Self–Organised Multi Agent System for Search and Rescue Operations

Panteha Saeedi

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University College London.

Department of Computer Science

University College London

August 18, 2010

Human beings are members of a whole,
In creation of one essence and soul.
If one member is afflicted with pain,
Other members uneasy will remain.
If you have no sympathy for human pain,
The name of human you cannot retain.

Saadi of Shiraz

Abstract

Autonomous multi-agent systems perform inadequately in time critical missions, while they tend to explore exhaustively each location of the field in one phase with out selecting the pertinent strategy. This research aims to solve this problem by introducing a hierarchy of exploration strategies. Agents explore an unknown search terrain with complex topology in multiple predefined stages by performing pertinent strategies depending on their previous observations. Exploration inside unknown, cluttered, and confined environments is one of the main challenges for search and rescue robots inside collapsed buildings. In this regard we introduce our novel exploration algorithm for multi–agent system, that is able to perform a fast, fair, and thorough search as well as solving the multi–agent traffic congestion.

Our simulations have been performed on different test environments in which the complexity of the search field has been defined by fractal dimension of Brownian movements. The exploration stages are depicted as defined arenas of National Institute of Standard and Technology (NIST). NIST introduced three scenarios of progressive difficulty: yellow, orange, and red. The main concentration of this research is on the red arena with the least structure and most challenging parts to robot nimbleness.

Acknowledgements

I would like to acknowledge the valuable contribution of many people who have helped me directly or indirectly in the accomplishment of this work. My supervisor, Soren Aksel Sorensen, has played a key role in guiding my first steps in the world of research. I am greatly appreciating Stephen Hailes for his kind support and valuable advice and Lica Capra for her generous guidance. I am also grateful to Michele Sama for his continues support and constructive comments all through my thesis. I would also like to thank Mohammad and Steffen for their interesting and helpful discussion on various topics related to this work. Many thanks to all those not named here and who showed interest in my work and guided me through this thesis . Last but not least, I take this opportunity to express my profound gratitude to my beloved parents, for their constant support and love all through my life.

Panteha Saeedi

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this thesis we propose and evaluate an adaptive algorithm for a disaster scenario in which the buildings of a major town or city have collapsed, major roads have become impassable and utility networks and power lines have failed. Victims of the disaster must be rescued within 48 hours, after which their chances of survival rapidly diminish, and the rescue becomes a recovery. Rescuers must make quick decisions, they must verify the location and status of victims and attempt to get them to safety, often endangering their own safety in the process. The initial rescue groups must ascertain the stability of the remaining structures as quickly as possible to ensure that medics and fire–fighters have access to the disaster area and the ability to perform their duties. These tasks are currently, mainly, performed by human operatives and trained rescue dogs, who must perform their work in potentially life-threatening circumstances. In many situations, rescue teams discover that even dogs trained and equipped for search and rescue are unable to negotiate the rubble and detritus. In addition to physical obstacles, the dust laden air diminishes the animals sense of smell, rendering much of their rescue capability useless. There is, however, an increasingly available alternative. Recent technological advances make it possible for certain practical and economic circumstances, robots can be sent into areas that are structurally unstable or contaminated and therefore unsafe for navigation by humans and rescue animals. Exploratory robots allow early penetration of a disaster site without placing the lives of human operatives at risk.

The 2008 earthquake in southern Sichuan (China) has shown that the number of trapped victims can be extremely high in a densely populated area (Official figures state that 69,197 are confirmed dead, and 374,176 injured, with 18,222 listed as missing). The impact of disasters in urban areas therefore requires the rapid response of specialist rescue teams, to locate victims within the rubble, devise an access route to extract them, and provide them with stabilising medical treatment. In order to maximise efficiency and effectiveness, one rescue team should be able to achieve all of these objectives, a possibility greatly enhanced by the provision of adequate numbers of search and rescue robots.

1.1 Research Challenges

A team of exploration robots should overcome six important challenges during their exploration procedure inside structural collapses: 1) Size limitations for robots to send inside collapsed buildings; 2) Urban Search and Rescue (USAR) has an extremely rugged terrain for smaller robots, therefore power

limitation will be an issue; 3) Sensor limitation for navigation and exploration, the absence of a laser range finder, and no accurate GPS data; 4) Transition errors, thus uncertainty in robot's information distribution; 5) Navigation inside completely unknown, and unstructured environments; 6) Traffic congestion caused by multiple robots negotiating long narrow pathways.

In this thesis our contributions are focused on the challenges as follows:

a performance aware robot should overcome its positional uncertainty, caused by transition errors;

a team of robots should avoid the initial congestion while entering a cluttered and confined terrain.

1.2 Motivation

To avoid the redundancy of revisiting an explored region as much as possible, robot should move to the boundary between open explored space and unexplored space. In such exploration techniques a robot moves to a frontier, that can see into unexplored space and add new information to its memory and constantly increase its knowledge of its world. However, the draw back of such techniques, for robots that rely on odometry data, is the negative effect of *transition uncertainty* on the collected information. This will lead the robot to an uncertainty that may cause dramatic errors in its positional understanding. Therefore, we need to develop a look ahead model that can be used to deal with such uncertainty through robot's belief distribution.

In this regard, we believe that there is an absence of good solutions to this problem and a good solution will be the key to improve the practical effectiveness of robot teams for urban search and rescue.

1.3 Research Hypothesis & Objectives

By undertaking mathematically established algorithm coupled with probabilistic information distribution (where beliefs of agents are included), a multi–agent system is able to reduce the redundancy of revisiting a search field inside the narrow gaps and small voids of a pancake collapsed building.

In other words we want to reduce the overall victim(s) discovery time, independently of victim's position, by performing a fair search to reach a certain percentage of optimal discovery time, i.e. our defined oracle search technique.

We detail our hypothesis as follows: To estimate the overall success of our hypothesis, for fast localisation of trapped victims, we will measure metrics (output). Robots spread out as much as possible to reach their goal (metric:'discovery time'). Robots should also report back their findings (recorded occupancy grid memory) to their base station. In this regard robots should cover the complete search field (metric:'coverage time'), since the number of trapped victims (in addition to their location) inside pancake collapses are unknown to the search team.

Our main objectives are: 1) To reduce the positional uncertainty for a search robot and to reduce search redundancies caused by revisiting explored areas; 2) How to prevent initial congestion inside narrow pathways; 3) To identify a *suitable* number of agents, within a team, according to the exterior size of a search terrain. *Suitable* number, is defined as the minimum number of agents that minimise the discovery time.

1.4 System Design & Implementation

To prove the effectiveness of our multi–agent exploration system we have designed and implemented a simulation tool that is a able to generate various model of structural pancake collapses as well as evaluating the system performances through a set of metrics.

We have designed an advanced frontier–based search agent that is aware of its positional uncertainty as well as its performances. It optimises its time performance while minimising its uncertainty. The performance of our developed system (metric: victim discovery time) is compared against other competing algorithms, such as ANT, TREE, and Brick & Mortar. We also design an oracle algorithm, which robots are aware of the victim's location (as well as their own location), this is known as our best performance in the case of discovery time. Finally, to avoid initial congestion when several agents are moving inside a cluttered and confined search terrain, a team formation is defined.

1.5 Contributions

A hierarchical control system for a search and rescue team consisting of heterogeneous robots has been designed, and is discussed in this thesis. It includes new aspects of area surveying, and exploration strategies in order to address the special requirements introduced by a search hierarchy inside disaster areas. We will address three main achievements through this thesis:

- We model our environment, to evaluate our exploration algorithms, even though there are no post–disaster maps available to define our scenario. In this regard we discuss an algorithmic approach to generate realistic after disaster test fields for search and rescue agents inside collapsed building. This has been published in Electronic Engineering and Computing Technology Journal [SS10].

- We introduce an *On–line metric* that is used by the agent during its task to estimate its performance against expectations and thus assist it to achieve its goal. This novel performance metric can be assessed by the agent during each time step to estimate the expected action utility. The performance of a frontier–based exploration algorithm can be improved when a reward function is used to reward it according to this on–line metric during each time step. This has been presented in IEEE international workshop on Safety, Security, and Rescue Robotics (SSRR09) [SSH09].

- We devise a multi–robot system, a novel path–planning algorithm for multi agent system in which behaviour is an emergent property of the approximate solution of an underlying optimisation problem. This is a novel and advanced frontier–based exploration strategy, which considers transition errors in to account, thus it is able to deal with agent's positional uncertainty. In result, victim discovery time and coverage time, for search fields of various complexity, has been reduced significantly comparing to competing multi agent exploration algorithms.

Additionally, we are able to predict the single goal discovery time for Brownian Random Walk algorithm according to the power spectral density of search fields. Therefore we manage to develop a mathematical–based benchmarking technique. This has been presented in IEEE International Conference on Computational Intelligence and Software Engineering (CISE09) [SS09b]. We investigate several

spreading techniques for a hybrid system of multi–agent. The spreading technique enables the system to avoid any initial congestion inside our cluttered and confined voids.

1.6 Scope & Assumptions

When traditional tracked robots search voids inside structural collapses they are not always able to crawl inside and look for survivors, size limitation. Therefore in our exploration system these robots send in a team of limited sensing mini robots for fast victim localisation instead of using camera mounted on the pole. To advance the state of the art in exploration strategy for these small and limited energy robots, carrying a short range sensors: 1) we model such robots and evaluate their performances inside simulated collapsed buildings, prior to implementation; 2) our simulated robot (agent) moves to collect new information about the simulated environment while considering its uncertainty through its collected belief points; 3) in turn, the collected information and the belief distribution provide clues to rescue team about the most viable areas of exploration that will further increase knowledge of the environment; 4) mother robot in a heterogeneous robot team is able to merge the collected information from several robots into a single global information to provide a more comprehensive view of the environment to the rescue team.

Inside damaged and partially collapsed buildings within disaster areas, there are locations which have not yet entirely collapsed, known as life safe voids [Mur04]. In USAR operations, robots are sent inside these confined spaces, in order of *0.5* to *2* meter. A robot platform in order of *0.33* meter square, or even less, is required to enter these voids. Furthermore, inside these confined and cluttered voids there are obstacles with unknown sizes, orientations and locations. USAR operations demand high power, while smaller robots carry smaller batteries. Unfortunately most of the energy might be spent getting to an area of interest- i.e. life safe voids. As a result of these limitations we assume a 'marsupial team' [Bal98] consists of a mother robot which transports and supports power for one or more 'daughters', much as kangaroo carries a joey.

A multi–agent system is performing its exploration strategies inside various *2–dimensional static simulated environments*. These search environments are all divided into equal square cells. All the passable cells (they are not occupied) inside the search environment are traversable by our agents. Additionally, this research obeys practical assumptions; therefore we have limitation on internal resources `(ENERGY-LIM)`, sensor limitation `(SENSOR-LIM)`, limitation on the number of mini robots `(NUMBER-LIM)`, and *transition errors*. Since we have a very limited observation view we assume that this limited sensor readings are perfect (i.e. the minimum observation to avoid collision).

1.7 Thesis Structure

This thesis is organised as follows:

• Background and related works are discussed in *Chapter* 2. We discuss all the phases for the search and rescue according to fire–fighters planning. In this Chapter we mainly focus on requirements of an autonomous self–organised exploration system. We present an introduction for each contribution. These are: an introduction on generating maze like search environment by agent's chaotic trails and

differentiating them by fractal dimension, introducing frontier–based algorithm and its disadvantages in the presence of transition errors, how to solve positional uncertainty by applying POMDP techniques, and finally team formations.

• Being able to compare different search strategies in various search fields is crucial to attain fast victim localisation. Thus we discuss *an algorithmic development and proliferation of realistic post–disaster test fields for search and rescue simulated robots* in *Chapter* 3. Additionally we introduce indices to differentiate simulated search terrains and define their complexity.

• we discussed our online metric to develop a frontier–based exploration strategy in *Chapter* 4. This expected utility will asset the agent to decide: i- how to move to the frontiers (regions on the border between open space and unexplored space); ii- what to do in the face of uncertainty (odometry errors that lead to positional uncertainties). Furthermore we have evaluated a random path–planning, ANT algorithm benchmark, in order to compare our strategy to an unstructured technique. To design our multi-agent system, we initially hope to develop an advanced technique for all agents individually, involving no collaboration.

• In *Chapter* 5 we will examine the risk/utility relationship. In our mathematical frontier–based search technique actions are executed according to their expected utility, their expected contribution to the acquired knowledge, and the risk they carry. This exploration approach is called EDAEA; it handles the problem of being stuck in local minima that affects traditional frontier–based algorithms when facing positional uncertainties. Sensors such as GPS devices are often unreliable in indoor environments and odometry data tend in practice to be noisy. In effect we have a problem of positional uncertainty, and in literature Partially Observable Markov Decision Processes (POMDP) are proposed to address the issue of goal pursuit in such environments. For kidnap robot problem we briefly point out some solution. However, we set up an experiment to recover from this problem in the next chapter.

• In *Chapter* 6 we discuss how to develop *our novel path–planning algorithms for multi–agent systems* in which behaviour is an emergent property of the approximate solution of an underlying optimisation problem. To explore an unknown environment, robots make use of their location and the history of their observations accumulated inside a search field as the robot performs the search task. They are therefore able to provide the rescue team, both human and robots, with a rough recorded grid memory of the explored areas. Furthermore, we introduce a team formation to solve the traffic congestion in our multi–agent systems.

• Our own model and strategies are evaluated against existing methods in *Chapter* 7. We also improved the ANT algorithm *extended ANT exploration algorithm (EANT)* to enable the team of agents to spread evenly inside a search terrain (adapting ANT algorithm to our scenario to avoid initial congestion). By introducing EANT algorithm we reduce the redundancy of ANT agents and enable them to be automatically aware of their full coverage of the search area, e.g. instead of exploring the field till they run out of the battery. We furthermore apply odometry errors and sensor noises on our agents, through Player/Stage tool, and test our superior multi–algorithm in more realistic experiments.

• Finally, *Chapter* 8 contains our conclusions and proposals for future works.

Chapter 2

Background

One of the challenges in disaster response is to get an overview of the degree of damage and to provide this information, together with optimised plans for rescue missions, back to rescue teams in the search and rescue field. Collapsed infrastructure, limited visibility due to smoke and dust, and overloaded communication lines make it nearly impossible for rescue teams to report the total situation consistently. It can be done by efficiently integrating data from many observers into a single consistent view. Therefore, for an effective emergency response, many decision makers are required to work together.

Naturally, local rescuers are the first rescue group that reaches a disaster area. In most cases these rescuers are emotionally attached to victims and they are not well qualified, to look for trapped victims under piles of rubble. Therefore, their rescue operation might be chaotic, e.g. causing further collapses and life threatening for both impatient rescuers and trapped victims. Nowadays with advanced technology, we can rely on *specialised robots* to provide us with a situation and damage assessment overview while exploring the entire disaster area accurately and completely to locate any survivors.

Promptness of response is a valuable quality when locating victims, since time is an important factor in search and rescue operation and therefore, the use of *multiple heterogeneous* robotic vehicles is an alternative way, to speed up the exploration process. However, it is unrealistic to think that a human operators will remotely control each of these platforms. Not only more human operators will be needed, the collaboration among human teams can also be a problematical issue, since in disaster incidents human operator's performances may be based on emotional rather than logical response.

The overall vision for disaster response is to have *multiple heterogeneous robots* operating mainly *autonomously* and actively exchanging information. They should explore the entire search field to locate all the trapped victims independent of their position. A designated rescue team should *map out* the search field while exploring the disaster site so that the location of possible victims or hazardous areas can be communicated among the rescue team during the search operation. Our research into heterogeneous teams is conducted primarily in the domain of Urban Search and Rescue (USAR), characterised by collapsed man–made structures with no controlled lighting. This terrain is often treacherous, requiring the robots to have sufficient navigational sensors in addition to their mission sensors and for rescuers wearing protective garments, and barrier materials to protect themselves while equipped with essential devices and sensors. False negative (e.g. sensors not registering an event) have deadly consequences for

a victim who goes undetected, while false positives might actively interfere with rescuers effectively in accomplishing their jobs. That is why an advance systematic approach is urgently required to deal with uncertainties, guide the rescue team and make the most of their equipment.

2.1 USAR Five–Phase Plan

Fire rescue departments are consistently responsible for responding to different USAR scenarios. Hazardous material incidents, bomb threats, collapsed buildings, and trench rescue each have their own specific requirements regarding tools and specialised tasks. Here we will focus on the rescue of trapped victims from collapsed buildings and trench rescues.

In order to understand how a robotic exploration system can be useful to the rescue team, it is helpful to review an existing rescue scenario for structural collapses. According to the Fire Department of New York (FDNY) collapsed building rescue efforts should follow a Five–phase plan [Adm96]:

1) *Global Survey of the site*

2) *Rescue of the casualties on the surface*

3) *Rescue of the lightly trapped casualties*

4) *Exploration of voids for trapped victims*

5) *Removal of selected debris*

The goals of these phases of search and rescue operations are: to rescue the victims, and to survey the site. Surveying the search field enables the rescuers for further safe investigation, safe rescue operation, and it also speeds up the search process significantly. A rescue team starts to survey the disaster site *globally* for damage assessment, then they apply further *local surveying* inside collapsed buildings to find trapped victims and locate all voids inside the rubble. They should explore all the located voids *exhaustively*, since an unconscious wounded victim may be trapped some where in the middle.

2.1.1 Global Survey of the Site

Damage assessment is the process of determining the location, nature and severity of damage sustained in a disaster situation. It includes global surveying of the affected area, estimating the amount of loss, selecting the areas of interest for further exploration and surveying, estimating the risk of operation, and determining the level of traversability inside the search field.

An aerial view provides a better perspective with the ability to cover a large area and it enables the rescue team to focus their resources and efforts on the vital areas. Aerial vehicles have the advantage of moving at higher speeds than ground vehicles, as most of the main roads will be impassable in disaster areas. The small sizes of these vehicles also allows them to be carried around the disaster field by the rescuers to deploy whenever it is deemed necessary. In addition unmanned vehicles have advantages over manned vehicles as most of the functions and operations can be implemented at a much lower cost, which is therefore faster, safer and less vulnerable.

