median exploration time decreases considerably. Using RNEPosInfl for 6 robots, the median exploration time drops to 608 seconds at a communication range of 80 meters from 883 seconds at a communication range of 40 meters. The same effect is seen in the case of 5 robots. As the communication range increases from 40 meters to 60 meters and then 80 meters, the median exploration time decreases from 852 seconds to 810 and 641 seconds respectively. At these communication ranges, the robots are able to explore frontiers unrestrictedly, thereby clearing areas of the map without going out of communication range of each other. This reduces the number of deadlocks with no deadlocks occurring for 5 and 6 robots and only 1 deadlock occurring the case of 3 robots. In these situations, RNEPosInfl does not receive the advantages from faster deadlock recovery. Nonetheless, it performs better than RNEOrig due to the fact that it prevents unwanted movements. This becomes extremely important in the end stages of exploration when robots have to travel large distances to reach the next frontiers. Such a situation is shown in Figure 7.2. In RNEOrig, robots tend to make many oscillating steps while reaching the frontiers, whereas in RNEPosInfl, robots always move towards the frontiers.

7.2.1 Experiment with only One Distant Frontier

The fact that RNEPosInfl reduces the time required to reach distant frontiers is further confirmed by an additional experiment conducted on the same map with robots needing to go to a single frontier far away (see Figure 7.5). In this experiment, the initial map consists of only explored cells, except for one frontier which is placed a considerable distance from the starting position of the robots. Hence, all robots need to move along the same direction in order to get close to the frontier and finish the exploration process. Figure 7.6 shows the average time taken by both RNEOrig
and $RNEPosInf$ in this scenario. As can be seen, as we increase the number of robots, the average time taken by $RNEOrig$ to reach the frontier increases steadily from 100 seconds in the case of 2 robots to more than 140 seconds in the case of 6 robots, an increase of 40 percent. Since the number of possible configurations increases exponentially with the number of robots, the chances of selecting the correct configuration decreases considerably. However, in the case of $RNEPosInf$, the average time needed to reach the frontier stays constant at 100 seconds. In this case, the greedy approach always chooses the configuration with all robots moving towards the frontier, providing no decrease in performance with increasing number of robots.

### 7.3 Conclusion

In this chapter, the distributive cooperative 3D exploration algorithm is improved upon it by preventing undesirable movements. This reduces the exploration time by allowing robots to only move in the desired direction, preventing oscillations. A detailed study of the effect of communication range and number of robots is performed. It is seen that the additional robots invariably reduce the total exploration time, in all experiments. Also, the drawbacks of the previous methods causing increased exploration time with increase in communication range despite the additional freedom of navigation is corrected. As a result of these improvements, the performance of the exploration is increased with increased communication range.
Figure 7.5: Experiment with one distant frontier in the *Azores-Crater* scenario. The experiments start with a completely explored map except for one frontier (blue cells) which is placed far away from the starting location of the robots (represented by axes).

Figure 7.6: The average time to finish exploration for the distant frontier experiment for *RNEOrig* and *RNEPosInfl* in the *Azores-Crater* scenario. As can be seen, the suggested improvements of *RNEPosInfl* speed up exploration.
Chapter 8

Cooperative 3D Exploration with Limited Field of View

8.1 Introduction

The previous work presented so far assumes an omnidirectional sensor. In such cases, the orientation of the robot is unimportant as the robot would clear the same volume in all poses. However, in most real world scenarios, robots have a limited field of view. Therefore, it is necessary to consider the field of view of the robot during exploration. It is not enough to use the distance to the frontier as the only measure for the utility. Such an approach does not take into account the fact that a higher number of cells are visible from a position further away than a position closer to the frontier. Therefore the utility calculation is modified to include the properties of the sensor, including sensor range, field of view, angular resolution.

This “Sensor” Utility $U_S$ is calculated using the following method:

\[
F(p) = \text{Set of frontier cells visible from pose } p
\]

\[
F(p_1, ..., p_n) = \bigcup_{1}^{n} F(p_i)
\]

\[
U_S(c) = |F(c)|
\]

where

$c = \{p_1, p_2, ..., p_n\}$ are the full 6 DoF poses consisting of position and orientation of each of the robots in the robot team.