When the global surveying is completed, rescue teams are sent inside the disaster field to provide first aid for surface victims, *the second phase of USAR*, and explore the search field for trapped victims. Their observations should be compared to auxiliary information from the disaster area (Deploying avail-

able pre–disaster maps of the city and the national seismic hazard maps), for instance in the case of an earthquake location of major population density like schools and shopping malls.

2.1.2 Local Surveying the Site

In the third phase of the *rescue mission*, the rescue team should save those victims that are lightly trapped and can be rescued without further damage to the search site. Additionally, they should explore the search field and look for more victims. Lightly trapped victims are those who can be removed by one or two rescuers, usually by removing a piece of furniture or light rubble. Traditionally lightly trapped victims are mainly localised by search dogs, acoustic devices, and thermal cameras [Adm96].

The use of robots for search and rescue had been discussed in the scientific literature since the early $1980s$ but no systems had been developed or fielded. The first known actual use of robots for urban search and rescue was at the World Trade Centre disaster on the 11th of September 2001. Robots were used here to search for victims, finding alternative routes that would be quicker to excavate by rescue team, structural inspection, and the detection of hazardous materials.

One robot that is ideal for the third phase of USAR operations is the RAPOSA robot by Marques et al. [MCL$^+$06]. It is designed and built to operate in outdoor environments hostile to the human presence, such as structural collapses. The robot is targeted for the tele–operated detection of potential survivors using a set of sensors gathering information that is transmitted to a remote human operator. An innovative feature of their work is the use of wireless communications, with an option for tethered operation. The tether carries both power and communications, with an access point on its end, and can also be used to suspend the robot inside a deep hole. The RAPOSAs mechanical structure consist of a main body and a front body, whose locomotion is supported on tracked wheels, allowing motion even when the robot is upside down. However, RAPOSA is incapable of transporting mini robots around the search field. This should be added to its capabilities, to carry several mini robots and deploy them in areas inaccessible due to size constraints.

Inside the collapsed buildings, where the roof is almost in contact with the ground, there are only few narrow accessible sites and the search site is not stable and safe for rescue team. Victims are often trapped under large pile of rubble, where even trained search dogs and infrared cameras are helpless to locate them. There is also size limitation that prohibits traditional tracked robots from crawling inside these small voids to look for survivors. Therefore these narrow entrances should be mapped carefully by the rescue team, and further advance exploration should be applied.

2.1.3 Microscopic Surveying the Site

The forth–phase plan, search inside the void spaces, requires trained and especially equipped rescuers working in conjunction with structural engineers. It is within these voids (i.e. life safe voids) that causalities may have had enough room and air to survive [HB07]. Therefore when narrow but accessible entrances are found they should be investigated thoroughly.

The potential number of victims are evaluated, search operations should commence in an orderly manner, beginning with verbally calling out for victims to identify their location, and searching using a systematic search pattern. Searchers should stop frequently to listen for noises or attempted commu-

nication from victims; often this can involve all searchers stopping activity at specified periods of time to listen. In situations where multiple structures are searched, the outside of buildings can be marked. Marking systems indicate which buildings have already been searched, the results of the search, and to avoid duplication of search efforts.

The conventional approach to searching under the rubble for survivors involves the use of 'bucket brigades'. In this traditional technique hundreds of rescue workers remove rubble by passing buckets from hand to hand. This is a very time consuming task, it involves physically checking under all the rubble; furthermore it may cause a second disaster by causing more collapses. Traditional search equipment for structural inspection are cameras mounted on poles [Mur04]. Through this method rescue team are able to speed up the operation to some extend, by limiting the available voids by removing debris. This technology has limited function; the available poles are less than two meters long, and for deeper voids they are not able to provide any useful information.

In this phase only *mini robots* are able to enter through confined access under the metal roof or through breaches. They should explore overall *maze of obstacles* and *debris* to look for trapped victims or even hazards. They are the rescuer' eyes inside the dangerous structural collapses. When the casualties are located inside unstructured and partially collapsed buildings, they can remotely assess and triage. Figure 2.1 is an image of a 'life safe void' that is evident from outside, as you can see the victim is observed easily by the rescue team. Unfortunately the majority of these unknown, and cluttered life safe voids are unobservable from the surface.

Figure 2.1: An example of a *life safe void*

Structural collapses come in three different types: *pancake collapse*, *lean–to collapse*, and *v–type collapse* [MCM01]. A pancake collapse is characterised by both supporting walls or the anchoring system is failing and the supported roof or upper floor falling parallel to the floor below. Small voids where victims can be found are created by debris. A lean–to collapse occurs when only one side of the supporting walls or floor anchoring system fails. One side of the collapsed roof should be attached to the remaining wall or anchoring system. The lean–to collapse creates a significant void near the remaining wall. A v–type collapse can happen when there is a large load in the centre of floor or roof above. Both

sides of the supporting wall are still standing but the centre of the floor or roof above is compromised and thus collapsed in the centre. The v–type collapse usually leaves a void on each side of the supporting walls. Knowing where the voids are located can help fire–fighters to locate survivors of the structural collapse much faster and safer.

In this thesis, we concentrate mainly on the pancake collapses, since life safe voids location on the other two types of collapses, are located next to the existing walls. However, in the pancake collapses the locations of these voids are completely unknown.

2.1.4 Rescue Operation and Recovery

As discussed above, the majority of robotic research for rescue situations has in the area of search and victim localisation [MJ07], or information gathering for the coordination of operators. Once victims have been found, human rescue personnel are typically called in to assess and finish rescue personnel, including victim (Here, we refer to victims as people who cannot rescue themselves; they are typically unconscious) manipulation. Manipulation here means moving the head, limbs or the whole body for extraction from dangerous situations.

Rescue operations have generally been avoided by the robotic communities because an improper manipulation often can result in further injury or death of the victim. Yim et al. [MY09] present some of the issues and steps towards robotically aided victim manipulation including simulation and empirical measurements for specifications of a robot, as well as solutions to components of this problem. Their work specifically focused on an automatically deployed mechanism that could be used by rescuers to quickly and easily drag victim or to attach a rope to them, so a personnel can pull the victim to safety from a safe distance. As a rule of thumb, any situation which either a human or a robot can rescue a victim, a human rescuer is usually a better option [MC09]. However, this might be different in situations in which only a robot can affect a rescue. These situations are typically characterised by difficult to access locations, or by the need for rapid deployment. The combination of proper equipment and trained personnel allows an accurate and fast rescue operation. When trapped victims are located by rescuers they can be removed (according to the provided data) and medical aid rendered as necessary. The removal of victims should be done with care so as to avoid any further injury. Where any neck or back injury is suspected, the cervical spine should be immobilised first before attempting to move victims, and dragging should be avoided in situations where the presence of debris (e.g. broken glass) would cause further injury.

After the last phase of the search and rescue mission is the recovery mission and cleaning up the incident area commences. Recovery is different from rescue, where it is assumed that no one is left living and rescuers are now removing bodies. However, there is a small chance that a live victim may be found. The recovery mission is usually applied $40-48$ hours after incident occurrence. Collapsed buildings can be brought down either piece by piece, as in hand demolition, or more rapidly with heavy equipment. In piece–by–piece wrecking, workers usually employ hand tools, either mechanical or thermal. Heavy equipment, such as bulldozers and wrecking balls, are faster but may yield further collapses to unstable buildings around.

2.2 Simulated Search Fields

We evaluate the performances of search and rescue robots inside simulated collapsed buildings, prior to implementation. Based on analysis of a variety of disaster site images and fire fighters reports [JME01, Sch01], we identified the essential attributes of a simulated world. Inside a structural *pancake* collapses there are pieces of material that are randomly composed. The robot is facing the following set of exploration challenges within its search field: *Maze–like environments, unstructured environments, unknown environments, and narrow pathways*.

These challenges also provide the basis for measuring performance according to the published performance guide for Urban Search and Rescue robots by the US Department of Homeland Security [oHS08]. In order to validate the functionality of the robot and its algorithms, robots must be confronted with a number of unique challenges. Realistic simulation enables researchers to perform experiments with defined conditions and to repeat these experiments.

The most complex and challenging arena defined by National Institute of Standards and Technology (NIST) is the *red arena*, i.e. Heavy structural damage. To validate an optimum search strategy, for the challenging parts of the red arena, we should test simulated millibots (selected robot in this activity) functionality inside several different simulation environments with various level of complexity. In Chapter 3 we generate various of these search fields by an agent(s) performing Brownian Random Walk. We also introduce an index, *Fractal Dimension of Pathways*, to differentiate these generated fields by their complexity. Complexity here is defined as the time to find a victim independent of its position, the larger is this time the more complex is the search field.

Fractals are irregular geometric objects with an infinite nesting in all scales. In this thesis we will verify that the fractal dimension of pathways inside the search fields, can be applied as a discriminative index to characterise test arenas for exploration robots.

2.2.1 Fractal Dimension

Many natural phenomena like discharges of rivers and outline forms of coastlines are filled with seemingly complex irregular shapes and random variations [Bur88]. We believe that we are living in a world of three–dimensional objects bounded by two–dimensional surfaces and outlined by one–dimensional lines. The mathematician *Benoit Mandelbrot* [Man93] pointed out that, clouds are not spherical, mountains are not cones and coastline are not circles. In other words the dimension of the real world does not come as tidy integers. Furthermore, in ecology there are several discriminative indices that is able to differentiate these landscape patterns. In this regard, the *fractal dimension* is an index of the complexity of shapes in landscapes. The fractal dimension provides an objective way to quantify the fractal properties of an object and distinguish it from other similar objects.

One of the important characteristics of fractals is the concept of *self–similarity*. However, for natural phenomena self–similarity refers to the statistical properties of the outline, trail or surface. For instance if the landscape is composed of simple geometric shapes the fractal dimension will be small while for a landscape that contains many patches with complex and convoluted shapes the fractal dimension will

be large. Self-similarity is a characteristic of a form exhibited when a substructure resembles a super structure: 1-growing from a unit object 2- subsequent division of original object.

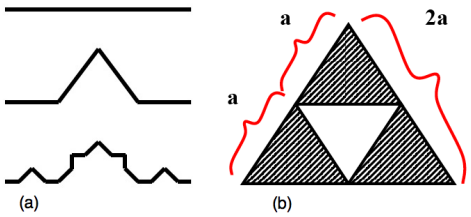

Figure 2.2: Self-Similar Fractal: (a) lines are growing from a line (b) 3 triangles divided from a triangle

Figure 2.2 (a) is indicating a line that is replaced first with 4 lines, $\frac{1}{3}$ of the original size, the fractal dimension is estimated as: $FD = \frac{log4}{log3} = 1.26$, and for the second replacement we have $FD = \frac{log16}{log9} = 1.26$. This nesting can continue to grow while the fractal dimension of generated objects remains the same as 1.26. In figure 2.2 (b) the fractal dimension is estimated as: $FD = \frac{log3}{log2} = 1.58$, since we have 3 triangles half the size of their original triangle. This is an example for fractals that are generating by dividing the original object.

The *scale transformation* described for *self–similar fractals* is *isotropic*, which means the dilation increases the size of the system uniformly in every spatial direction. Fractal objects can also be rescaled using an *anisotropic* transformation, e.g. *self–affine* fractals. Therefore in contrast to self–similar fractal, their scaling properties are depending on the direction of space, these fractals does not look similar by human eyes. We are interested in quantifying disorder surfaces (i.e. generated search fields by chaotic movement trails), thus we apply the special subclass of anisotropic fractals that is described by a single–valued (roughness) function called the *self–affine function*.

The analogue of the scaling relation for a self–affine function can be formulated as:

$$h(x) \sim b^{-H} h(bx) \tag{2.1}$$

Where H (*H* in honour of both *Hurst* and *Hölder*) is the self–affine exponent and gives a quantitative measurement of the roughness of the function $h(x)$- and x is defined on the interval of $[0, 1]$.

Mandelbrot [BS95] estimated the relationship between the roughness and fractal dimension of self–affine objects as follows:

The function 2.1 formulates the fact that a self–affine function should be rescaled both horizontally and vertically. If we scale the function with *a factor* b ($\forall\ b \in \mathbb{R}^+$) horizontally, i.e. $x \rightarrow bx$, then we should scale the function with a factor b^H vertically as well, $h \rightarrow b^H h$, in order to make sure the resulting object overlaps the object obtained in the previous generation. For the special case $H = 1$, the transformation become isotropic (instead of anisotropic) and the system therefore becomes self–similar (instead of self–affine).

To associate the self–affine function for a surface, we cover that surface with boxes of size γ. Then we divide the horizontal domain of the function into n_s segments, so the width of each segment is

estimated as $\gamma = \frac{1}{n_s}$, when horizontal domain is assumed as $x = 1$.

In the horizontal interval of size γ, the height changes according to the power law, $\Delta \sim \gamma^H$, thus we require $\frac{\Delta}{\gamma} \sim \gamma^{H-1}$ boxes to cover the function. Since we need γ^{H-1} boxes to cover the variation in one segment, for n_s segments the n_i number of boxes is required, is approximated as:

$$n_i \sim n_s \times \gamma^{H-1} \sim \gamma^{H-2} \tag{2.2}$$

Now, to measure the fractality of an object, we measure its *Hausdorf dimension*. To define the Hausdorff dimension [DDDD98] for a metric space X as a non–negative real number, we first consider the number $N(\gamma)$ of balls of radius at most 'r' are required to cover X completely. Clearly, as γ gets smaller $N(\gamma)$ gets larger. Very roughly, if $N(\gamma)$ grows in the same way as γ^{-FD} (as 'r' is squeezed down towards zero), then we say X has dimension FD. In fact the rigorous definition of Hausdorff dimension is somewhat round about, as it allows the covering of X by balls of different sizes.

Since, we have $N(\gamma) \sim \gamma^{-FD}$, we define the fractal dimension (i.e FD) for very short length scale $\Delta \ll \gamma$, from equation 2.2 as [BS95]:

$$FD = 2 - H \tag{2.3}$$

Thus, fractal dimension of self-affine objects are defined by their Hurst exponent, known as object roughness. In Chapter 4 we deploy the *Hurst exponent* (H) to predict *Brownian Random Walk* behaviour inside our generated search fields. To estimate the FD of self–affine fractals we deploy a systematic technique called box–counting.

2.2.2 Box–Counting Dimension

The box–counting dimension proposes a systematic measurement, which applies to any structure (both self–similar and self–affine) in the plane and can be readily adapted for structures in space. We put the structure onto a grid with mesh size γ, and count the number of grid boxes which should contain at least some of the structure. This gives a number, say N. The value of N is highly dependent on the size of the box, i.e. γ. In this technique we should change the value of γ to progressively smaller sizes and count the corresponding numbers, $N(\gamma)$. Through the estimated values we make a $log - to - log$ diagram, that is the logarithms $log(N(\gamma))$ versus $log(\frac{1}{\gamma})$. We apply ridge regression to the collected data to fit a straight line to the plotted points and then measure its slope, we refer to box–counting dimension as FD_{box}. This estimated FD_{box} is the box counting dimension, another special form of Mandelbrot's fractal dimension [PJS04]. Note that in the case of self–similar object the FD_{box}, might slightly differs from the real values of FD as estimated in the section 2.2.1.

Available tools that are able to compute box–counting dimension, such as MATLAB (boxcount), FD_{box} is estimated as follows: 1- the tool applied several grid boxes with different sizes on the image 2- the spatial frequency, for each grid size is estimated, i.e. the total boxes that are covering at least some part of the image (IB) over total number of grid boxes (TB), i.e. $SF = \frac{IB}{TB}$; 3- the smallest spatial frequency will be selected as SF_L; 4- $FD = D - \frac{logSF_L}{log\gamma}$, where D is selected by the user as 1D, 2D, or 3D box counting. For instance for 3D, $FD_{box} = 3 - \frac{logSF_L}{log\gamma}$. The user should also select the

minimum and maximum size of the grid cells, normally there are some limitations, for selecting these values, which are defined by each tool.

In this activity we have selected FracLac [Kar] to estimate the fractal dimension of the search fields, because of its ease-of-use. *FracLac* is an image analyse software, developed for *image J tool*. This tool is used to objectively analyse complexity and heterogeneity as well as other measures of binary digital images. It has a global binary grid scan option that applies box–counting technique across an image. It calculates the fractal dimension of our maze–like search environments using the mathematical procedure presented above.

On the contrary to MATLAB that we select the value of D, FracLac dimension is always set as $D = 3$. Therefore the FD_{box} estimated by this tool is always between 2 and 3.

2.2.3 Related Work

Voshell, et al. [VW05a, VW05b] used path tortuosity measured by fractal dimension to compare the effectiveness of two robot user interfaces. Its studies were shown that navigation through smoother paths will be resulted in lower fractal dimension values. These papers suggest that the lower fractal dimension (tortuosity) of the path indicates that the operators had higher situational awareness when using the folded perspective display, thus were able to reach the goal quicker with less searching. A similar measure of path complexity, the Lyapunov exponent, has been used to show how path complexity is related to the performance of human searching a maze [CG07]. While fractal dimension is a scale–less measure of path complexity in the spatial domain, the Lyapunov exponent is a scale–less measure of the rate of divergence of two trajectories in a dynamical system. Generally the Lyapunov exponent is used to assess the predictability of a system, in this case it is used to measure path tortuosity in the time domain (i.e. measure if the path likely to stay on the same course). A positive Lyapunov exponent indicates a chaotic path while a negative exponent indicates a smooth path. Clarke, et al.'s experiment [CG07] involved 120 participants who wore tracking equipment and searched a $5m \times 7m$ maze constructed in a lab. The results showed that the participants whose paths were most chaotic (positive Lyapunov exponent) were able to find all of the hidden items in the maze in a much shorter period of time than the participants whose paths were smooth (negative Lyapunov exponent).

At first glance it seems that the works of Voshell and Clarke contradict each other, one shows that a less complex path is related to higher performance and the other shows that a more complex path is related to higher performance. However, the works actually support each other. In the Voshell study the operators were directed to go from point A to point B within the environment; the Clarke study asked participants to completely search a maze for small tags. These goals are two sides of the same coin, one rewarded travelling in a smooth path (direct travel between points) while the other rewarded chaotic movement (complete coverage in a search).

As discussed above, the smoother recorded movements has a lower fractal dimension values while the more chaotic paths have a larger fractal dimension. Narrow pathways inside maze environments are chaotic and has a large fractal dimension compared to smooth paths.

2.2.4 Maze Generation Technique

From Clarke et al. results we were convinced to apply chaotic movements (i.e. Brownian random walk) to generate chaotic trails and we record these trails inside an empty field. This is how we are generating maze terrains. In Chapter 3, we will investigate how to develop our cluttered and confined maze–like search environments through these chaotic recorded movements. We estimate the fractal dimension of the recorded chaotic trails inside each terrain.

Clarke et al. experiment evaluated that an agent performing a chaotic movement is able to solve a maze terrain much faster than an agent performing smooth movement. Therefore, we later send in a maze solver, performing chaotic movement, to move inside such a recorded trails looking for a randomly located goal and we measure its average solving time, among 100 run times. We analyse the correlation between the fractal dimensions of recorded trails, which is developed by a *maze generator*, to the average solving time by a *maze solver*.

2.3 Exploration Strategies

When a robot functions in an unknown or partially known environment, it needs to acquire knowledge about its surroundings. Robots gather information through their sensors as they move through the search area. Sensors provide basic information that is required for localisation and navigation. Exploration has been considered as a fundamental problem for mobile robots. It involves *localisation*, *navigation*, and *planning*. An *advanced exploration process* allows the robot to efficiently, in terms of exploration time, gain knowledge about its environment and facilitates higher–level functions such as goal navigation, and modify its planning to reduce position uncertainty.

In this thesis we divide the exploration strategies in two categories: memory–based and memory–less. We mainly focus on exploration techniques that can be applied in maze searching, since our millibots are suppose to explore a maze like void inside a rubble (i.e. voids are cluttered and confined).

In the literature, famous motion planning in an unknown maze like environment techniques are:

- memory–based algorithms: Depth First Search (DFS), [Sed01], Spanning Tree [GR01], Frontier–based [EP94, Yam97], ANT algorithms [KL01] , Brick & Mortar [FTL07].

- memory–less algorithms: Random Brownian Walk [ML05]

Brownian random walk (BRW) is an unstructured search technique, we deploy this random algorithm to generate our search fields. Since this algorithm is following the Brownian motion, *we are able to differentiate the BRW agent movements by fractal dimension*, this is completely discussed in Chapter 3. We further use this algorithm as our upper limit time to discover a single goal. BRW algorithm has no memory of what cells it has visited. In USAR applications memory–based algorithms can significantly reduce the exploration time, *however observation and transition errors can become very problematic*.

Below we investigate the above memory–based algorithms (i.e. our competing algorithms), their advantages and disadvantages. We eliminate the DFS algorithm, memory–based algorithms, since DFS is also known as the *classic* spanning TREE algorithm.

2.3.1 Advantages & Disadvantages

TREE, ANT (both discussed in chapter 3, Frontier–based, and B&M (both discussed 5) algorithms are adapted to our simulation tool and simulated agent. We introduce these competing exploration techniques completely in Chapters 3, and 5.

TREE algorithm has the tendency to revisit cells, e.g. going back to its parent cell. We definitely need a more time efficient search technique for USAR applications than TREE algorithm. Agents running the ANT algorithm cannot determine when the exploration task is completed. Additionally this algorithm does not take transition errors, caused by odometry errors, into account. Therefore, an ANT agent might face dramatic errors in its understanding of its position and as a result it performs poorly. Frontier–based algorithms are known as a very effective search algorithms, just like ANT *long term transition error is a problematic issue*. Agent might not even be able to find its frontiers according to its recorded data. Brick & Mortar(B&M) algorithm, which is vulnerable to odometry errors, might end up trapping its agent inside the search field. However, Ferranti et al. [FTL08] purposed a Hybrid B & M technique, this is a combination of B&M and ANT algorithm. This hybrid technique is able to rescue an agent from trapping inside a search field, because of the transition errors, but it can not promise a full coverage of the search terrain.

In this activity our frontier–based agent estimates and records its *position belief distribution* while performing exploration to avoid uncertainty. This enables the agent to reduce its positional uncertainty and therefore redundancy of revisiting a search field significantly.

2.3.2 Heuristic & Probabilistic Strategies

To create a representation of the environment based on sensory information, a robot must know where it is. Conversely, a robot can only estimate where it is located, if it has some sort of world representation. We are assuming a scenario where the robot has been placed into an unknown environment, thus it starts exploring with neither a map nor any idea of where it is. In this scenario a robot should localise itself while recording its location on its *occupancy gird* memory [Thr02].

Path planning algorithms can operate with different space representations. Most representations use cell decomposition with a fixed number of cells to divide the search space. Most obvious is decomposition into uniform grid cells. Additionally there are spaces consisting of multi points, with various pathways (road map) that can connect these points together. Creating the essential framework of an existing memory, can serve as a pre–processing step and means the reduction of the search space for the planning algorithm. Clearly, a reduced search space has performance advantages and offers an easier application of existing path planning algorithms. Visibility graphs, Voronoi diagrams (e.g. Spanning Tree) [GR01, GMAM06] and probabilistic roadmap are all belong to this category [RN03] and there are able to generate road maps [SN04] through free space. We have selected an occupancy grid memory for our agents, since our simulated environment is divided into uniform square cells.

In potential field motion planning technique, a robot navigates in an environment. It fills its working space, e.g. occupancy grid memory, by empty or occupied information about the space. More importantly there is an artificial potential field that attracts a robot to a goal position. The potential field method

is particularly attractive because of its mathematical elegance and simplicity, and has been studied extensively for mobile robot motion planning [LWS07]. For instance *Frontier–based* exploration algorithms is a very simple technique and widely tested [EP94, Yam97]. Using this exploration technique a robot moves toward boundaries or frontiers between known and unknown environment. In this technique agent is seeking to visit new areas without considering any uncertainties in its collected data. Frontier–based exploration is relatively easy to integrate into most existing mobile robot architectures, although long term uncertainty may cause dramatic errors in the robot's understanding of its position.

The potential field motion planning techniques are categorised as: *Heuristic planners* and *Probabilistic planners*. In general *heuristic methods* are used to rapidly come to a solution that is hoped to be close to the best possible answer, or 'optimal solution'. Heuristic planners [CKK96] have been developed that are able to solve particular difficult problems in impressively low running times. However, the same planners may also fail or consume prohibitive time on seemingly *simpler ones*. The main problem with basic heuristic planner is that it does not consider any uncertainty in its collected data. This has lead to the design of a *Probabilistic planner*. A planner is called probabilistically complete if, given a solvable problem, the probability of solving it converges to 1, as the running time goes to infinity. Such a planner is guaranteed to solve any solvable problem within finite time. An advantage of such system is that it can optimise plans during missions and thus deal robustly with information uncertainty [AK01].

2.3.3 Probabilistic Exploration Strategies

In unknown and complex terrain, like our USAR scenario, frequent re–planning will be necessary. A predetermined path based on partial acquired knowledge may be too difficult to follow or reach a dead end. Therefore, enforcing any deterministic strategy would slow down the exploration time significantly.

We investigate relevant non–deterministic exploration strategies for our scenario as below:

• *Pareto* Frontier–based exploration algorithm [Bal00] is presented that visualisation uncertainty are taken into account. However, still errors in the robot transition are ignored. *Freda and Oriolo* [FO05] introduced a *frontier–based probabilistic* technique based on Local Safe Region (LSR) to avoid creating any map of the visited regions and solving transition errors. However, their technique is not applicable on limited sensing agents as our millibot (it requires laser range finder).

• More sophisticated methods, take into account other factors such as *environment uncertainty* and the *knowledge gain potential* of candidate locations for exploration. For instance, the *Next–Best–View* (NBV) algorithm (also known as greedy exploration algorithms [Moo01]) evaluates new candidate locations based on the expected information to gain, and the overlap between the current and candidate safe region [GBL02]. Unlike the frontier–based technique, the robot may choose a more distant location if that location have more information gain potential than closer places. In NBV overlap criterion is an example of incorporating the requirements of the localisation and mapping system into the exploration process. In this case, the robot will only move to new regions that overlap somewhat with the current observable region, so it can remain localised.

In NBV algorithms agent is required to build an accurate map of its environment while exploring the search field, performing SLAM [GBL02]. However, our agents are not able to create any accurate map

with their sensing limited sensors, especially in the absence of laser range finders (laser range finders are heavier and larger than our assumed robot platform).

• Another famous approach in multi–agent exploration strategies is to apply the reinforcement learning (RL) for *adaptive agents* [SSH94]. The adaptive agent adapts its strategy according to the rewards received while interacting with other agents. A major problem with this approach is its slow convergence. An effective interaction strategy is acquired only after processing a large number of interaction examples. During the long period of adaptation the agent pays the cost of interacting sub–optimally. In application such as search and rescue operations inside randomly generated maze–like environments such a RL is not practical (computationally expensive algorithms).

• Caramel and Markovitch [CM97b] present a *model–based* approach that tries to reduce the number of interaction examples needed for adaptation. Their approach splits the learning process into two separate stages. In the first stage, the learning agent infers a model of the other agents based on past interaction. In the second stage the agent utilises the learned model for designing effective interaction strategy for its future. The model–based approach gives priority to actions with the highest expected utility, according to the *current* accumulated knowledge. Thus, there is no consideration of behaviour effects of the agent on its learning process.

Later on Carmel and Markovich extended their model–based and introduced *lookahead–based* exploration strategy [CM97a, CM98] as a very interesting technique that have received significant attentions in RL research. This technique is a combination of a frontier–based and RL algorithm. In lookahead–based approach actions are executed according to *their expected utility, their expected contribution to the acquire knowledge, and the risk they carry*. Thus instead of holding one model, *the agent maintains a mixed opponent model*. Their presented work is only focussed on multi–agent decision making in game theory, rather than multi–robot exploration system. However, through this technique we are able to present an exploration strategy, which hold a practical solution among all other strategies, for our complex and unknown environments that are required to be explored in a short time–frame.

In this thesis we conform to *Caramel and Markovich technique* and introduce a dual–mode exploration system as follows:

- if the *uncertainty* of taking an action is lower than a predefined value it selects the action with the highest *utility*.

- if the *uncertainty* of taking an action is higher than a predefined value the agent selects the action with the lowest *risk*.

Where the risk is defined as a state of uncertainty where some of the possibilities involve undesirable outcome. The undesirable outcomes, for our scenario, will be investigated and defined by this author (through this thesis). Moreover, to estimate the above *utility* we introduce a *novel on–line metric* in Chapter 4, that should be estimated by an agent at each time step while exploring a search field.

2.4 Uncertainty

Imperfect sensors and approximate knowledge about the sensor readings leads the robot to uncertainty. Different compensation techniques exist for dealing with uncertainty in this context. One commonly used method is the Bayesian approach. For example the occupancy grid is commonly used in a combination with a binary Bayes filter. We later address this uncertainty in agent's memory by *coverage grid memory*, this is by storing a belief posterior over each grid cell [SB03].

As argued by *Halpern* and *Fagin* [HF90] there are two useful and quite different ways of understanding belief functions. The first is as a generalised probability function and the second is as a way of representing *evidence*. A belief function can be defined as a function that assigns to every subset of a given set $\mathcal{S}$ (or events) a number between 0 and 1. If we view belief as a representation of evidence, then it makes sense to combine them but not updating them.

In this thesis we view the belief as a representation of evidence, the more evidence we have to support a particular proposition, the greater our belief in that proposition. For instance consider the evidence that comes in the form of an observation of some event B, then this updating is typically done by moving to the conditional probability. Starting with a probability function Pr, we update it to get the conditional probability function $Pr(.|B)$. This suggest that evidence can be viewed as a function that takes as an argument a probability function and returns an updated probability function. For this viewpoint of the belief function, we discuss two existing frameworks: *Partially Observable Markov Decision Process(POMDP)* and *Dempster–Shafer theory*:

- *Partially Observable Markov Decision Process*: Markov decision processes (MDPs) provide a principled mathematical framework for modelling decision–making that can be applied on an agent making decisions in a system described by discrete set of states. In each state there are several actions from which the agent must select. Depending on the chosen action the agent receives a reward. A limitation of using the MDP framework is the assumption of an ideal observer. In particular it assumes the existence of an agent capable of observing directly the real states of a given system, which in practice is infeasible because noises and sensors' limited capabilities compromise the reading. This problem has been addressed by POMDP framework. A *POMDP* is a generalisation of an MDP in which observation disturbances of the system state can be modelled.

  A POMDP represents an agent taking a sequence of actions under uncertainty to maximise its total reward. Although the policies resulting from solving POMDPs can handle the uncertainty caused by agent's observations, the problem of that is to find an optimal policy, which is computationally expensive. There are various techniques for 'exploration robots' to reduce the complexity through POMDP planning . As a result, there have been a lot of work on computing *approximate POMDP solutions* [Hau00a, Roy03, LCK95] specifically some number of approximate point–based POMDP algorithms [NS04, ZZ01].

- *Dempster–Shafer theory*: This theory [Mor76] provides a framework for the mathematical representation of uncertainty that is based on modelling 'belief' about a set of possibilities. Dempster–Shafer differs from traditional probability theory in several key ways [BH04] as follows:

The idea of belief function was first introduced by *Dempster* [Dem67, Dem68] and later put forth as a framework for reasoning about *uncertainty* by *Shafer*. Shafer talks of belief as being a representation of an evidence. $Bel(A) = p$ is to say that, as a result of the evidence encoded by Bel, the agent has a degree of belief p in the proposition represented by the set A.

The allocation of belief mass to set of mutually exclusive possibilities, not just individual possibilities, is allowed. For this reason, Dempster–Shafer theory can represent 'ignorance', as belief mass assigned to a set of multiple possibilities (reflecting lack of knowledge about which specific possibility a piece of evidence supports). Because of its ability to represent ignorance, Dempster–Shafer theory requires no a priori knowledge about the world; in the presence of no knowledge, all belief mass is assigned to ignorance.

2.4.1 MDP and POMDP Preliminaries

When our agent is confronted with a decision, there are a number of different alternative actions from that it can choose. Choosing the best action requires considering more than just the immediate effects of its actions, e.g. one–step lookahead [CM97a]. The immediate effects are often easy to see, but the long term effects are not always as transparent. Sometimes actions with poor immediate effects, can have better long term ramifications. Ideally, you would like to choose the action that makes the right trade-off between the immediate rewards and the future gains, to yield the best possible solution. People work through this type of reasoning daily, and a *Markov decision process* might be a way to model problems so that we can automate long term decision making for our agents without increasing their complexity.

MDPs provide the principal mathematical decision modelling framework to be applied on an agent making decisions in a system described by discrete set of states. In each state there are several actions from which the agent must select. Depending on the chosen actions the agent receives a reward. Formally, a MDP can be described by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S}$ is a finite set of states of the world, $\mathcal{A}$ is a finite set of actions, the state transition function $\mathcal{T}(s', a, s) = Pr(s'|s, a)$ is the probability to reach state $s' \in \mathcal{S}$ while being in state $s \in \mathcal{S}$ and performing action $a \in \mathcal{A}$. Finally, $\mathcal{R}(s, a)$ defines the reward for choosing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. The agent goal is to maximise its expected total reward by choosing a suitable sequence of actions.

Formally, a POMDP can be described by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{Z}, \mathcal{O} \rangle$, where $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ defines an MDP, $\mathcal{Z}$ is a discrete set of observations and $\mathcal{O}(s', a, z) = Pr(z|s', a)$ is the probability of making observation $z \in \mathcal{Z}$, give some state $s' \in S$ after performing some action $a \in A$ [KLC98]. To represent the agents' preference over future rewards, a discount factor $\gamma \in [0, 1)$ is usually given, and the goal of the agent is to maximise its expected future reward by choosing a suitable sequence of actions, given the state. The expected future reward (V) is defined as $V = \mathbf{E}\left[\sum_{t=1}^{\infty} \gamma^t R_t\right]$, where R_t is the reward received by the agent at time step t.

In POMDP, the agent state is only partially observable, thus many algorithms are based on a value called a *belief*, that is simply a probability distribution over $\mathcal{S}$. A POMDP policy dictates an action for every possible belief. The belief space is continuous and has a dimensionality of $\mathbb{R}^{|S|-1}$. An approximate POMDP solution is therefore applied for solving POMDPs in a large state space. Because Markov chains

have no memory, the next state is simply defined by the action and the state transition probability. There are situations where the effects of an action might depend not only on the current state, but also on some of the previous states. The MDP model will not let you model these situations directly therefore the *state transition probability matrix* is given.

Definitions:

- **States**: When making a decision, we need to consider what are its immediate effect. The state is the way the world currently exists and an action will have the effect of changing the current state. The set of every possible way of the world will be defined as the set of states.
- **Actions**: The actions are a set of possible alternatives you can make. The problem here is to know which of these actions to select, for a specific state of the world.
- **Transitions**: When we are deciding between different actions, we have some idea of how they will affect the current state. The transitions specify how each of these actions might change the state. Since each action could have different effects, depending on the state, we need to specify the effect of each action. The most powerful aspect of the MDP is that the effects of an action are probabilistic. Imagine we are specifying the effects of doing action 'a1' in state 's1'. We could say that the effect of 'a1' is to leave the process in state 's2', and there is no consideration about how 'a1' changes the world. However, many decision processes have actions that are not this simple. For instance an action might usually results in state 's2', while occasionally it might result in state 's3'. MDPs allow us to specify these more complex actions and specify a set of resulting states and the probability that each state results.
- **Immediate Rewards**: If we want to automate the decision making process, then we should be able to have some measure of value of action, by that we are able to compare different actions. Thus we specify the immediate value for performing each action in each state.

Since POMDPs are difficult to solve, the most common approach up to this time, at least for real time applications such as mobile robot navigation, is to use a heuristic approach to planning rather than the full POMDP solution. Heuristic approaches can be divided into three categories: (a) those which do not consider uncertainty at all, (b) MDP–based heuristics which consider stochastic actions but not future uncertainty, and (c) those which can act in order to resolve uncertainty. While this section provides a brief overview, more details are available in [CKK96], [Hau00a] and the references therein.