Therefore, the Sensor Utility $U_S$ is the total number of frontier cells visible from a particular configuration. However, a robot may reach a pose from which the frontiers are further away and cannot be reached in one step. In this situation all the neighbouring poses have the same utility (namely 0) causing a local maximum. To avoid this, we must take into account the distance to the frontiers as well. Therefore, the utility of a configuration is defined as
\[ U(c) = k \cdot U_S + U_{RNE}(c) \]

where

- \( U_S \) is the Sensor Utility of the pose
- \( k \) is the weight factor to be applied to the Sensor Utility
- \( U_{RNE} \) is the utility of the original algorithm which is dependant on the sum of the Manhattan Distances of the robot team to the frontiers and returns \(-\infty\) for disallowed configurations.

This ensures that robots get closer to frontiers when no frontiers are visible from the generated robot poses. However, the value of the scaling factor \( k \) must be chosen to avoid local maxima. It is possible, that the robot may have to move away from the closest frontier in order to view more frontier cells. Hence, the two components of the utility act in opposite directions. For two robot poses \( p_1 \) and \( p_2 \).

\[
\begin{align*}
U(p_1) &= k \cdot U_S(p_1) + U_{RNE}(p_1) \\
U(p_2) &= k \cdot U_S(p_2) + U_{RNE}(p_2) \\
\Rightarrow \quad U(p_2) - U(p_1) &= k \cdot (U_S(p_2) - U_S(p_1)) + (U_{RNE}(p_2) - U_{RNE}(p_1))
\end{align*}
\]

Consider the situation where the number of frontier cells visible from \( p_2 \) is more than the number of cells visible from \( p_1 \). In this situation, it would be preferred that the total utility of \( p_2 \) is more than the total utility of \( p_1 \).

\[
\begin{align*}
U(p_2) &> U(p_1) \\
\Rightarrow \quad U(p_2) - U(p_1) &> 0 \\
\Rightarrow \quad k \cdot (U_S(p_2) - U_S(p_1)) + (U_{RNE}(p_2) - U_{RNE}(p_1)) &> 0 \\
\Rightarrow \quad k &> \frac{U_{RNE}(p_2) - U_{RNE}(p_1)}{U_S(p_2) - U_S(p_1)}
\end{align*}
\]

The maximum value for the above fraction is achieved by maximising the numerator and minimising the denominator. The minimum possible difference between number of visible frontiers is 1. Since, we are only looking at neighbouring positions of the current robot position, the maximum possible Manhattan distance between two cells being compared is 6. This is the maximum Manhattan distance between two configurations generated during one step of the exploration algorithm. Therefore, a scaling factor of \( k > 6 \) prevents local maxima. Prevention of local maxima is necessary because the Deadlock Recovery step assumes that a deadlock occurs because of communication limits, which is only true if the utility function does not have any other local maxima. In this chapter, the method using this utility will be referred as \( RNESens \), and the previous method as \( RNEPosInfl \).

Furthermore, in order to ensure that the robot moves towards newer areas, we have to redefine \textit{Frontier Cells}. Earlier, \textit{Frontier Cells} were defined as free cells
which have at least one neighbouring cell which is unknown. However, with this definition, when a frontier cell comes into view of the sensor, its status remains unchanged as the cell is already known. Consider the situation in Figure 8.1. In this situation, the frontier cells are present in the boundary of the sensor’s field of view at position $P_1$. Since the frontier cells are cells whose values are known, no new map areas are being discovered and the frontiers do not change. The utility at position $P_2$ is less than $P_1$ since the number of frontier cells which can be viewed is less than from $P_1$. Therefore, there is a possibility of the robot getting stuck in a local maximum even though there is more area to be explored. To prevent this, we redefine Frontier Cells as unknown cells which have a neighbouring free cell. In this way, when a frontier cell comes into view of a robot sensor, its status will be changed from unknown to known as a result of the map update, thereby decreasing the utility of the current pose $P_1$ and the exploration can progress towards new areas.

![Figure 8.1: Frontier cells at the edge of the field of view of the sensor. The grey area represents the unknown area, and the blue cells represent the frontier cells. $P_1$ is the current pose of the robot. If the robot moves towards the frontiers to $P_2$, the number of frontier cells in its field of view decreases. Therefore, the robot will stay in its current position.](image)

8.2 Initial Results without Changing Orientation

For the purpose of these experiments, the Azores-Cliff scenario was used (Figure 3.8c), which represents the eastern slopes of Monta da Guia Volcano in the Azores. This map covers an area of $140\text{m} \times 150\text{m} \times 14\text{m}$ and the robots start at a corner of the environment. A front looking sensor with a $120^\circ$ horizontal and vertical field of...
view with a sensor range of 10 meters was simulated. As in previous experiments, there is no change in orientation once the scenario begins. Each experiment was repeated 10 times and the results aggregated.

Figure 8.2 shows the effect of the number of robots on the total exploration time and the travel time for \textit{RNEPosInfl}. The travel time is the time during which at least one robot is travelling to its goal. Figure 8.3 shows the results using \textit{RNESens}. As can be seen, the amount of time needed to explore the environment is drastically reduced from the previous method.

Using \textit{RNEPosInfl}, the median exploration time decreases from 2700 seconds in the case of one robot, to 990 seconds in the case of 6 robots. However, in the case of \textit{RNESens}, the median exploration time drops from 1300 seconds in the case of 1 robot, to 680 seconds in the case of 6 robots. This is close to half the exploration time of \textit{RNEPosInfl}. In \textit{RNEPosInfl}, the robots try to get close to their nearest frontier. This method works in cases where the process of getting close to the unknown frontier leads to the discovery of the frontier. However, with sensors with limited field of view, frontiers only get discovered when they come within the field of view of the sensor, i.e. only when a robot approaches from a particular direction. In cases where the robot approaches the frontier cell from a different direction, the frontier cell only gets “discovered” when the robot arrives on the cell. This implies that the robots have to travel great distances as they have to come close to the frontiers, thereby increasing the amount of time needed to explore the entire area.

In \textit{RNESens}, a configuration is chosen depending on how many frontier cells are visible from the pose of the robots. As a result, frontiers are approached from a position which maximise the number of visible frontier cells. This is despite the fact that the rotation is not changed according the position of the frontiers.

Also, it should be taken into account that the amount of time needed to calculate the utility in \textit{RNESens} is significantly longer than the amount of time needed to calculate the utility in \textit{RNEPosInfl}. This results in robots having to wait for other robots to calculate their best configuration, adding overhead to the total exploration time. Therefore, the decrease in exploration time is very significant. The amount of time needed to calculate the utilities can be reduced by parallelising the process, and by using more efficient map representations. This can be further seen in the difference in total amount of time spent travelling by the robots in the two methods. In \textit{RNEPosInfl}, the median travel time decreases from 2300 seconds in the case of one robot to 850 seconds in the case of 6 robots, a decrease by a factor of approximately 2.7. In comparison, in \textit{RNESens}, the median travel time decreases from 1030 seconds in the case of 1 robot, to 287 seconds in the case of 6 robots, a decrease by a factor of more than 3.5. Furthermore, comparing the median travelling time between the two methods, in the case of one robot, the median travelling time in \textit{RNESens} is half that of \textit{RNEPosInfl}. This difference continuously increases with \textit{RNESens} taking 2.5 times less than \textit{RNEPosInfl} in the case of 3 robots, to taking almost a third of the time in the case of 6 robots.
(a) Time taken with respect to number of robots in the Azores-Cliff scenario. The robots are equipped with sensors with $120^\circ$ horizontal and vertical field of view and range of 10 meters.

(b) Total time spent travelling with respect to number of robots in the Azores-Cliff scenario. The robots are equipped with sensors with $120^\circ$ horizontal and vertical field of view and range of 10 meters.

Figure 8.2: Results from using the utility without ray tracing (\textit{RNEPosInfl}) on the Azores-Cliff scenario.
(a) Median, upper and lower quartile of time taken with respect to number of robots for robots exploring the *Azores-Cliff* scenario. The robots are equipped with sensors with 120° horizontal and vertical field of view and range of 10 meters.

(b) Median, upper, and lower quartile of total time spent travelling with respect to number of robots for the *Azores-Cliff* scenario. The robots are equipped with sensors with 120° horizontal and vertical field of view and range of 10 meters.