Heuristics Without Uncertainty Considerations – *Re–plan* is a simple strategy and probably the most widely used in practice. It simply uses the assumption that the most likely state is true, and that the world is deterministic. If, during plan execution, the most likely state drifts far enough from the plan, it generates a new plan.

Our presented frontier–based DAEA search algorithm is following this heuristic technique, since we have not considered the effects of odometry errors (uncertainty) on the agent memory grid.

MDP–Based Heuristics –If the discrete state–space is sufficiently small, the solution to the underlying MDP is relatively easy to obtain. Two common heuristics based on the MDP solution are MLS

and Q_{MDP} . MLS, or **M**ost **L**ikely **S**tate, assumes that the most likely state is in fact the true state, and takes the corresponding action from the MDP policy [NPB95]. This is a good approximation to the full POMDP solution when distributions are compact, and the most likely state therefore never is far from the true state. Q_{MDP} requires the entire MDP value function, and can be viewed as a voting system [LCK95]. Given a belief over discrete states, each state votes on actions. The number of votes a state s_i can cast is proportional to the probability that s_i is the true state. The state casts its allotted votes by voting on actions in proportion to their MDP value. After voting, the agent selects the action with the most votes. Q_{MDP} effectively assumes that all uncertainty will disappear after it takes its action [CKK96]. Indeed, it would be optimal if this assumption was true. However, it can fail when uncertainty is large, and it is unlikely to disappear after a single action. In highly uncertain scenarios, Q_{MDP} will take an action that is reasonable (in terms of reward) in most states. If this action does not resolve the uncertainty, Q_{MDP} will continue to take it ad infinitum.

The problem with the heuristics discussed so far is that they only act to seek reward, possibly taking into consideration their current uncertainty and the uncertainty of their actions. Unfortunately, they will never act to decrease their future uncertainty. This kind of behaviour can be extremely important for an agent, that is supposed to operate robustly and autonomously.

Probabilistic With Uncertainty Considerations – Action entropy is an example of a probabilistic that can help to *reduce uncertainty* [CKK96]. It switches between two distinct modes: seeking reward and seeking information. Recognising that uncertainty is problematic only when it introduces uncertainty about the appropriate action, action entropy uses the belief–optimality distribution as its switching criterion. When the entropy of this distribution is above a given threshold, and the agent therefore is uncertain about which action to take, the action entropy probabilistic takes the action that will best reduce its belief uncertainty over a one–step horizon (when it is lower , it follows the MDP–based heuristic).

An example is *Coastal navigation*, presented by *Roy* and *Thrun* [RT99]. It plans a fixed path, but then considers the quality of localisation along that path. It begins by calculating the information content of each state, based on the extent to which an observation from that state would modify a fixed path. It then assigns a cost to each state as: a weighted sum of an information–based cost and a goal–related cost. These uncertainty–aware probabilistic can be an improvement over simpler heuristics, but they all have short comings. Firstly, they are unable to make longer–term plans, reasoning about how uncertainty will evolve over the course of a plan. Secondly, they rely on a human designer to decide on the importance of certainty. This is a difficult parameter to specify, especially because it is not constant for a given problem or environment.

In our scenario, uncertainty is not problematic unless it prevents a search agent from achieving its goal, i.e. finding a trapped victim independent of its position in a minimum time. Thus, an agent may sometimes be forced to persist with a high level of uncertainty, in a portion of the belief–space in which uncertainty reducing actions are very expensive in terms of time. In this activity we present a probabilistic approach that it deal with uncertainty only when it passes *an uncertainty threshold*, defined by this author.

Hierarchical Approaches –Hierarchical approaches aim to decrease computational requirements by decomposing a large POMDP into a number of POMDP subsets. The cost of solving the constituent sub–POMDPs can be significantly less than the cost of solving the original. *Theocharous and Mahadevan* [TM02] have proposed a *hierarchical model* for robot navigation in an office environment. A set of abstract states are posited, each of which encapsulates a set of underlying states. A macro–action from an abstract state is equivalent to a set of actions through the underlying concrete state–space. They show that the entropy of beliefs over abstract states is significantly less than the entropy over concrete states. Therefore they assume complete observability of the abstract states, and the application of heuristics based on this assumption, provide a better approximation than for concrete states. While experiments show that planning is simplified by this hierarchical approach, Theocharouss environment is highly structured, consisting of corridors and junctions and resulting in interfaces between abstract states that are tightly constrained. It is unclear how well the approach will generalise to problems that exhibit less structure like those inside our unstructured search fields.

Rather than specifying a hierarchy of states, *Pineau* [PGT03] specifies a *hierarchy of actions*. A set of abstract actions is posited, each of which consists of a number of sub–tasks. This approach is much more applicable to problems involving discrete sets of actions. *Foka* [FT05] specifies a hierarchy of both states and actions for robot navigation problems. The hierarchy of states is reminiscent of a quad–tree decomposition [Sam84]. The discretisation of both states and actions is finer at levels deeper in the hierarchy. Individual sub–POMDPs are solved using an MDP–based heuristic. While extensive results of the computational requirements are presented, the effects on performance are not really clear. The most serious problem limiting the application of hierarchical approaches is the requirement that the hierarchy be specified by a human designer, based on the perceived structure of the particular domain. A method for automating this process would be extremely expensive.

This thesis describes a scalable approach to POMDP planning which uses *low–dimensional representations of the belief space* to deal with agent's uncertainty. In this approach we are applying one–step planning presented by Roy [Roy03]. In this planning process uncertain agent takes advantage of actions that gather required information to reduce its uncertainty (known as low risk actions), even if the reward function (reward function based on the agent's utility) does not make immediate and explicit advantages of gathering such information.

2.4.2 POMDP–Solver

There are different POMDP solver tools available on the web that enable us to solve POMDP problems to some extent. For instance, *Cassandra* has written software [Cas] that solves problems that are formulated as POMDPs. It uses a basic dynamic programming approach for all algorithms, solving one stage at a time while working backwards in time. It is also able to solve finite horizon problems with or without discounting.

For instance, in Chapter 5 we define a simple *24–state pomdp*. In state $s < n >$, an agent wants to perform action $a < n >$, which $a < n >$ is a low risk action. Observation probability is set to 1, i.e. perfect observation. At each step a belief distribution is adjusted for the agent, according to its recorded

information during exploration procedure. This is a belief probability of 24 state cells around the agent, as indicated below.

Table 2.1: Collected belief points, surrounding the agent:

s16	**s15**	**s14**	**s13**	**s12**
s17	**s4**	**s3**	**s2**	**s11**
s18	**s5**	**s0**	**s1**	**s10**
s19	**s6**	**s7**	**s8**	**s9**
s20	**s21**	**s22**	**s23**	**s24**

We define: 1- the states as: {s0, s1, s2, ..., s24}(agent is located in s0, the start point); 2- the actions as: {a0, a1, a2, a3, a4, a5, a6, a7} (respectively are: sw, s, se, e, w, nw, n, ne); 3- the start point as s0; 4- the end points as {s9, s10,...,s24}; 5- the transition probability for taking an action.

Then for each sequence of actions that enable the agent to reach an end point,we estimate its associated risk. Finally the tool is able to predict the lowest risk actions, according to the values that is defined for it before any movement (in Chapter 5 that is selected as the action entropy).

2.5 Exploration Formation

Formation behaviours in nature such as flocking and schooling are able to benefit the animals that use them in various ways. Each animal in a herd, for instance, benefits by minimising its encounters with predators. By grouping, animals also combine their sensors to maximise the chance of detecting predators or to forage for food more efficiently. Studies of flocking and schooling show that these behaviours merge as a combination of a desire to stay in the group and yet simultaneously keep a separation distance from other members of the group. Since groups of artificial agents could similarly benefit from formation controls, robotic researchers and those in the artificial life community have drawn from these biological studies to develop formation control for both simulated agents and robots.

Bahceci et al. [BSS03] presented a review of pattern formation and adaptation strategies in multi robot systems. They divided pattern formation studies into centralised and decentralised pattern formation categories. They claimed that having a central unit assumption in centralised pattern formation might make the approach more costly, less robust to failures and less scalable, especially for a large number of individuals. However, in nature we have flock of wildebeest that are decentralised (since there are large number of wildebeest in a flock, thus the rule of 5% creates a decentralised system) while pack of wolves are centralised (the rule of 5% among small number of wolves in a pack makes the system centralised, it involves a leader).

Formation control may be one of the fundamental problems in the control of multiple mobile robots. Many interesting tasks and behaviours [CFKM95] can be demonstrated with multiple mobile robots that move freely without strict constraints on their relative positions, and there are many important tasks that require robots to closely coordinate their movements . For instance, an aircraft flying in a V–shaped formation could maximise fuel efficiency by taking advantage of aerodynamics similar to the way flocks

of birds use formation flying to increase their efficiency [LS70]. In addition, if formation control is automated, multiple aircraft could be flown by one. Thus a single piloted aircraft could control a large formation of semi–autonomous pilotless aircraft.

Approaches to formation generation in robots may be distinguished by their sensing requirements, their method of behavioural integration, and their communication abilities. In general we can consider four formations for the team of multi agents [ARM]:

1) *Line:* where the agents travel line–abreast, side by side

2) *Column:* where the agents travel one after the other

3) *Coil:* where the agents travel in a coils (e.g. school of fish)

4) *Wedge:* where the agents travel in a *Wedge* (e.g. flock of birds)

Line and Wedge movement formation provide larger line of sight for the multi–agent system, therefore it leads the system towards faster coverage. In these two formations all agents follow one direction. However, following these formations and maintaining the implicit communication among all the agents, inside a cluttered environment, is impossible. They are suitable only for open areas (e.g. high in the sky). In contrary to above formations, Column and Coil are able to maintain their implicit communication. Since agents follow each other, they replace one another. However, during their team movement their coverage is very small (e.g. for column line of sight, for unexplored area, is only one agent). Coil formation (i.e. multi *column* moving in a *line*) is like creating a larger agent from several smaller agents, therefore they may not be able to enter narrow pathways. As we are dealing with narrow pathways inside our search fields, coil, wedge, and line are not practical formations for our confined and cluttered maze–like environments. Therefore the best movement formation in these kinds of environments has been selected as *column*.

Columns have been used for perimeter detection by mobile agents [FDS$^+$02]. This technique has been inspired by pack of wolves and their agents are able to track a dynamic perimeter, e.g. an oil spill on the sea surface. Furthermore Feddema et al. [FLS02] introduced an outdoor perimeter surveillance over an open large area using a swarm of agents that investigate alarms for intrusion detection sensors. *Dorigo and Nouyan* [DN06] implemented column formation behaviour in a sensor–based simulation. The robots had two states: explorer and chain member. In explorer state, the robots search for other chain members or the nest. Whenever a robot finds the nest or a chain member, it tries to keep permanent visual contact with it using an omni–directional camera. The aim of the robot is to create a chain that connects the nest to a prey. Robots can distinguish other chain members according to the colour of their equipped LED ring.

As indicated in figure 6.18 all agents, moving in a formation within their team, have a unique *ID* tagged on them, i.e. a unique number. This enables all the member of team (or even other rescue team operators) to distinguish these mini agents from each other. Furthermore, all agents have a limited line of sight, i.e. indicated as blue circle around agents. Each agent inside the team should have at least one other agent in its line of sight.

In this activity we are applying the column formation to send our agents inside the search field. *This*

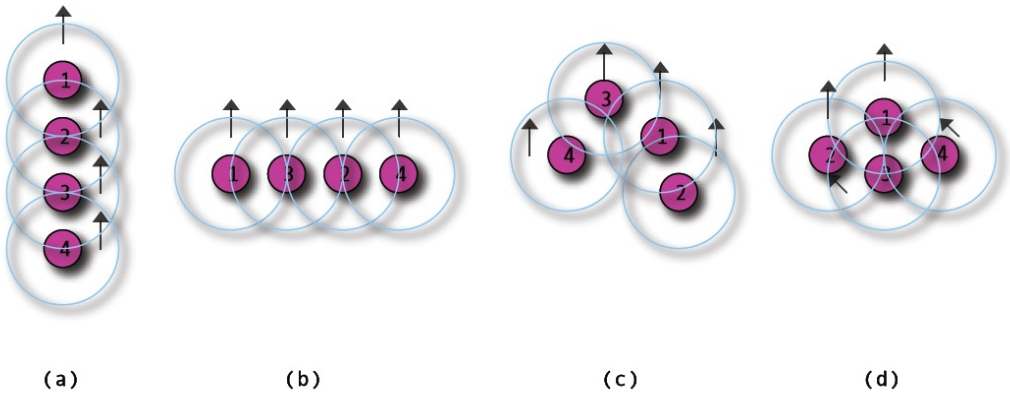

Figure 2.3: Formation for the team of robots: (a) Column (b) Line (c) Wedge (d) Coil

formation enables the team of agents to: 1- avoid any traffic congestion inside cluttered and confined field; 2- divide the search field into smaller areas; 3- dedicate each subspace to one or more agents, which agents will explore them exhaustively.

2.6 Summary

In a disaster scenario the goal of a search and rescue operation is to rescue the greatest number of victims in the shortest amount of time, while minimising the risk to the rescuers. Rescue team should gather essential data and make decisions on the course of action in 'five' phases. They must ascertain what types of structures are involved, the extent of damage, the layout of the building(s) involved, what hazards are present (such as downed power lines, natural gas leaks, flooding, animals, hazardous materials, or a structure susceptible to additional collapse during the rescue).

Techniques to search for potential victims are based on identifying possible locations of victims, or areas of entrapment. Areas of entrapment inside collapsed structures are called voids. There are several types of voids, e.g. the pancake void (multiple floors of a building have collapsed diagonally onto each other). Voids can also include spaces where victims may have entered for self–protection during a disaster, such as under desks or in bathtubs or closets. Inside such a voids victims might find enough space and air to survive. i.e. 'life safe voids'. To identify the location of trapped victims a systematic search pattern with special sensors and equipment is required. For more details of *multi– robot in USAR operations* refer Roher's thesis [Roe08].

We investigate exploration inside pancake collapses with limited sensing mini robots. We will model such a pancake collapses in 2D simulation environments. We introduced the competing exploration techniques, additionally their advantages and disadvantageous. To advance the state of the art in exploration strategies (against the existed algorithms especially frontier–based techniques) our agents record their belief distribution in addition to collected data (e.g. visited cells, and occupied cells). Thus, the agents are able to reduce their uncertainty while performing an efficient exploration, in terms of exploration time. Furthermore, we define a column formation for our team of agents to avoid initial congestion while exploring a search field.

Chapter 3

System Simulation

Autonomous navigation inside unknown cluttered environments is one of the main challenges for search and rescue robots inside collapsed buildings. Being able to compare different search strategies in various search fields is crucial to attain fast victim localisation. Standardisation of the challenges, through the use of simulated search terrains, can provide a meaningful comparison of exploration algorithms. In this regard we are able to send our simulated robots inside these test terrains. They can proceed to assess victim condition, initiate rescue, or attempt localisation (i.e. using odometry readings). We desire an intelligent system that is able to autonomously fulfil its mission requirements. One of our goals, in this thesis, is to evaluate the performance of several multi–robot search strategies inside the structural collapses, for limited sensing mini robots, by reproducing various cluttered test fields. Therefore we discuss an algorithmic development and proliferation of realistic after–disaster test fields for search and rescue simulated robots.

How should we develop *2D* cluttered and confined voids, to test our exploration algorithms, while there are no systematic after–disaster search field generator to develop life safe voids? It seems reasonable, from details described in Chapter 2, to describe them as a mixture of open space voids with maze–like paths providing connectivity of adjacent places (*connected cavities*). Additionally a *discriminative index* is needed to define the complexity of the search fields and grade reference modules through performance metrics. In other words, this enables the researchers to create synthetic fields with which to compare their search strategies.

In entomology, scientists distinguish tortuosity of the insect trails by their fractal dimension [DB98]. This is known as long memory Brownian motion, an example of random process with fractal properties. Although there are several tortuosity indices, the fractal dimension of Brownian motion has been known as a quantitative discriminator with which to characterise tortuosity of the trails in a comparative approach. In this thesis we deploy Random Walk algorithm, which follows a chaotic Brownian movement, to our agents and by their recorded trails we generate various confined search fields.

Our main contribution in this Chapter is the provision of the pre–requisite requirements to set up our research experiments. These are the development of an algorithmic approach to the generation of realistic after disaster test fields (i.e. search field generator) for multi agent search and rescue system. We will also introduce the fractal dimension of agent's movement trails as an index to differentiate simulated

search terrains and define their complexity. This index is tested on various after–disaster search fields, which was developed by our search field generator.

As discussed in Chapter 2, studies [VW05a, Cra09] ascertained that traditional robotic performance measures for path planning were incomplete and inadequate for analysing control and exploration tasks inside complex environments. They therefore introduced a novel analysis approach based on *fractal path tortuosity*. This index shows how well a robot is able to handle the difficulties in an environment, from its movement trails. For instance, the movement trails of the robot inside an environment with a high density of obstacles and closed passages (dead ends) has a larger fractal path tortuosity than that of an open space environment with few obstacles.

3.1 Agents

Our simulated robot should be able to navigate inside dense, cluttered and confined search fields. The natural choice for motion inside such an environment is the *obstacle avoidance methods*. We consider a high level and a low level design for our multi–agent system. The low level design is dealing with robot control system (e.g. navigation and orientation) and localising system, while exploration strategies are considered as the *high level* design. In this activity we mainly focus on the *high level* design and we assume that the lower level design has been completed out by Mechatronic researchers, this has been summarised as a handout by *Mehmet Bodur* [Bod07].

In our framework we have:

- only one type of robot, that is a model of *millibot*. Thus, there is only one size of agent ($100mm$) and it is able to move only one cell in each time step.
- a `SENSOR-SET` that includes simulated Infrared, IMU, VGA camera directed towards a spherical mirror to provide an *omni directional view*, laser barcode device, and NIDR sensors. The user is able to reconfigure the `SENSOR-SET`.
- visual interaction among agents. Therefore, agents are always able to communicate with each other with their equipped LEDs, restricted to a cell in each direction. Thus, they send visual information by their triangular full colour LEDs.
- each agent has a unique ID. We assume small tags mounted on the back of each robot that is readable through laser barcode devices. These tags use strips of $3M$ *Scotchlite retroreflective tape*, arranged in n bit binary patterns. The laser barcode device requires the first and last bits to be 1s (reflective), i.e. 2^{n-2} possible IDs [NPR$^+$03]. Thus, for the m number of robots: $n = \log_2 m + 2$.