Figure 8.3: Results from using the utility with ray tracing (*RNEsens*) in the *Azores-Cliff* scenario.
8.3 Cooperative Exploration with Sensor Utility and Communication Range

8.3.1 Adding Rotation to Exploration Goals

After establishing that RNESens performs significantly better than RNEPosInfl, we further modify the method by allowing the robot to rotate when approaching a frontier in the direction from which the frontier is not visible. When no frontier is visible from the original rotation, i.e. the utility from $U_S$ is 0, the robot turns in the direction of motion towards the next goal.

8.4 Results

Using the modifications described above, the effect of communication range was investigated in the same scenario.

![Exploration Time vs Communication Range](image)

Figure 8.4: Median, upper and lower quartiles for exploration time with respect to communication range using RNESens with sensors with 120° field of view in the Azores-Cliff scenario.

Figure 8.4 and Figure 8.5 show the effect of communication range on the total exploration time and travel time. As can be seen, the median time needed for exploration drops as the number of robots is increased at all communication ranges. At a communication range of 15 meters, the median exploration time drops from 1200 seconds in the case of 1 robot to 935 seconds in the case of 6 robots. The same effect is seen in the case of an infinite communication range, where the median
Figure 8.5: Median, upper and lower quartiles for travel time with respect to communication range using RNESens with sensors with 120° field of view in the Azores-Cliff scenario.
exploration time drops to 720 seconds in the case of 4 robots and further to 687 seconds in the case of 6 robots.

This is more apparent when looking at the total time the robots travels. In the case of communication range of 15 meters, the median travel time drops from 985 seconds in the case of 1 robot to 500 seconds in the case of 4 robots and down to 430 seconds in the case of 6 robots. In the case of a communication range of 30 meters, we see the median travel time drop to 450 seconds in the case of 4 robot and further to 300 seconds in the case of 6 robots. This trend shows the added robots are able to explore areas not visited by other robots, thereby aiding the exploration process.

However, it is interesting to see that we see only a small, though consistent, improvement when looking across the communication range. As can be seen, in the case of 3 robots, the median travel time decreases from 600 seconds at a communication range of 15 meters to 410 seconds in the case of an infinite communication range. This is an improvement of 10 percent. The improvement is much higher with using a higher number of robots. In the case of 6 robots, the robots spend 460 seconds reaching their destinations at a communication range of 15 meters, to spending 280 seconds in reaching their goals at an infinite communication range. This is an improvement of close to 35 percent. Due to the limited field of view of the robots, the frontiers are formed much closer to each other. The robots approach these frontiers from different orientations, almost moving around the environment in formation. Due to this, the robots do not have to travel too far to reach new frontiers and do not reach the edge of the communication range even for extremely low communication ranges. However, the increase in performance is higher for greater number of robots as the teams explore the areas much faster, causing the robots to explore more independently at higher communication ranges.

8.5 Conclusion

In this chapter, the distributive cooperative 3D exploration algorithm was adapted to allow for sensors with a limited field of view. This is done by ray tracing from candidate configurations taking into account the properties of the sensors mounted on the robot. This addition is compared with the previous method which does not take into account the properties of the sensor. It is seen that despite the additional computational time required for the new method, there is a considerable improvement in the time needed for exploration. Also, the use of a more realistic sensor configuration brings the experiments much closer to reality.
Chapter 9

Cliff Exploration

9.1 Introduction

This chapter deals with multi-robot exploration of a 3D structure that appears from a more global perspective as a 2D surface with a predominant orientation - the motivating example is underwater cliffs.