Moreover we are able to send different number of robots inside our simulated search fields, from one robot to a team of hundred robots.

3.1.1 Eight Segmented Vision

An IR range finder provides the agent information concerning empty or occupied volumes in the space subtended by the beam (45 degrees cone for the present sensors, which covers adjacent grid cells around

the agent) in front of the sensor, illustrated in figure 3.1. Therefore, the agent performs a 360 degree sensor sweep. It has been decided that each robot should have a view of eight cells in its surrounding area (*Window vision* (Wv)). Cells can be empty or occupied. Mobile agents can only move one cell in any direction, if the chosen cell is empty. They are aware of their direction, e.g. North, South..., with their equipped simulated miniature inertial measurement unit (IMU).

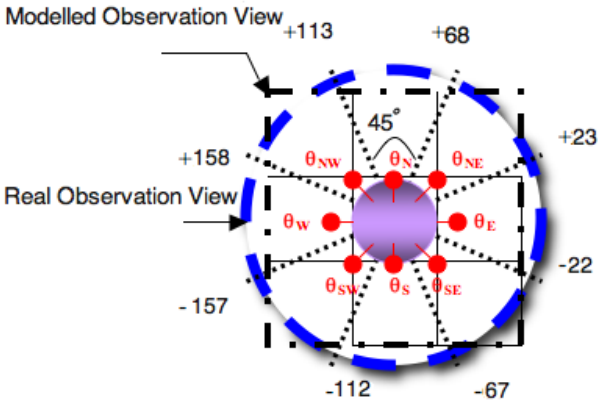

Figure 3.1: Modelled and Real observation view

Victims are assumed to be created as having parts that are detectable (e.g. uncovered head through its temperature) and parts that are undetectable, they are covered. Undetectable parts of victims are classed as obstacles. To distinguish live victims from other warm objects, the agent is equipped with a *NDIR* (non dispersive infrared absorbance) sensor, which is able to recognise humans from their CO_2 signature.

3.1.2 Agent Tasks

There are two main tasks defined for our agents inside the grid cell simulated environment:

Field Generator– They are used to create search fields, and generate *life safe voids* as described by Murphy et al. [Mur04]. For this task agents perform the Brownian random walk (BRW), looking for victim(s). Their trails across the field are recorded and defined as the pathways while unexplored cells subsequently are selected as obstacles.

Field Explorer– They are used to evaluate the performance of search robots modelled inside a simulation environment of various *exploration algorithms*.

3.1.3 Brownian Random Walk

The brownian random walk algorithm [SK04] is selected to generate simulation environments, test the developed fields to introduce a discriminative index and also compare its performance against other competing exploration algorithms. When agents are released at the entrance of the confined simulation search field, they immediately start their random walk, inspired by "Brownian movement" [ML05].

We define the Brownian motion of our random walk , on a regular lattice, as follows: Suppose that the agent executes a random walk by taking a time step on a *d–dimensional lattice*, agent starts at time $t = 0$ from an arbitrary point of the lattice. The numbers of possible orientations are, $\phi = 3$ for $1D$ chain

(i.e. North, South, and Idle) and $\phi = 5$ for $2D$ square lattice (i.e. grid cell). Our agent walks in each direction with an equal probability, the probability of choosing any given walking direction is $P = \frac{1}{\phi}$ at each time step.

These agents follow a straight line of random length headed to some initial random direction. When they reach the end of this segment, a new direction is selected. They can select one direction out of these four directions $\{ \theta_E, \theta_N, \theta_W, \theta_S \}$. *Svennebring* and *Koenig* [SK04] eliminated all the intermediate directions, such as NE, to have equal steps. Agent selects one of the sensor directions randomly if it reads empty space.

There are boundaries defined to limit the environment and set a bounded field for agents to navigate within. Agents should change direction every time they meet the borders or occupied cells. This is also known as the end of their initial segment. However there is a selected time limit for all agents. Every time an agent reaches its time limit it chooses a random direction and resets the ω. These time segment lengths (time limit i.e. ω) are derived, through the exploration procedure, from equation 3.1.

The generation of Brownian random walks, requires random numbers that fall into a Gaussian distribution. The *Box–Muller* [BM58] method is used to generate a pair of Gaussian random numbers (GRN) from a pair of uniform numbers. The generation of Brownian random walks, requires random numbers that fall into a Gaussian distribution (e.g. a bell curve).

$$\omega = |\frac{1}{GRN} \times \Delta t| seconds \tag{3.1}$$

The time delay and therefore the time step, is here assumed to be $\Delta t = 36$ seconds- corresponding to that of *Millibot*.

Figure 3.2(1-3) illustrates agent performing Brownian random walk to explore a search field. It shows agent selecting another direction to avoid obstacle collision. While, 3.2(4), indicates the time limit, whenever the agent meets this limit, it should select another direction to follow.

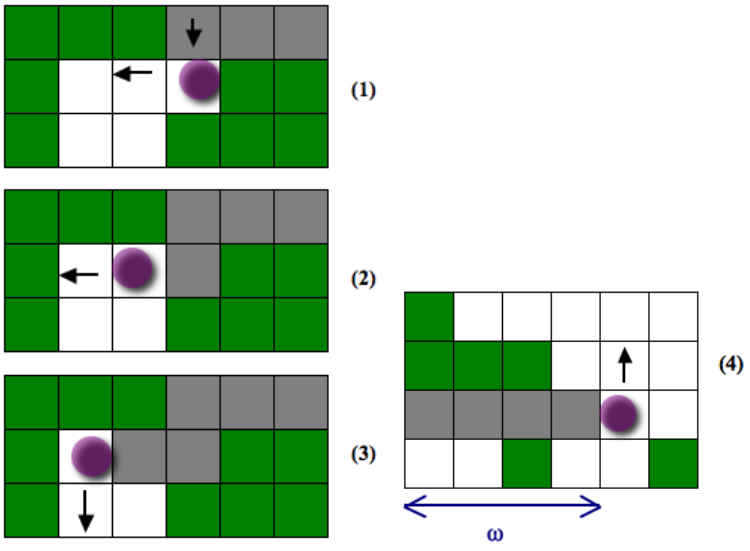

Figure 3.2: ANT behaviour inside our generated search field

3.2 Simulated 2D Search Field

The main goal of generating any test bed is to facilitate the trial and evaluation of real world experiments. As discussed before inside structural collapses there are confined voids connected through narrow pathways [JME01]. These life safe voids are the most demanding locations for exploration robots. Search robots should explore the entire maze of obstacles and debris in search of victims and the appropriate exploration technique must ensure that the overall victim discovery time is minimised.

Due to the limitation of mini agents, especially in their functionality, they are deployed by their mother wherever it is required to perform an exhaustive search operation. Thus in our scenario there is a mother robot [MAB$^+$99] sitting at the start point, i.e. S $= (0, 0)$, and sending in smaller robots (we assumed larger robots discovered these narrow voids), one by one inside a pancake collapse.

By performing random brownian walk, our agents can generate various $2D$ simulated environments. The generated simulation environment is divided into several equal square cells, length L. In all our experiments, in this activity, we assume that the grid size is $L =$ *100 mm*. There are four *states* available for our square cells inside the simulation search field.

1- Occupied by obstacle or undetectable victim's body part

2- Occupied by victim's detectable part i.e. only four adjacent cells, creating a square.

3- Occupied by other agent(s)

4- Empty

The first three states make the cell inaccessible to the agents. We assumed that each agent occupies only one cell every time it moves. Any cell that is explored by our agent will be indicated on the simulation field. Therefore empty and unexplored cells are white, while explored cells are coloured grey. This feature demonstrates the time sliced evaluation of agents behaviour, e.g. how well the search strategy is able to spread its agents inside a search field.

3.2.1 Search Field Generator

To generate various confined and cluttered 2D simulation environments we follow an algorithmic approach [SS09a], algorithm 1, which is precisely defined as follows:

i- a victim(s) is located randomly inside an empty field; *ii*- a restricted entrance to enter the empty field is selected; *iii*- an agent(s) is sent inside, from the restricted entrance, the field to perform random walk and locate a victim(s); *iv*- All the recorded trails taken by the agent(s) will be considered as pathways and untouched free cells will become obstacles. In this case we may have multi paths, with only one leading to our victim. Therefore, by running our random walk algorithm several times we are able to generate various search paths inside our empty field. We are also able to change the number of victims and robots to generate even more simulation environments. The size of the search field (frame size) can be increased up to 100×100 grid cells.

One of the advantages of such a search field generator is that it creates an accessible space (i.e. all free cells are reachable), with a complexity that is rigidly defined through its fractal dimension (Discussed in Chapter 2). Fractional Brownian motion is produced when the random steps taken by an agent are correlated in equally spaced spatial. Since the search fields are generated through these motions, we

investigate the relationship between the fractal dimension of the search fields and their complexity.

Algorithm 1 *Search Field Generation*

Input: *(x,y):Search field frame size; v:No. of victims; a:No. of agents;*

Output: *S:Generated Search field;*

1: *Place v number of victims randomly inside an empty search field (x,y)*
2: *Select a narrow restricted entrance in the corner (start point (0,0)).*
3: *Send in the (a) :agent(s) through the narrow entrance*
4: **while** *no victim is located, by the agent,* **do**
5: *Perform Brownian random walk*
6: *Record explored cells as pathways $(1s)$ & untouched cells as obstacles $(0s)$ inside S.*
7: **end while**
8: **return** S;

Figure 3.3 shows an overhead of an example environment generated by our search field generator. This is a 33×33 frame size, with a single victim in a middle (to generate such a field we have sent in two agents to discover a victim, thus the pathways are the footprints of the two agents). The red single grid cell is our agent that is sent inside the generated search field, to explore the field (i.e. *Field explorer*).

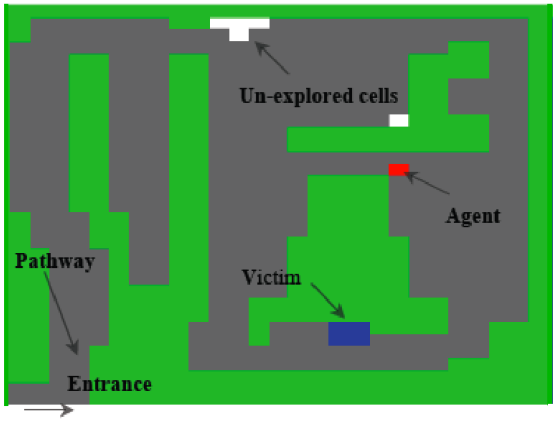

Figure 3.3: An image of a generated simulation search field

3.2.2 Discriminative Complexity Index

In literature, self–affine fractal dimensions have been used in various applications to indicate fractal complexity. Every fractal has a numeric fractal dimension that can be used to indicate fractal figure complexity. There are several methods to calculate fractal dimension. This method has been used in various applications such as graphic image processing [Jon87]. The Box counting technique superimposes a regular *box of length* γ on the object and counts *the number of occupied cells* (n_i) in every type of cells. We follow the power law relationship defined by Voss et al. [Vos88] to calculate the box–counting dimension of our generated search field. The Power–law relationship is derived as:

$$n_i = K\gamma^{-D} \tag{3.2}$$

Where D *is the box–counting dimension* and K *is the total number of grid cells* on our field of all states:

$$K = \sum_{i=1}^{N} n_i(\gamma) \tag{3.3}$$

We have N states of cells where n_i is the number of individuals belonging to the i_{th} types of cells ($i \in I = \{1, 2, ..., N\}$) with the size of γ. Here we consider only two types of cells (occupied and free, $N = 2$), and to estimate the box–counting dimension of obstacles on the search field, we calculate the n_{oc} (number of occupied cells). Also we estimate the box–counting dimension of pathways (path tortuosity). n_{path} is the number of free cells inside our search field that are used as pathways for the exploration agents.

In the box counting method a non–overlapping scanning applies boxes to an image (boxes are superimposed over an image) by counting the number of boxes required to cover the whole image it determines its box–counting dimension. We should adjust the minimum and maximum size of the boxes (i.e. the length of the square box) in advance. The minimum size can be selected as one grid cell and the maximum size can be adjusted to: max size $= 40\% \times$ (image size), where image size is the boundary containing occupied cells. In this technique the most appropriate size for the boxes in the box counting technique depends strongly on the complexity of an image. For instance, the more complex an image is the smaller is the size of its selected box. The most efficient cover is generated from box counting data over multiple grid positions using the least number of boxes that is required for covering an image. For each size, only the box count that was most efficient (i.e. the smallest number of boxes) is selected from the entire grid position collections, as indicated by an example in figure 3.4.

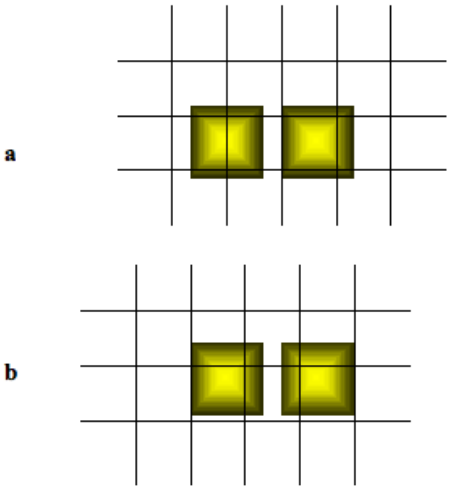

Figure 3.4: grid position in **(a)** requires 12 boxes to cover the image, while in **(b)** requires 6 boxes

From equation 3.4 we are able to calculate the box–counting dimension as:

$$D = \frac{\log\ K - \log n_i(\gamma)}{\log \gamma} = \frac{log(\frac{K}{n_i(\gamma)})}{\log \gamma} \tag{3.4}$$

D is a metric dimension, therefore its definition depends on metric scaling properties. Although the number estimated by the box–counting dimension differs from fractal dimension, this difference in average is less than 0.02, we will refer to this value as the estimated *fractal dimension* (FD) all through this thesis.

Figure 3.5 illustrates the correlation between victim discovery time and the fractal dimension. 101 simulated search fields (generated by our search field generator) have been chosen to determine the most suitable index for search terrain complexity. These fields are differentiated first by the FDO (Fractal Dimension of Obstacles) and then by the FDP (Fractal Dimension of Pathways).

The default values in our experiments are: a frame size of 33×33 square grid cells, we vary the environment according to their fractal dimension. The number of the agents in this experiment was set as the best performance for random walk algorithm. Thus we selected 3 agents to run the experiment. Search agents are sent inside a generated void, from its restricted entrance path, to explore the unknown search environment, with its various branches of different lengths. The average time steps taken by a single agent to locate the first trapped victim in each field, is recorded as *'victim discovery time'*. This is our *primary fitness metric* applied to select a discriminative index for our generated search fields. Each time step in our graphs (figure 3.5) is the average of running the random walk algorithm 100 times. It indicates that:

- fractal dimension of the obstacles has a negative correlation to time steps (with some fluctuations), in other words to search field complexity.

- on the other hand there is a positive correlation between fractal dimensions of the pathways (FDP) and search field complexity. The larger the FDP, the larger is the victim discovery time. Therefore, the more complex is our generated search terrain.

From the results of this experiment we were convinced to use the fractal dimension of path tortuosity as the complexity index in our search field generator. This index is able to differentiate between random developed environments according to their path complexity, because they were distributed using Brownian motion. Therefore, all users are able to produce various fields by our algorithmic approach and save the image as a binary image (zero for obstacles and 1 for pathways), scan it and use *FracLac/J image* to estimate the fractal dimension of the generated search fields.

The fractal dimension of pathways has no correlation with the number of agents (Field generators), nor with the number of victims (i.e. located inside an empty search field) and the search field frame size. For instance by applying *an agent*, *two victims*, and a frame size of 33×33 we mange to generate five environments with the complexities as follows: $FDP = 2.45$, $FDP = 2.49$, $FDP = 2.53$, $FDP = 2.58$, and $FDP = 2.62$. Furthermore, we applied *an agent*, and *two victims* inside a frame size of 50×50 and we manage to generate five different environments as follows: $FDP = 2.51$, $FDP = 2.55$, $FDP = 2.58$, $FDP = 2.60$, and $FDP = 2.64$. We also generated a search field

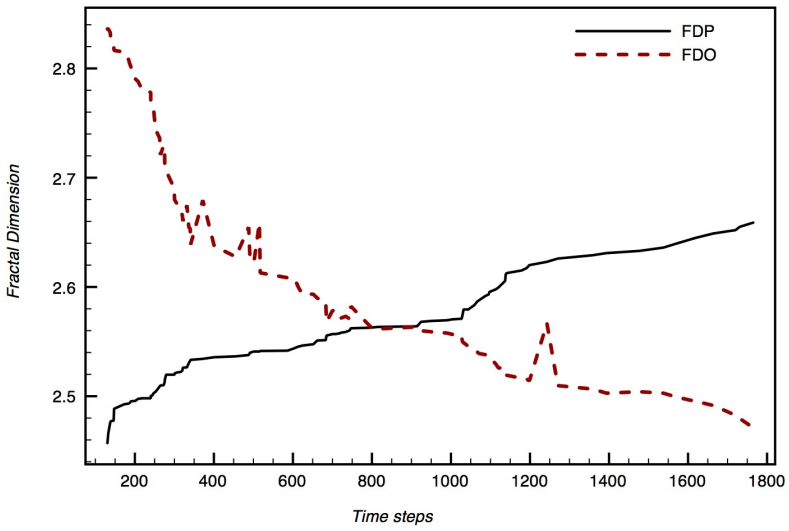

Figure 3.5: Fractal dimension of obstacles and pathways (one time step=36 sec.)

with $FDP = 2.58$, which was generated by *three agents* and two victims inside a 50×50 frame size. However, no matter how these fields are generated, FDP is highly correlated to the victim discovery time.

We set a maximum number for the trapped victims (Max_{vic}) inside an empty search field according to their frame size: i-frame size of $33 \times 33, Max_{vic} = 3$; ii- frame size of $50 \times 50, Max_{vic} = 4$; ii-frame size of $100 \times 100, Max_{vic} = 7$. These settings enable our map generator to generate more realistic pancake collapses.