This work is carried out in the context of the EU-project “Marine robotic system of self-organizing, logically linked physical nodes (MORPH)” where exploration of cliff scenarios is a main project goal. Cliffs are challenging to explore even with the most advanced Remotely Operated Vehicles (ROVs) steered by humans (Fig. 9.1a). The idea of MORPH is to use a number of spatially separated Autonomous Underwater Vehicles (AUV) as mobile robot-modules that form a joint MORPH supra-vehicle (MSV) in the spirit of modular robotics (Fig. 9.1b). The use of multiple components in this scenario has several advantages; among others, the AUVs can change their relative spatial configuration to adapt to different parts in the mission - e.g. provide a wider or denser sampling of the environment or have less costly components engage in higher risks by getting closer to the cliff. The AUVs are connected via virtual links that rely on the flow of information among them, especially underwater communication networks - both investigated at distant and close ranges - and visual perception at close range. Without rigid links, the MSV can reconfigure itself and adapt in response to the shape of the terrain. The intent is to map underwater environments with great accuracy, especially in situations that defy existing technology: namely, underwater surveys over rugged terrain and structures with full 3D complexity. This includes walls with a negative slope, where precise localisation of a single vehicle is not possible.

Underwater coverage algorithms often deal with generation of bathymetry maps or planning on them. Many approaches use motions on the surface of the water in order to produce bathymetry maps of the sea floor [Jung et al., 2009, Hert et al., 1996]. Recent approaches have started 3D navigation to investigate 2.5D bathymetry maps [Galceran and Carreras, 2013].

As the purpose of this exploration is to explore only the cliff, and not the entire
(a) Even missions with human operated ROVs (left) are challenged at cliff scenarios (center), which is the reason why the concept of supra-vehicle in form of multiple closely coupled AUVs is investigated in the MORPH project (right).

(b) The MORPH supra-vehicle consists of several closely coupled AUVs that adapt their relative spatial positions during the mission, i.e. they "morph" into different “shapes” that best support the current task.

Figure 9.1: Summary of MORPH project.
Figure 9.2: Sensor (S) viewing a local plane. The area between R and L represents the area in the field of view of the sensor.

map, the task then is changed from finding all the free space in the environment, to finding all the occupied space. In order to achieve this, only frontier cells which are border an obstacle cell in the map are taken. Also, when investigating an environment such as a cliff, it is necessary to sample the environment uniformly. A state of the art lawn mower exploration pattern, whether carried out by a single robot or by multiple robots, leads to a very uneven sampling of the terrain and even many parts where small patches are not sampled at all due to the local 3D structure.

In order to get the highest density of sensor data, it is necessary to orient the sensor perpendicular to the object. Though this may seem intuitively correct, it is formally proven below. Consider, a large object such as a cliff in the sensor’s field of view at a distance $H$ from it (Figure 9.2).

![Diagram](image)

| $S$ | The Sensor |
| $R$ | The point where the rightmost sensor beam hits the object |
| $L$ | The point where the leftmost sensor beams hits the object |
| $\theta_R$ | The angle of rightmost beam of the sensor |
| $\theta_L$ | The angle of the leftmost beam of the sensor |
| $\theta$ | the angle of the centre beam of the sensor, i.e. the 0 angle of the sensor |
| $H$ | Distance to the object |
| $FOV$ | Field of view of the sensor |
| $D$ | The length of the object inside the sensor’s field of view |
In $\triangle SAR$

$$\frac{SR}{\sin(\frac{\pi}{2} - \theta_R)} = \frac{H}{\sin(\frac{\pi}{2} - \theta_R)}$$

$$\Rightarrow \quad RB = \frac{H}{\cos(\theta_R)}$$

In $\triangle SRL$ using Law of Sines

$$\frac{D}{\sin(\theta_L - \theta_R)} = \frac{\overline{SR}}{\sin(\frac{\pi}{2} - \theta_L)}$$

$$\Rightarrow \quad D = \frac{H}{\cos(\theta_L)} \sin(\theta_L - \theta_R)$$

$$\Rightarrow \quad D = \frac{H}{\cos(\theta_L)} \sin(FOV)$$

Let $H \sin(FOV)$ be a constant $K$

$$\Rightarrow \quad D = \frac{K \sin(FOV)}{\cos(\theta_R) \cos(\theta_L)}$$

$$\Rightarrow \quad D = \frac{K \sin(FOV)}{\cos(\theta - \frac{FOV}{2}) \cos(\theta + \frac{FOV}{2})}$$

$$\Rightarrow \quad D = K \sin(FOV) \sec(\theta - \frac{FOV}{2}) \sec(\theta + \frac{FOV}{2})$$