3.2.3 Probability Mass Function

The probability mass function (pmf) for a discrete random variable X is a function ($f_X(x)$) that assigns probability to each value of X This function shows how much probability, or mass is given to each value of the random variables. The total mass for a probability distribution is equal to one. The notation used to denote the probability mass function for X is $P(X)$. In general, by applying the probability mass function we are able to calculate the long–term average outcome of a random variable, which is called the expected value.

The figure 3.6 shows the discrete time steps of an agent inside three search fields, different fractal dimensions as: ($FDP0 = 2.658 < FDP = 12.581 < FDP = 22.494$) The probability of visiting (not revisiting) new cells, for every 100 time steps, average over 100 run, is known as $P(X(t))$.

$$f_{X(t)}(x) \; : \begin{cases} P(X(t)) = x, & \text{if } t \in T_i \\ P(X(t)) = 0, & \text{if } t \notin T_i \end{cases} \tag{3.5}$$

Where, $T = \{a, b, c, d, ..., q\}$: $a = \{0, ..., 100\}, b = \{101, ..., 200\}, ..., q = \{1601, ..., 1700\}$.

a is the first time slot of 100 time steps, b is the second 100 time steps,... . In this experiment we have considered the total of 17 time slots of 100 time steps (in this experiment 1685 is the maximum time steps taken by the agents).

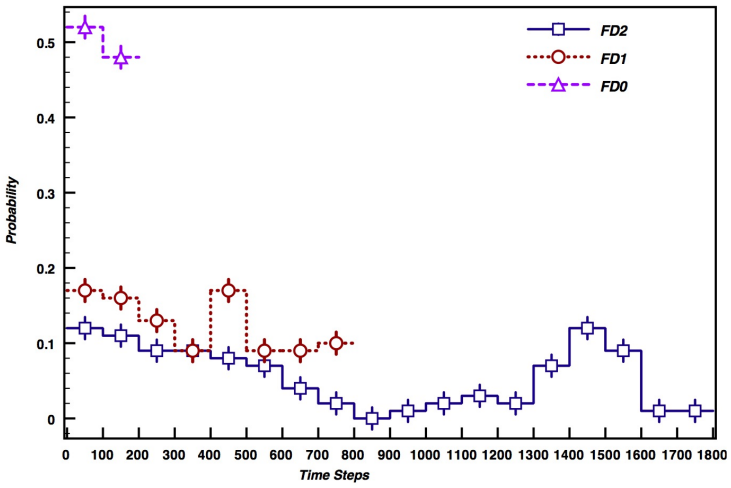

Figure 3.6: Probability mass function for three different search fields, $FDP0 < FDP1 < FDP2$

For instance to estimate the pmf of the agent time steps, to discover a single victim, we have: if $t1 \in a$ then we assign $P(X(t1)) = x1$, where $P(\{ti \in T; X(ti)\}) = \{x1, ..., x17\}$.

- For $FD0$, we have $T_0 = \{a, b\}$, the $c - q$ are empty, e.g. $c = \varnothing$.
- For $FD1$, we have $T_1 = \{a, b, ..., h\}$, the $i - q$ are empty, e.g. $m = \varnothing$.
- For $FD2$, we have $T_2 = \{a, b, ..., q\}$.

We apply *pmf function*(equuation: 3.5) to the search fields as illustrated in figure 3.6. The average probability (expected value) of visiting new cells for each field with respect to its fractal dimension are: $0.05, 0.12, 0.5$. To sum up, the larger the fractal dimension of a search field the smaller is the number of new cells that are visited in an interval of 100 time steps.

3.3 Complexity Index Validation

In figure 3.8 we compare *TREE* and *ANT* exploration Algorithm to the random walk algorithm for a range of fractal dimensions of pathways (i.e. 101 search fields). In this experiment we are investigating the positive correlation between FDP and the victim discovery time for other well known exploration algorithms. Through this experiment we validate our complexity index, FDP, and we apply it to differentiate our generated search fields all through this thesis.

ANT– We define our ANT algorithm [KL01] as an agent that keeps count of the number of visits of each cell and moves to the least visited free cell among the 8 cells that surrounds it. All the visited cells are recorded in an occupancy grid memory, and for each cell inside this memory (Nr) we have allocated a counter, $C[Nr]$. The number of steps taken by an agent to discover a single victim is recorded as Ns.

TREE– The TREE algorithm, is an implementation of the on–line STC (Spanning Tree Covering) algorithm by Gabriely and Rimon [MT04]. In this algorithm agents run a Depth–first Search (DFS) algorithm during the incremental spanning tree construction. We fix the size of sweep covering width to the length of our grid cell, i.e. $100mm$.

Figure 3.7 indicates the spanning tree. An agent scans its 4 neighbours (free cells) and compare

them to its recorded data to differentiate visited cells from un–visited cells. Visited cells (old cells) are those recorded in the agent's memory (e.g. odometry records). In Chapter 5 we will introduce occupancy grid memory to store cell data. During each step, the agent selects the first free cell (CCW) that is not visited before, i.e. y in the algorithm 3. If all the neighbours were visited , $y = NULL$, it returns to its parent and rechecks the neighbours (i.e. function *A.goback-parent()*), 'flagparent' are reset to one. If the parent cell did not scan any new cell, it selects a random free cell around it. It moves forward until it reaches an un–visited cell and then sets its 'flagparent' to 0 again. In environments like our maze–like search fields, the agent must revisit some cells to escape a dead end and reach to an area with un–visited cells.

All the freshly expanded nodes are added to an exploration stack array, through which we are able to guide the agent back to its parent cell.

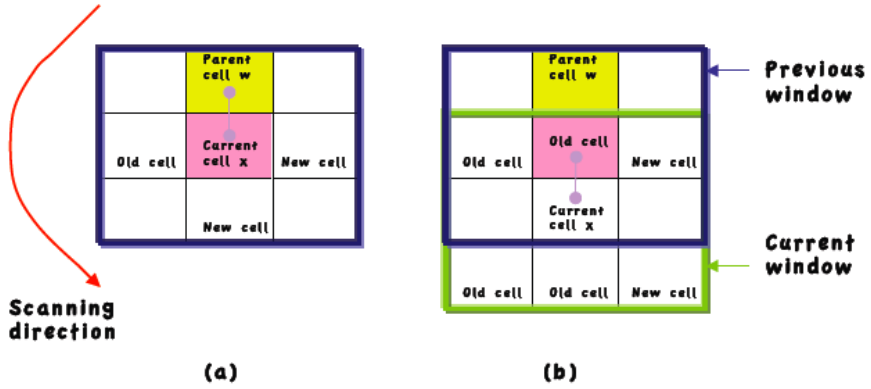

Figure 3.7: (a) Counter clockwise scanning the neighbours (b) selected new cell is our current cell

Algorithm 2 *performANT*

Input: Ns; Nr; *C[Nr]; ogm:occupancy grid memory; x: current cell;*

Output: Ns; Nr; $C[Nr]$; *ogm:occupancy grid memory;*

1: $Ns = 0$; *MIN=2;* $Nr = 3$; $i = 0$;
2: **while** *true* **do**
3: *scan all four neighbour cells around the x(cell)*
4: **for** *each empty cell neighbour* **do**
5: **if** $C[Nr] = 0$ *and* $i = 0$ **then**
6: *Fill the ogm;*
7: $Nr++$;
8: $C[i]++$;
9: $i = Nr$;
10: **else if** $MIN > C[Nr]$ **then**
11: $MIN = C[Nr]$;
12: $i = Nr$;
13: $C[i]++$;
14: **end if**
15: **end for**
16: $Ns++$;
17: $i = 0; MIN = 2$;
18: **return** Ns, $C[i]$, Nr;
19: **end while**

Algorithm 3 *performTREE*

Input: *A:agent;* x: current cell; stack[ ]:stored cells to parent.

Output: x: current cell; $parent\_flag$; $parent\_cell$; Ns.

$parent\_flag = 0$; $parent\_cell : (0, 0)$; $Ns = 0$
while *true* **do**
  scan all four neighbour cells around the x *cell*
  $w = parent\_cell$
  if $parent\_flag == 0$ **then**
    select first CCW un–visited free cell $y = neigbourcell(x)$
    if $y \neq NULL$ **then**
      $stack = x$
      $x = y$
    else
      $parent\_flag = 1$
      A.goback_parent()
    end if
  else
    if $x = w$ **then**
      select first CCW un–visited free cell $y = neigbourcell(x)$
      if $y \neq NULL$ **then**
        $stack = x$
        $x = y$
        $parent\_flag = 0$
      else
        select randomly a free cell $x' = neigbourcell(x)$
        $stack = x$
        $parent\_cell = x'$
        $x = x'$
      end if
    else
      A.goback_parent()
    end if
  end if
  return x, $parent\_flag$, $parent\_cell$, Ns;
end while

Algorithm 4 *performBrownian Random Walk*

Input: *A:agent;* θ*:agent movement angle;*

Output: *p:agent_next_position;* ω*:time limit;* θ*:agent movement angle;* Ns*;*

withdraw an ω *[equ: 3.1]*;
$Ns = 0$;
A:$pos\_start(0, 0)$;
while *true* **do**
  if $\omega \neq 0$ **AND** cell.θ ='FREE' **then**
    ω- -;
  else
    withdraw an ω *[equ: 3.1]*
    select a free cell randomly, direction α
    $\theta = \alpha$
  end if
  according to θ*: select p=A.next_position();*
  return *p,* θ*,* ω*,* Ns
end while

The default values in this experiment are the same as in section 3.2.2 . *Three agents* are sent in to discover a *single trapped victim*. In contrast to fractal dimension, which is independent of trapped victim positioning, discovery time for all algorithms (except the random walk algorithm) is highly sensitive to the position of trapped victims. Thus, for each field we perform 20 experiments with the trapped victim (occupying 4 adjacent cells) placed in different, randomly selected, positions inside the fields. Thus, we perform the TREE and ANT algorithms $20 \times 100 = 2000$ times for each field. We have implemented an omniscient oracle algorithm that knows the true status of all states, even the goal states. It is therefore able to determine the optimal path to the goal.

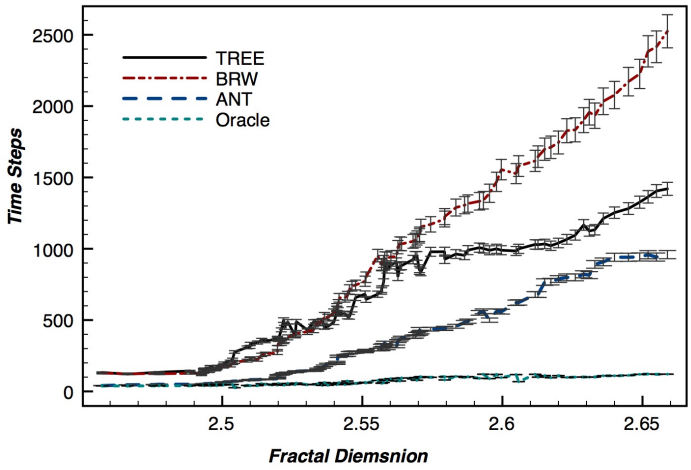

Figure 3.8: Tree, ANT, Brownian Random Walk (BRW), and Oracle algorithms comparison

As expected, agents using the random walk algorithm, that was used to generate the environment, repeat their behaviour. However, agents using the ANT algorithm are able to reduce the number of revisits they make to cells. This reduction is largest in more complex search fields, $FDP > 2.55$. Each data point shown for the ANT and TREE algorithm in figure 3.8 represents an average over 2000 experiments.

The larger is the fractal dimension of pathways (FDP), the more complex is the generated search terrain. Thus random walk agents, with high redundancy behaviour, spend a much longer time to discover a victim. For instance, from our additional test results, for a 10000 grid cell frame size with complexity of $FDP = 2.68$ to test the BRW algorithm, running it for 100 times for each search fields (100 search fields), has a run time of more than a day which makes this test inefficient in terms of exploration time. Fortunately we are able to predict the time steps of the Brownian motion (discussed in chapter 4). This feature allows us to avoid running this time consuming random algorithm and ease our comparing process.

As discussed in Chapter 2 TREE algorithm has the tendency to revisit cells, e.g. going back to its parent cell, and agents running the ANT algorithm cannot determine when the exploration task is completed. This is problematic when applying on a search field with unknown number of trapped victims. The ANT agents are never aware when they have covered the complete search field. Additionally both

of these algorithms do not take transition errors into account and this will have negative effect in the discovery time. Later in this thesis, we introduce a predefined occupancy grid memory for all our search agents. This enables the agents to be aware of their full coverage while performing their exploration strategies. We assume that the size of the grid memory is estimated from the outside of the pancake collapses, by the mother agent.

3.4 Discussion

The introduction to intelligent algorithms and cooperative behaviour for multi–robot systems enables the researchers to improve the performance of their systems significantly. However, the subject is in need of formal methods that will enable researchers to make objective comparisons between approaches and evaluate their performance in a controlled environment. Above we have suggested an approach that can lead to a most rigid classification of a wider range of search areas that introduced by the NIST courses. We also demonstrated a formal approach that can classified generated search fields by a discriminative index. In this regard the fractal dimension of pathways inside a generated search field has a positive correlation to the goal discovery time, and we select this as the search field complexity index.

Our approach is not limited to two dimensional search fields. It is possible to use a similar procedure to work on search fields composed of cubic cells. We can here apply "shape from shading"(SFS) technique [TS94] to transform $3D$ into $2D$. The "Fraclac" tool can then easily be applied to the $2D$ search field as discussed in this chapter. The complexity will now be defined by two indices: FDP and surface depth. SFS aims to derive a $3D$ scene description from a single monocular image. They recovered can be expressed in various ways including surface normal $N = (x, y, z)^T$ and depth $Z_{x,y}$. A surface normal is a unit vector perpendicular to the tangent plane at a point on the surface while depth can be seen as the relative height of the surface from the xy–plane.

Chapter 4

Performance Metric Analysis

As mentioned previously, performance is a vital aspect of disaster exploration and it is essential for researchers in the field to have benchmarks that allow them to compare the quality of their designs and strategies against those of other developers. Benchmarks provide an opportunity to perform controlled experiments under well defined conditions that can be produced with results that can be evaluated through measurements. When they report their results, researchers must therefore provide detailed information about their experiments regarding the types of scenarios, hardware requirements and underlying software libraries. In the literature, there are a great number of autonomous exploration methods [CL04, BMSS05] for multi agent systems and benchmarks that allow developers to compare their results, and to evaluate which approaches are superior under which circumstances. According to *Dillmann* [Dil04] there are three essential aspects of benchmarks for AI (Artificial Intelligence) and robotic research:

i- (*Task*) the agent has to perform a mission, e.g. a specific goal to achieve.

ii- (*Standard*) the benchmark has to be accepted by experts in the field.

iii- (*Precise definition*) the task should be described exactly, for instance:

the execution environment, performance metric, and limiting constraints.

Here we are aiming to further develop, respecting the above aspects, systematic benchmarking for the exploration mini–agents that will allow results to better reported, assessed and compared. Benchmarking relies on the evaluation and comparison of selected measurement based performance metrics. Without that, it is only possible to decide whether or not a given system is able to perform a mission (i.e. robustness). The use of a single measure is rarely possible. To evaluate and compare the performance of all the aspects of exploration algorithms, we have therefore developed a set of metrics that allow comparison between both homogeneous agents using a range of algorithms.

We divide the performance metrics into two distinct groups, Off–line and On–line. Off–line performance metrics are estimated at the end of the task. They may include aspects such as speed (how fast they can explore?) and success (is the desired result achieved?) and will allow the researchers to make an overall evaluation of how well their algorithms perform. They enable us to select optimum search strategies for the various search fields and are essential for evaluating the performance of a system against one or more competing algorithms. This makes it possible to provide an experimental validation of the effectiveness of a new algorithm. It will also allow researchers to optimise their algorithms. *On–line metrics*

on the other hand are used by the agent during its task to estimate its performance against expectations and thus assist it to achieve its goal. For this purpose, we here introduce *effectiveness of coverage*, a novel performance metric that can be assessed by the agent during each time step to estimate the expected action utility. We will evaluate whether the performance of the exploration procedure improves if a *reward function* is used to reward it according to its effectiveness of coverage during each time step.

In this chapter we will also evaluate the *power spectral density* of each search field from its fractal dimension, and from this value we are able to predict the BRW algorithm behaviour. This predicted time series dataset is used as the *BRW algorithm benchmark*, since BRW algorithm [ML05] is one of the most popular random exploration techniques applied on search agents. This search technique is highly reliable and robust and has the widest applicability to different types of environments among the exploration procedures. However, its performance is poor because it is repetitive and redundant and more effective and robust search strategies are therefore urgently required.

4.1 Preliminary Remarks about Metric

The main purpose of the USAR simulation competition is to provide emergency decision support by integrating disaster information, prediction, planning, and human interfaces. The following presents some of the ideas derived from past RoboCup competitions [Bal06]:

- Modify victim discovery requirements to include not only discovery, but also that a team provides a data sheet for each discovered victim. In order to receive full score, this sheet would need to include a path leading to the victim, information about the victim's status, and information about any hazards that exist along the route to the victim.

- Exploration may be based on the amount of areas that a robot clears. Here 'clearing a given area' means that all victims in the area have been located.

- Penalties will be assigned for failing to discover victims and reporting an area as clear that has victims or hazards.

The Virtual Robot competition at *RoboCup* creates an excellent forum in which to discuss performance metrics for search and rescue robotic systems. These metrics are defined in terms of mapping, victim discovery, and exploration time based on the completely explored area. However, detailed mapping of the disaster area at the speeds that are required is impossible , especially from the current search and rescue robot technology. The ultimate goal for search and rescue robots is to explore the entire disaster area and make certain that the area is clear from any trapped victim.

4.2 Multi–Agent Exploration System Taxonomy

As indicated in figure 4.1 we divide the task of multi–agent exploration system in two categories: Individual–level and team–level. In individual–level tasks, an agent should be able to fill its occupancy grid memory with its collected data. It should also be able to return to its base station (mother robot), to report back its findings, i.e. the generated memory from its microscopic surveying. All these tasks

are considered as agent's individual higher level behaviours, while in its lower level an individual agent should be able to avoid any collision with both, other agents and obstacles.