Differentiating both sides

$$\Rightarrow \quad \frac{dD}{d\theta} = K \sin(FOV) \left( \sec(\theta - \frac{FOV}{2}) \sec(\theta + \frac{FOV}{2}) \tan(\theta - \frac{FOV}{2}) + \sec(\theta - \frac{FOV}{2}) \sec(\theta + \frac{FOV}{2}) \tan(\theta + \frac{FOV}{2}) \right)$$

$$\Rightarrow \quad \frac{dD}{d\theta} = K \sin(FOV) \sec(\theta - \frac{FOV}{2}) \sec(\theta + \frac{FOV}{2}) \left( \tan(\theta - \frac{FOV}{2}) + \tan(\theta + \frac{FOV}{2}) \right)$$

For minimum

$$\Rightarrow \quad 0 = \tan(\theta - \frac{FOV}{2}) + \tan(\theta + \frac{FOV}{2})$$

$$\Rightarrow \quad 0 = \tan(\theta - \frac{FOV}{2}) + \tan(\theta + \frac{FOV}{2})$$

$$\Rightarrow \quad \tan(\theta - \frac{FOV}{2}) = -\tan(\theta + \frac{FOV}{2})$$

$$\Rightarrow \quad \theta - \frac{FOV}{2} = -\theta - \frac{FOV}{2} \pm n\pi \text{ where } n = 1, 2, 3...$$

$$\Rightarrow \quad \theta = 0 \pm \frac{n\pi}{2}$$

90
However, for $n > 1$, the sensor can no longer see the object, therefore

$$\theta = 0$$

is the only valid solution.

Therefore, in order to get the densest scan of an area, it is necessary to view the object with the sensor pointing directly at the object. Furthermore, the power of the returned signal is higher when the beams hit the object directly, than at steeper angles.

### 9.2 Algorithm for Determining the Orientation for a Pose

This section describes the algorithm to determine the orientation of one of the robots based on the current map and the current pose of the robot. The goal is to align each robot such that its sensor looks perpendicularly at the local surface of the cliff, i.e. it

- maximises the sampling density
- follows the local gradients of the 3D structures and hence achieves a (as) complete (as possible) coverage

For this purpose, a local plane is fitted into the environment. The desired rotation of each robot is calculated for each generated pose by performing ray traces on what has already been mapped to get the potential obstacle points in the sensor’s field of view. These points are then used to generate the optimal local plane containing these set of points using the algorithm from [Vaskevicius et al., 2010]. The robot is then oriented towards the normal of the plane. However, since only the yaw of the robot can be controlled in typical AUVs, the normal is projected onto the XY plane and the robot turns towards the projected normal.

Furthermore, it is possible that turning towards the normal would cause the robot to turn drastically away from the cliff or the previous orientation. This generally happens when not enough information is available in the potential area to be scanned by the robot or due to a rapid change in the slope of the cliff. Such a situation is shown in Figure 9.3. In this figure, Pose 1 represents the original pose of the robot, Pose 2 represents the new pose of the robot at the same orientation as Pose 1, and Pose 3 represents the pose aligned with the normal of the newly fitted plane on the local environment patch. As can be seen, due to the rapid change in the observed cliff, the robot will turn away from its current orientation, almost facing away from the cliff. This behaviour is not desirable, as this would result in the sensor data not having enough of an overlap with the previous scan, respectively the map generated so far, preventing proper registration of the sensor data for localisation.

Therefore, the maximum angular extents of the obstacle points are calculated. The maximum rotation is then capped such that at least a certain percentage (in
Figure 9.3: Sudden changes in the local orientation of the cliff can cause the robot to change orientation drastically. Pose 1 is the current pose of the robot. Pose 2 is the proposed position of the robot with the same orientation. Pose 3 is the proposed pose after changing orientation.

In this case 50% of the obstacle points lie in the new field of view of the sensor. The complete algorithm for calculating the rotation of the robot at a certain pose is shown in Algorithm 3.