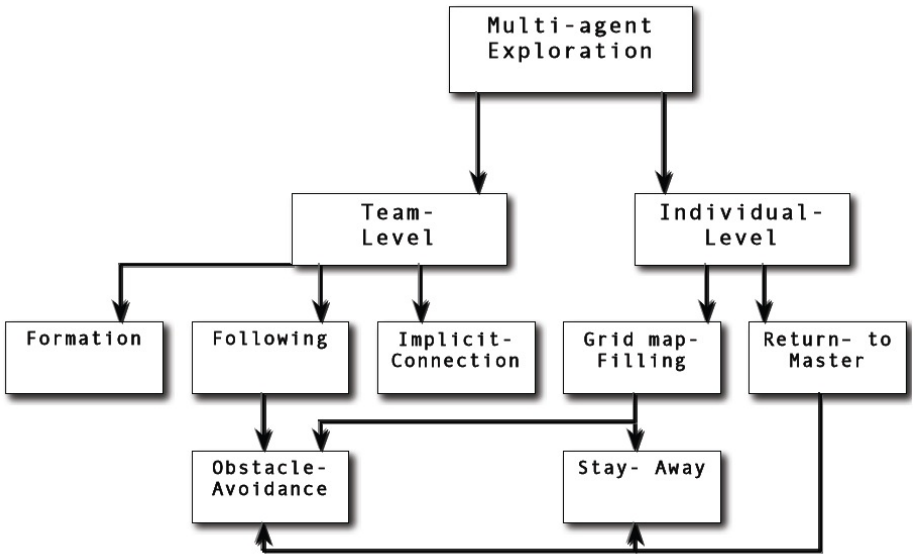

Figure 4.1: Taxonomy of our multi–agent exploration system

For team–level tasks, team of agents should be able to perform a formation (enable them to spread much easier and more thorough, while avoiding the initial congestion). To maintain the formation while moving, they should be able to communicate with each other. Additionally, lower level behaviours, such as obstacle avoidance while following the team movement is essential. In this defined 'team work', agents coordinate their activities with each other to satisfy group goals. They decompose the full coverage of the search field among themselves and then distribute this data. This approach can reduce the complexity of agents task while smaller sub–tasks require less capable agents and fewer resources.

We investigate our team exploration technique in Chapter 6, and our novel individual exploration algorithm will be presented in Chapter 5.

4.2.1 Team–Level Metrics

In order to evaluate and compare the performance of a team of robot, we will develop a set of metrics. Well defined metrics allow comparison between robot teams with different compositions, and algorithms. The metrics introduced by *Balch and Parker* [BP02] to characterise the performance of a real robot team are:

The time taken to perform the mission, the area explored per unit of time (coverage time), the dimensional accuracy, and the power consumption. The dimensional accuracy is obtained by comparing the distance between mapped features (e.g. two walls) with the actual distance. The energy consumption normally is measured by comparing the battery charge for each robot before and after the experiment.

We have applied some of these metrics to compare various algorithms. For instance, in Chapter 6, we use the 'Coverage Rate' metric to evaluate and compare the performance of various teams of our simulated mini robots. Our goal in this activity is to enable the group of mini agents to estimate var-

ious metrics on–line through the group exploration and perform urgent behaviours whenever it deems necessary.

In order to quantify the results for multi agent system, we apply two additional metrics [XGCS07](i.e. off–line metric) to indicate how well an algorithm is able to spread its agents inside a search field (see in Chapter 6), they are:

(i)- Exploration Area percentage p_{EA}

$$p_{EA} = \frac{A_{Ac}}{A} \times 100 \tag{4.1}$$

Where, A_{Ac} is the area covered by the search team and A is the total area of the search field. However, here we will approximate A_{Ac} by the number of visited cells by the team of agents and set $A = N_f$ which is the total number of empty cells available inside the search field.

(ii)- Exploration Efficiency percentage Q_{EE}

$$Q_{EE} = \frac{A_{Ac}}{\sum_{i=1}^{tna} d_i} \times 100 \tag{4.2}$$

Here we define d_i as the number of steps taken by each agent.

Metrics, to some extent, are determined by the scenario in which agents are operating. Choosing the right parameters requires understanding and careful analysis of the experimental setup. The criteria are very dependent on the attributes of the robots and their environment. Experiments where the robots operate on an empty soccer field are very different from when they are in a crowded train station. Fortunately, most experimental work involves measuring the effects of a small number of limited modifications to the algorithms or parameter set while most of the experimental setup and infrastructure remains unchanged.

4.2.2 Individual–Level Metrics

In this activity we are deploying both off–line and on–line metrics to measure the performance of the agents as follows:

– Off–line metric: *i*- Victim discovery time; *ii*- Full coverage time.

– On–line metric: *Expected Utility*.

The expected utility provides an agent, at each step, to act efficiently in terms of minimising the exploration redundancy. We investigate two expected utility for each agent movement : *i*- without transition errors *ii*- with transition errors.

4.3 Expected Utility

The exploration performance is normally determined by a set of information metrics, each reflecting one of the goals of the exploration task in an experiment. Therefore, as these exploration tasks become more complex it may be necessary to increasing the number of information metrics. This is especially true for on–line metrics. In order to perform a successful and efficient search, it is often desirable to obtain advise from multiple metrics simultaneously during exploration. In Chapter 5 we introduce a novel exploration technique that attempts to improve the efficiency of the explorer decision making by basing it on an on–line metric derived from information collected on the fly. In this regard, the ideal method is

that the explorer knows the relative importance of its information metric and weight them appropriately when making decisions about where and how to look. More important in our scenario, the notion of *the expected utility of an action* provides an agent with a method for deciding how to act in the face of uncertainty.

In this activity agents are able to assign utility (or a value) to the situation in which they find themselves and therefore, to the actions that they can perform. Given a general ability we are able to order situations, and actions with definite probabilities that will yield to particular utilities. Imagine a utility function as $U(O_i|A)$, *where O_i is the possible outcomes, and A is a particular action*. We also define a probability for each such outcome, as $Pr(O_i|A)$, then we can compute the *expected action utility* for action A.

Our agent expected action utility is defined below as the *effectiveness of coverage.* In this chapter we only investigate a *SISO* system, single input and single output. Therefore, for taking each action (movement) we have only one possible outcome $(i = 1)$. Later in Chapter 5, by considering transition errors, we will further investigate 3 possible outcomes of an action.

4.3.1 Effectiveness of Coverage

To explore an unknown environment, we make use of the location of a robot and the history of the observations it accumulates inside a search field as it performs a search task. In this section we use synthesised environments to evaluate the efficiency of various search techniques. To be effective a search strategy should divert robots towards unobserved areas but during these efforts some areas may be revisited frequently while others remain unexplored. So to measure the efficiency of each movement we will introduce an 'effectiveness of coverage', our novel on–line metric.

As mentioned before, we use an agent with a 360 degree view, divided into 8 segments. As shown in figure 3.1(Chapter 3), this view is mapped onto the eight cells that surround the agent. The agent is aware that it is located in the centre of this field, and of state (empty or occupied) of each of the surrounding cells. Every time the agent moves, its window vision shifts in each ordinal direction as well. A victim is only observable when part of it occupies one of the eight cells.

Table 4.1: *Effectiveness of Coverage* parameters

Parameter	**Description**
Ns	Number of steps taken by the agent
Nr	Number of visited cells, unique cells
NT	Total number of accessible cells inside the search field (All the available free state cells).
N_a	Total number of grid cells inside each field, e.g. $N_a = 33 \times 33 = 1089$.
Ni_k	Number of times that cell k has been *revisited*
ni	Expected number of visits to a cell.
n	An upper limit for the number of times a cell can be visited.
P	The probability that a cell is *revisited*.

The "effectiveness of coverage" metric (U) is defined in equation 4.3 as:

$$U = \frac{Ns}{\sum_{k=1}^{Nr} Ni_k + NT} \tag{4.3}$$

Where $\sum_{k=1}^{Nr} Ni_k + NT$ is an estimate of the maximum number of steps that can be taken by an agent. In order to optimise our exploration algorithm, we assume there is an upper limit (n) to the number of times a cell can be visited. If all cells are considered equal, we can then derive an expression for the expected number of visits (ni) from the probability (*P*) that a cell is revisited. We follow the same technique that *William Stallings* applied to approximate the expected degree of improvement in data link transmission, known as stop-and-wait ARQ (Automatic Repeat Request) [Sta04].

The probability that single cell will be visited exactly i times by the agent during the exploration procedure is $P^{i-1}(1-P)$, i.e. the cell is being revisited $(i-1)$ times. The probability of this occurring is simply the product of the probabilities of the individual events. Then:

$$ni = Expected[\text{ visiting a cell }] = \sum_{n=1}^{\infty} n \times Probability[\text{n times visits}] \tag{4.4}$$

or

$$ni = \sum_{n=1}^{\infty} n \times P^{n-1}(1-P) = (1-P)\sum_{n=1}^{\infty} n \times P^{n-1} \tag{4.5}$$

Let us develop some approximation to determine the expected number of visits (ni), *by applying the infinite sum equality:* $\sum_{i=1}^{\infty} \mathrm{i} * \mathrm{X}^{i-1} = \frac{1}{(1-\mathrm{X})^2}$ *, when* $-1 < \mathrm{X} < 1$ *it give us*:

$$ni = \frac{1}{(1-P)} \tag{4.6}$$

Approximating $\sum_{k=1}^{Nr} Ni_k$ to give us:

$$\sum_{k=1}^{Nr} Ni_k \approx Nr \times ni = Nr \times \frac{1}{(1-P)} \tag{4.7}$$

we see that

$$U = \frac{Ns}{\frac{Nr}{1-P} + NT} = \frac{Ns(1-P)}{NT(1-P) + Nr} \tag{4.8}$$

Because the agent has no prior knowledge of the number of free cells inside the search field, it needs to estimate NT from the number of cells that it observes during the exploration procedure. At each time step our agent can only see 8 cells that surrounds it. So it assumes that the maximum number of empty cells, at each time step, is approximately $8 * Nr$. The effectiveness of coverage at time step t can then be expressed as:

$$U^t = \frac{Ns(1-P)}{Nr(9-8*P)} \tag{4.9}$$

To enable our agent calculating above values, all the cells visited by the agent should be stored in a matrix (*Pos*). Each time step, the 'Pos matrix' (od_t=(x_t,y_t)) is updated together with *Ns* and *Nr*. We

assume that our agent is aware of its direction θ_t by its *IMU* to reduce the odometry error [Duc00] as much as possible. In addition the robot has a counter vector *C*[*Nr*] to store the number of times each cell has been visited. Therefore each element inside vector *C* [*Nr*] is representing one unique cell. We introduce the maximum number of expected steps as $mes = \frac{Nr(9-8*P)}{(1-P)}$. This number is estimated by the agent at each step. The value of mes only increases when a new cells is visited, a revisit to a cell causes the value of mes to remain the same. U^t enables our agent to switch to another behaviour . For instance if $1 \leq U^t$ agent should terminate its performance (exploitation). In Chapter 5, we will use this metric to define a reward function for our agents.

The search agent explores the environment by moving to a free cell among the 8 cells that surrounds it and the environment then presents the agent with a reward provided by a task–dependent reward function. If an agent always selects an action that maximises the value of the reward function it is said that the agent is following a greedy policy, i.e. it exploits what it has learned during the exploration.

4.3.2 Parameter Evaluation

We should estimate the most suitable upper limit (n) to the number of times a cell can be visited and the probability (*P*) that a cell is revisited during exploration. In this regard, we apply ANT agent to select these numbers. As discussed in Chapter 3, ANT algorithm is attempting to visit least visited cells as much as possible. Therefore, it is a suitable exploration algorithm to help us selecting the above values.

We set up two experiments, the default values are a search field of 50×50 grid cells, we change the position of the goal 20 times and we run the algorithm 100 times for 70 search fields.

- to reduce the redundancy of revisiting the explored cells, and minimising goal discovery time we need to set a limit of revisiting a cell. Revisiting some cells several times is inevitable during exploration procedure, especially in our cluttered and confined fields. However, to avoid unnecessary revisiting as much as possible Ant agent estimates the largest number of revisiting each cell in each run time. For the minimum discovery time of each search field, among 2000 times running the ANT algorithm, we select its maximum number of revisiting a cell. The largest number among all 70 search fields will be selected as our upper limit, n. From our result this is set to $n = 7$.

- now we should estimate the probability that a free cell inside a search field is revisited up to 7 times, we follow: i-recording the number of cells that they have been revisited up to 7 times (as estimated above) to find a goal ($R_{revisit}$); ii- recording the total number of free cells (T_{free}); iii- $P = \frac{R_{revisit}}{T_{free}}$; this is estimated for all available runtimes; iv- estimate the average $\bar{P}$ among all estimated probabilities. From the result this is set to $\bar{P} = 0.83$.

The effectiveness of coverage for our Oracle algorithm remains the same all through the exploration procedure, $U_{oracle} = A$. This is $U = \frac{Ns}{Nr} \times A$, where $A = \frac{9-8P}{1-P}$, since the total number of steps and the total number of unique cells are the same for finding a single known goal, $Ns = Nr$. However, for all other algorithms, this value will increase in some point during the exploration. This is because the agent starts revisiting some cells, $Ns > Nr$.

4.4 Evaluated Benchmark

Benchmarking is a common scientific comparison instrument. A good example, in this regard, for a successful performance measurement is the computer vision community. There are several projects that aim to provide image data bases to other researchers [AoT]. These image databases are supplemented by ground truth images and algorithms that calculate performance metrics. In doing so, the community is able to make progress and to document this progress in fields like image segmentation and object recognition. Unfortunately this kind of controlled performance monitoring is not widespread in the robotics community. Even though there are several ways of comparing the performance of robotic algorithms and their systems, a basic first step is to provide experimental data and results to a community of well established research groups but up to now this has only been done by small projects [SFG] or individual researchers.

There are several frameworks that are used as network servers for robot and sensor control and run on the robot/sensor hardware as well as on the computer modelled hardware. For instance USARSim [Ut] is a simulation tool that provides NIST sponsored reference test arenas for autonomous mobile robots. It has the ability to model buildings in various stages of collapse and is intended to provide objective performance evaluations for robots as they perform a variety of urban search and rescue tasks. It provides users with the capability to build their own environments and robots. Its socket–based control is designed to allow users to test their own control algorithms. However, they have yet to define any complexity index that enables the users to differentiate and classify test fields.

The most popular options to compare robotic systems are competitions like RoboCup [Fed] (covering both inside simulation environment and real world replication), ELROB [TERT] or the Grand Challenge [Cha]. Through these kind of competitions it is possible to measure the level of system integration and the engineering skills of a certain team, but it is not possible to evaluate the performance of a subsystem or a single algorithm, especially in its early stage of development.

In addition to the fractal dimension introduced in Chapter 3, we are able to formally characterise the Brownian random movement characteristics of the BRW algorithm, through its *power spectral density* (PSD). This index characterises the random movements made by the mobile agents used to produce the pathways for our simulation environments. Furthermore, the fractal dimension (of a time series) of the Brownian random movement is directly related to the Hurst exponent "*H*". H for the statistically self–affine dataset (referred to as α in Chapter3) is defined for each search field as: $H = 2-FD$.

The power spectral density is:

$$P(f) \propto f^{-\lambda} \tag{4.10}$$

with $\lambda = 2H + 1$. The Brownian motion has a spectrum of [Lo89]:

$$P(f) = Kf^{-2H-1} \tag{4.11}$$

The spatial frequency for free grid cells in the search field is $f = \frac{N_f}{N_a}$.

Table 4.2: Power Spectral parameters

Parameter	Description
f	Spatial frequency
K	Constant value
N_f	Total number of *empty* grid cells inside each field
N_a	Total number of grid cells inside each field, e.g. $N_a = 33 \times 33 = 1089$.

As shown in 4.2 the power spectral density of each search field is inversely proportional to the number of time steps an agent spends searching before it discovers the first trapped victim. Each point in this graph represents the average obtained from running the BRW algorithm 100 times.

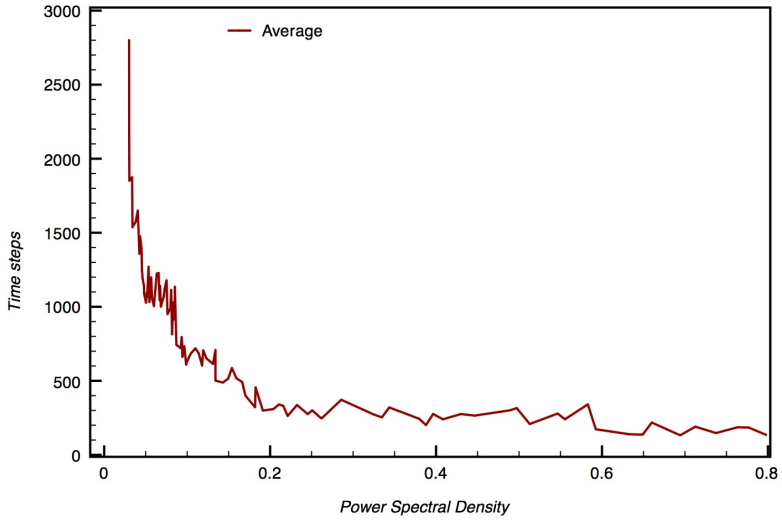

Figure 4.2: Average victim discovery time steps for BRW algorithm

Furthermore, as $x \to 0$ then $y \to \infty$, and as $x \to \infty$ (i.e. no empty cell to explore) then $y \to 0$. We are therefore able to express the relationship shown in figure 4.2 between the victim discovery time measured in time steps y and power spectral density $P(f)$, which $x = P(f)$, by the approximation:

$$y \approx \frac{\beta}{x}, \quad \beta \in \mathbb{R}^{+} \tag{4.12}$$

Using standard regression technique we can now determine the function that provides the best fit to the data. This approximation technique is called regression. Roughly speaking regression is the exercise of fitting a function to data (e.g. fitting function for one parameter is a line and for parameters is a plane). In this regard defining an error measure and minimising this error usually estimate the parameter(s). The measure of error used in standard linear regression [STC04] problem is the mean squared error.