### 9.3 Results

Experiments were conducted in the *Azores-Cliff* scenario (Figure 3.8c) which is based on a real location at the eastern slopes of Monte da Guia, Faial Island in the Azores. This environment covers an area of 140m×150m×14m. The robots were equipped with a front looking sensor with the same horizontal and vertical field of view and a sensor range of 10 meters. The robots are set in a corner of the environment at the start of the exploration process. Each experiment is performed 10 times and the results are aggregated. The effect of communication range on the cooperative exploration algorithm is tested under different fields of view, namely 60° seen in Figure 9.4, and 120° seen in Figure 9.5.

Firstly, it can be seen that the total time needed for exploration decreases with the increase in the number of robots. For a field of view of 60°, we see a change
Algorithm 3: Changing the Orientation of the Robot at a Certain Pose

**Input:** Input Pose $inputPose$

**Output:** The Output Pose with the desired rotation $outputPose$

1. $obstCells = \text{find all obstacles in the sensor field of view}$

2. $\{\text{minAngle, maxAngle}\} = \text{findMinMaxAngles}(obstCells)$

3. $midAngle = \frac{\text{minAngle} + \text{maxAngle}}{2}$

4. $\text{maxAngleRotLimit} = \frac{\text{FOV}}{2} + \text{midAngle}$

5. $\text{minAngleRotLimit} = -\frac{\text{FOV}}{2} + \text{midAngle}$

6. $\text{plane} = \text{findOptimumPlane}(obstCells)$

7. $\text{forwardVector} = \text{plane.normal}$

8. $\text{forwardVector}.z = 0$

9. $\text{upVector} = \{0, 0, 1\}$

10. $\text{rotationAngle} = \text{find rotation using forwardVector and upVector}$

11. **if** $\text{rotationAngle} < \text{minAngleRotLimit}$ **then**

12. \hspace{1em} $\text{rotationAngle} = \text{minAngleRotLimit}$

13. **end**

14. **if** $\text{rotationAngle} > \text{maxAngleRotLimit}$ **then**

15. \hspace{1em} $\text{rotationAngle} = \text{maxAngleRotLimit}$

16. **end**

17. $outputPose.rotation = inputPose.rotation + rotationAngle$

18. $outputPose.translation = inputPose.translation$
in the median travel time from approximately 900 seconds in the case of 1 robot to less than 300 seconds in the case of 6 robots, a decrease to almost a third of the one robot case. The same effect is seen in the case of 120° field of view, where the median travelling time drops from 475 seconds in the case of 1 robot to around 200 seconds in the case of 6 robots, i.e. a drop of more than 50%. The robots are able to explore independently of each other as the utility function ensures that configurations in which robots explore frontiers different from each other are granted higher utilities. This is done by taking the set union of all frontier cells that can be seen by a particular robot configuration, instead of adding the number of cells seen by individual robots. Hence, the utility function penalises robots exploring the same area while other options are available, enabling the exploration to take advantage of the increasing number of robots.

Another result which is apparent is that the increase in the field of view of the sensor decreases the total exploration time. This is seen across all communication ranges and robot numbers. For 1 robot, the median exploration time decreases from 870 seconds in the case of 60° field of view to 470 seconds in the case of 120° field of view. The same effect is seen in the case of 6 robots, where the median exploration time drops from approximately 300 seconds in the case of field of view of 60° to 200 seconds in the case of 120° field of view. When the field of view of the sensor increases, the amount of area that is visible from a particular pose also increases. As such, the robots can stay further apart and cover the environment much faster, allowing the exploration time to decrease further.

A surprising result which is seen is that the total exploration time is not affected significantly by the communication range. It would be expected that when the communication range increases, the robot team would have greater freedom to move around. Hence, they would be able to explore the environment more efficiently. However, this does not happen. The reason for this can be explained by looking at the progress of the exploration algorithm over time in this particular scenario. As mentioned previously, the robots are spawned in one corner of the environment. In this situation, all robots have to move in the same direction to explore the environment. Due to this, the distance between the robots does not increase, causing the communication range to have very little effect on the exploration time. Even at low communication ranges, the robots are able to explore the environment without going out of communication range of each other and hence causing no deadlocks (local maxima during exploration). Roughly speaking, the limitations in the fields of view of the sensors are stronger constraints than the maximum possible distances between the robots as a result of increasing or decreasing communication ranges.

9.4 Conclusion

In this chapter, the previous method taking into account the properties of the sensor is adapted in the context of cliff exploration. Furthermore, the multi-robot exploration is augmented with a method which aims to maximise the sampling of the
Figure 9.4: Median, upper and lower quartile of travelling time with respect to communication range with $60^\circ$ field of view sensor in the Azores-Cliff scenario.

Figure 9.5: Median, upper and lower quartile of travelling time with respect to communication range with $120^\circ$ field of view sensor in the Azores-Cliff scenario.
environment, causing much denser scans than a simple lawn mower pattern. Also, by adapting the rotation of the vehicles to the environment, the method does not rely upon the user to chose a suitable orientation for the robots beforehand. The effect of sensor field of view and communication range is investigated. It is shown that the multi-robot exploration is able to take advantage of the increased number of robots.
Chapter 10

Conclusion

10.1 Summary of Contributions

As mentioned, almost all work in the field of multi-robot exploration involves 2D navigation or at most 2.5D motions for land robots that are bound to a surface even in 3D environments. Even for aerial vehicles (UAV/MAV) cooperative exploration is so far been done predominantly in 2D, i.e. all vehicles are kept at a fixed elevation above ground to survey an area underneath them. This thesis dealt extensively with the problem of unrestrained 3D exploration.

During the course of this thesis, we started off with a centralised multi-robot exploration algorithm controlling abstract point robots operating in a 2D grid, i.e. the work done by Rooker et al. [Rooker and Birk, 2007]. This was extended to a distributed system operating in realistic environments in Chapter 5 and tested in a simulator which was developed during the course of this project and is described in Chapter 3. The simulator allowed for testing the exploration algorithm in the presence of vehicle dynamics which allows for an easier transition to real world systems. The next state in the path to real world systems was to extend the algorithm into three dimensions, which was done in Chapter 6. This allows underwater vehicles to travel across the depths of their environment which is vital for many marine tasks. Chapter 7 further improves upon the exploration algorithm by preventing unnecessary movements, thereby making the exploration more efficient and faster. As most robot systems are not equipped with sensors which provide an omnidirectional view of their surroundings, the algorithm is adapted to take into account the properties of the sensor in Chapter 8. It is seen that the system which takes into account the parameters of the sensors performs significantly better than the previous method. Finally, in Chapter 9, the algorithm is adapted to explore a cliff structure. Here, an additional feature was added to the algorithm which attempted to maximise the sensor coverage of the environmental structure by viewing the local environment perpendicularly. Furthermore, it is ensured that there is a considerable overlap with prior collected data, allowing for “easy” registration of the data into the map. Thus, constraints from other modules such as mapping constraints are also taken into ac-
count by the exploration algorithm. Therefore, the system is made more realistic during the course of the thesis.

10.2 Future Work

During the course of this thesis, a lot of effort has been put into ensuring realistic constraints on the different modules of the system. This can be further extended by taking into account more realistic communication models such as adding line of sight constraints. Also, instead of using simple binary communication constraints, a more realistic communication model with varying signal strength can be used. Also, any model used will be a simplification on the reality of complex factors affecting the communication link. Therefore, the algorithm should be further tested in conditions where the communication link does not match the model.

A problem that was realised during work done on this thesis is that during the process of exploration, the unknown areas are broken up into smaller patches, which are practically ignored until the end stages of exploration. In the end, the robot teams spend considerable time to complete the exploration process to get to these “skipped” areas. While this thesis attempts to mitigate the problem by performing this end stage more efficiently by preventing unnecessary movements, perhaps more work can be done to prevent these areas from forming or detecting them faster in the first place.

While the aim of this thesis was to develop an exploration algorithm which always keeps the robots in communication range, with the advantage of any failure being immediately detected, some work might be done to allow the robots to break the communication link for the purpose of efficiency. Safe ways to do this could be investigated.

Finally, the exploration module has been integrated into a full fledged control system. However, since the focus of this thesis was only on the exploration module, there were assumptions and simplifications made for the other modules. In order to implement the system in real robot systems, these modules need to be replaced by variants which can operate with real data and under real conditions. Once all the factors stemming from real world systems are taken into account, the exploration algorithm can be implemented on a real world multi-robot system. This is expected to happen during the course of current ongoing projects.
Bibliography


100


104