4.4.1 Ridge Regression

Linear regression allows us to determine the best–fitting line for the set of data points (X, y), represented in scatterplot in the figure 4.3. We define that, as the line that corresponds to the smallest residual. Let

X be the data set matrix of dimension $(2, q)$ and y the target matrix of dimension $(1, q)$. The regression line model is then given as:

$$\hat{y}_i = w_0 + w_1 * X_i^T + \varepsilon \quad (i \in \mathbb{Z}^+) \tag{4.13}$$

Where $\hat{y}$ is the predicted value of y, X, where $X = \frac{1}{x}$ (to have a linear system), $\varepsilon \sim (0, 0.5)$ represents the Gaussian noise, and $W(weightfactor) \in \mathbb{R}$ [Gro03].

$$\mathbf{X} = \begin{pmatrix} 1, X_1 \\ 1, X_2 \\ :, : \\ 1, X_q \end{pmatrix} \quad , \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ : \\ y_q \end{pmatrix} \quad , \quad \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ : \\ \varepsilon_q \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} \tag{4.14}$$

The predicted weight constant $\hat{W}$ should be selected so the prediction error is minimised and because we have 100 search fields $q = 100$. We use the regression analysis to predict the time–series for the BRW exploration algorithm. In this regard we use ridge regression least–square estimator [Cle08] to estimate the weights as

$$\hat{W} = (X^T X)^{-1} X^T y; \tag{4.15}$$

Where X and y are the training data obtained by running the BRW algorithm inside 100 search fields. The goals of linear least squares are to extract predictions from the measurements and to reduce the effect of measurement errors.

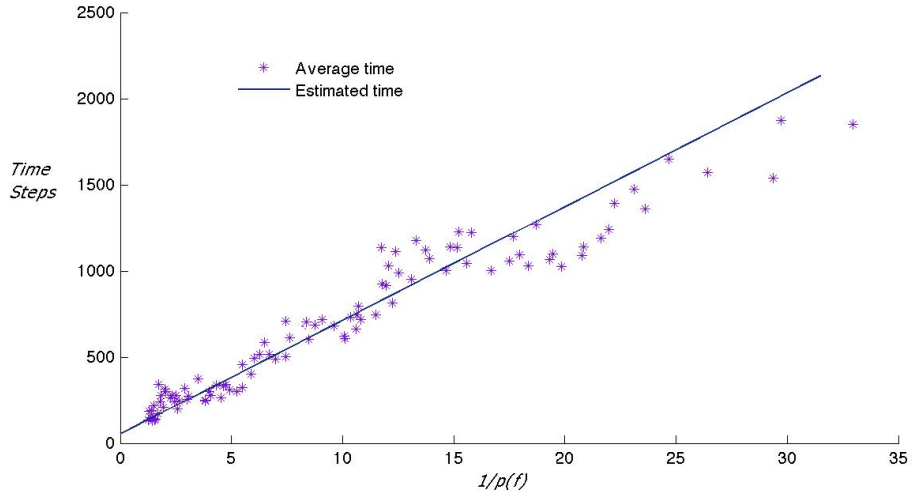

Figure 4.3: Best fitting line and average discovery time steps (scatter) for BRW algorithm

Using equation 4.15 we estimate the constant W as $\hat{w}_1 = 64$ and $\hat{w}_2 = 42$[1]. This enables us to develop a mathematical benchmark through equation 4.13 and users are then able to develop various

[1]There are estimated and plotted by MATLAB

Algorithm 5 *Benchmark Generation*

Input: *f:spatial frequency; fd:Fractal dimension;* S*: Generated Search field ;*

Output: y*;*

$K = 0.1$;

$W = \begin{pmatrix} 64 \\ 42 \end{pmatrix}$;

for S **do**

  Calculate the PSD of S*: PSD=compute(*W*, f, fd,* K*) [equ: 4.11]*

  $X = \frac{1}{PSD}$

  Calculate the time step y *from [equ: 4.13]*

end for

return y;

exploration search fields, according to our search field generator, calculate their fractal dimension using the FracLac tool, and predict the victim discovery time steps required by the BRW exploration algorithm.

4.4.2 Predicted Time Series

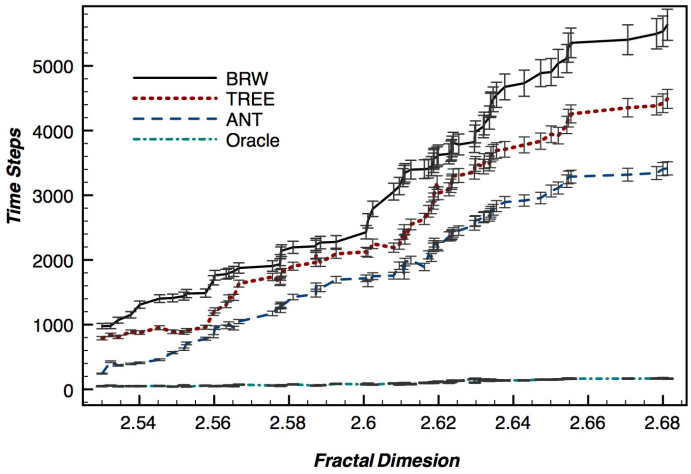

Figure 4.4: Oracle against Predicted benchmark (BRW), TREE, and ANT (one time step=36 seconds)

The default values are: frame size of 50×50, 5 agents, and one single goal (the goal is removed randomly inside the search field 50 times). The performance of a single Oracle agent, multi-agent TREE and ANT algorithms are compared to our predicted benchmark for a range of fractal dimensions in figure 4.4. In an area as large as 50×50 the available algorithms perform poorly (in terms of discovery time) comparing to our oracle agent. Therefore, in the next chapter we investigate frontier based exploration algorithm. With no transition errors, this algorithm is able to reduce the discovery time significantly. However, we managed to adapt this algorithm to deal with positional uncertainty (outcome of odometry errors) by introducing *Directed Ahead Exploration Agent (DAEA)*.

4.5 Discussion

In previous chapters we have discussed how to model the dangerous, cluttered and confined parts of the NIST red course through simulations. Our search field generator uses fractal path tortuosity to differentiate its random generated confined fields. Furthermore, in this chapter, we have introduced essential standard metrics that will be applied through this thesis to evaluate the multi–agent system performances. Through our presented technique, we are able to predict the BRW algorithm according to the power spectral density of the search fields. Therefore we manage to develop a mathematical–based benchmarking technique.

Later on we deploy our on–line metric to develop an advanced frontier–based exploration strategy. This expected utility will asset the agent to decide: *i*- how to move to the frontiers (regions on the border between open space and unexplored space); *ii*- what to do in the face of uncertainty (odometry errors that lead to positional uncertainties). Our goal is to reduce the discovery time (compared against other competing algorithms) inside the narrow gaps and small voids of a collapsed building, by including realistic transition errors.

In the next chapter we will introduce our novel directed path planning toward frontiers for narrow pathways and small tunnels. Directed exploration methods utilise some exploration–specific knowledge that increases the 'knowledge gain' and decrease the 'knowledge uncertainty' so that the value function is learned faster or more completely (it is categorised as the greedy exploration). This advance search technique will be tested in various complex and ambiguous test fields generated by our search field generator. We compare our work to existing algorithms, as well as our evaluated benchmark, which is a well–known search technique for autonomous mobile robots.

Chapter 5

Performance Aware Agents

Consider a scenario where the exploration robots are provided with the map of a search territory in advance. This map can either be metric, giving the exact locations of obstacles, or a graph representing the connectivity between open spaces. However, even in this scenario most of the mobile robots are unable to navigate efficiently when they are placed in an unknown position, even inside a known environment (kidnapped robot problem). While many robots can navigate using maps, there are robots that can build their own representation of the environment e.g. map, by recording information about their surrounding in an unknown search field. In this chapter we refer to exploration as the act of filling an occupancy grid memory while moving through an unknown environment. This memory grid can in turn be used to approximately locate trapped victims or for the basis of further exploration.

A good exploration strategy is here one that is able to generate at least a nearly complete occupancy grid memory of a search terrain in a reasonable amount of time. In real world scenarios it is a challenging task to make such grid memory because odometry data in practice tend to be noisy and GPS devices often are unreliable in an indoor environment. This means that we are facing a problem that is not deterministic and in the literature, Partially Observable Markov Decision Process (POMDP) are often proposed to address the issue of goal pursuit in such environments [KLC98]. Systems resulting from solving POMDPs can handle the uncertainty such as agent observations (*POMDP technique is completely discussed in Chapter 2*). However, to find an optimal policy is computationally expensive.

There are various techniques to reduce the complexity of POMDP planning for "exploration robots". This is by assuming a discrete environment (e.g. square grid) to explore rather than a continuous search terrain. As a result, there has been a lot of work on computing *approximate POMDP solutions* [Hau00b, Roy03, LCK95] and specifically approximate point–based POMDP algorithms [SV05, ZZ01]. Roy [Roy03] has presented an algorithm to find a low–dimensional representation of the belief space. Through dimension reduction, the planning process becomes much more tractable. However, the algorithm relies on a rough metric map of the environment. Littman et al. [LCK95] introduced a framework between POMDP and MDP (Markov Decision Process). They converted POMDP belief space to MDP with a continuous state space. Their results are only satisfactory for up to 100 state cells. Spaan and Vlassis [SV05] introduced a randomised approximate point based value iteration algorithm for solving POMDP for robot planning. Their algorithm first needs to collect a set of belief

points by exploring the environment and based on this set of belief points a control strategy is learned. Collecting belief points randomly is very time consuming and is not practical for time critical tasks such as search and rescue. We are interested in the latter algorithm. However, in our approach the agent *collects belief points with one–step optimal planning instead of randomised procedure*. In this chapter we investigate agents performing ANT, TREE, Brick and Mortar, BRW, and frontier–based exploration procedures with and without odometry errors. For agent that their performance is based upon its collected data, accumulated errors will cause a dramatic positional uncertainty. We further investigate our frontier–based algorithm as follows:

Without positional uncertainty- we design a mathematical model–based control approach to direct an agent inside a search field toward the boundary between explored and unexplored regions. We deploy our novel on–line metric, *effectiveness of coverage* (introduced in Chapter 4), that is assessed by the agent for each time step through its limited sensor readings. This mathematical–model frontier–based strategy, called *Directed Ahead Exploration Agent* (DAEA) algorithm, and we test its performance over various search fields.

With positional uncertainty- after investigating our approach without transition errors, later we investigate the extended DAEA (EDAEA) by adding positional uncertainty. The EDAEA algorithm is a mathematically established algorithm coupled with probabilistic information distribution, where the beliefs of agent regarding to its surroundings are included. These additional data will enable our search agent to reduce its uncertainty through POMDP decision making.

In this chapter: *i*- We will assess whether an exploration procedure can be improved by responding to such an on–line metric. In this regard, an agent is strictly rewarded according to its effectiveness of coverage at each time step through a *function reward*. *ii*- We will investigate how an extended DAEA algorithm enable our agents to reduce their uncertainty and therefore minimising the exploration redundancy compared to other available strategies.

5.1 Reward Function

If an agent can be considered independent, i.e. its performance does not depend directly on the actions of others, it may be desirable to reward the agent based on its performance. In this regard, we apply a reward function (equ: 5.1) after each time step.

$$\mathcal{R}(s) = \begin{cases} 1 - U^t & \text{if } \ Cr \leqslant n \ or \ U^t < 1; \\ -1 & \text{otherwise} \end{cases} \tag{5.1}$$

Where U^t is the effectiveness of coverage ($U = \frac{Ns}{mes}$ & $mes = \frac{N_r(9-8*P)}{(1-P)}$) of current cell. *Except* in our *new algorithm* that *t-1* refers to current cell and *t* represents the *future* cell selected by our agent to move into, e.g. the immediate effectiveness of coverage. As defined in the previous chapter, Cr vector stores the number of each unique cell that is visited by an agent while exploring a search field. We consider an upper limit as n. To achieve a time efficient exploration, our agent should not revisit a cell more than its adjusted limit. We have set $n = 7$, as discussed in Chapter 4.

For all exploration algorithms (i.e. BRW, ANT, TREE, and Brick &Mortar algorithms), in this chapter, the search agent receives reward after taking each step. Therefore their exploration task and the reward function are treated separately. However, for *DAEA* algorithm, the exploration task is defined by an on–line metric, and we build the reward function upon this on–line performance metric, see equation 5.1. All the accessible cells around the agent have their immediate expected reward. Note that among all the defined exploration strategies in this chapter *only DAEA aims to maximise its reward in each time step*. In DAEA, action selection is designed to direct the agent through an effective coverage procedure, will discuss further below, which is moving to frontier regions.

5.2 Frontier–Based Technique

A *conventional control theory*, tries to make a *mathematical model* to direct a system. In contrary to rule–based control system, e.g. [Yam97], that produces a model of skilled agents, in the *mathematical control–model*, e.g. [RK04], usually no agent is factored in the design. *We present the basic frontier–based algorithm [Yam97] in a mathematical model rather than its original rule–based control system, because by this model we are able to:*

1- define a systematic reward function for non–deterministic decision process to asset our agent toward frontiers. A frontier is any known and unoccupied cell that is an immediate neighbour of an unknown (unexplored) cell;

2- recognise *problematic uncertainties*, through a predefined threshold;

3- apply approximate–POMDP technique to reduce positional uncertainties (because of transition errors). EDAEA agent reduces positional uncertainty, which prevents it from finding a trapped victim independently of its position in a minimum time. Thus, an agent may sometimes be forced to persist with a high level of uncertainty, in a portion of the belief–space in which uncertainty reducing actions are very expensive in terms of time.

Basic Frontier–based– Using this exploration technique a robot moves toward boundaries or frontiers between known and unknown environment. *In this technique agent is seeking to visit new areas without considering any uncertainties in its collected data*. In occupancy grid memory an occupancy probability value (P_{om}) of zero corresponds to a free cell and a value of one corresponds to a cell occupied by an obstacle. Initially all the available cells inside a predefined matrix memory are assigned a value of 0.5, i.e. equally likely to be occupied or free.

- frontiers are detected within its occupancy grid memory this is by scanning probabilities stored inside its grid memory. It marks them as its goal cells, and attempt to find them as fast as possible. A set of simple rules are applied as follows:

- a path planner uses a depth-first search [Sed01] on the memory grid, starting at the agent's current cell and attempting to take the shortest obstacle–free path to the cell containing the goal location. Among multi goal cells one will be selected randomly.

- if the agent is unable to make progress toward its destination for a certain amount of time, then

the robot will determine that the destination is inaccessible, and its location will be assigned to 0 probability.

DAEA– There are two modes defined for DAEA technique(5.2) : *i-* maximising the expected reward, *ii- Basic Frontier–based* approach. This search technique implements immediate optimal planning by maximising the expected reward (E[(R(s)]). If there are more than one solution it selects one randomly. When all eligible cells around the agent have $\mathcal{R} = -1$, the agent moves to a boundary between open space and unexplored space, performing *frontier–based* algorithm as discussed above. When a robot moves to a frontier, it can see into unexplored space and add new information to its occupancy grid memory. The agent follows frontier–based algorithm till $\mathcal{R} \neq -1$, then it switches to its first mode, maximising the expected reward (by its on–line metric). Where a^* is defined as the optimum action selection for our agent, we present action selection:

$$\text{Action selection} \ : \begin{cases} a^* = maxE[R(s)], & \text{if } \mathcal{R} \neq -1; \\ a^* = maxNr & \text{otherwise} \end{cases} \tag{5.2}$$

We set up an experiment to compare the behaviour of DAEA performance aware algorithm to other competing performance aware algorithms such as : B&M, ANT, and TREE. We also compare the result with BRW, unaware performance. Performance aware algorithms are techniques that utilise recorded odometry data. We later evaluate the importance of reducing positional uncertainty for such algorithms.

5.3 Exploration Without Positional Uncertainty

To test the performance of our DAEA algorithm and compare it to other available algorithms we set up an experiment. The default values in this experiment are: frame size of the simulated environment is 33×33 state cells, they are differentiated according to their fractal dimension of search field's pathways. Three agents are sent in to perform a search algorithm to explore a single goal. They are rewarded after their next cell selection to move into, according to the 5.1. The available algorithms in this experiment are: Oracle, BRW, ANT, TREE, and B&M algorithms. The Brick and Mortar (B&M) exploration algorithm [FTL07] is specifically designed for *intelligent environments*. However, it is possible to apply some changes to the algorithm so that our mini robots can utilise it even inside a non intelligent environment such as ours. Through this algorithm, the agent is able to safely eliminate small regions and reduce the size of its search field.

All agents fill an array of their visited cells and the number of times each cell is visited. Therefore we define an occupancy grid memory for each search field, all the available cells inside it are assigned to 0.5 with 33×33 cells. This grid memory will be filled through the exploration procedure by all agents. There is no interaction among the agents (no team work). We run the experiments for all above search strategies, and estimate their discovery time and their total earned reward to locate a single goal. Their results are compared in figures 5.1, 5.2.

B&M– An agent performing B&M algorithm marks cells in the search field as visited by checking the occupancy grid memory for walls and virtual walls in its surroundings. Agents create the virtual wall

around their matrix cells in order to create a subspace and limit their explorations to within the subspace. For instance, when an agent is moving west or south, it scans its south wall to check the cells to its south–west and south, if they are occupied or marked as visited, then it marks the current cell as a visited cell. To accomplish this each agent has to record two arrays, one of visited cells and another of explored cells, as in algorithm 7. Explored cells are those that have been traversed at least once by the agent. However, the agent might need to go through it again in order to reach unexplored cells. Visited cells are cells that have already been explored and no longer are required to go through them again. Conceptually these cells are equivalent to obstacles and we will refer to them as virtual walls.

If the robot reduces its search field correctly during exploration, the exploration time will be reduced noticeably. The Brick &Mortar algorithm aims to progressively thicken the blocks of inaccessible cells, whilst always keeping accessible cells connected. The latter can be achieved by maintaining corridors of explored cells that connect all unexplored parts of the search area. The main rule that an agent must obey locally is never to mark the current cell as visited if, by doing so, it blocks the path between two accessible cells.

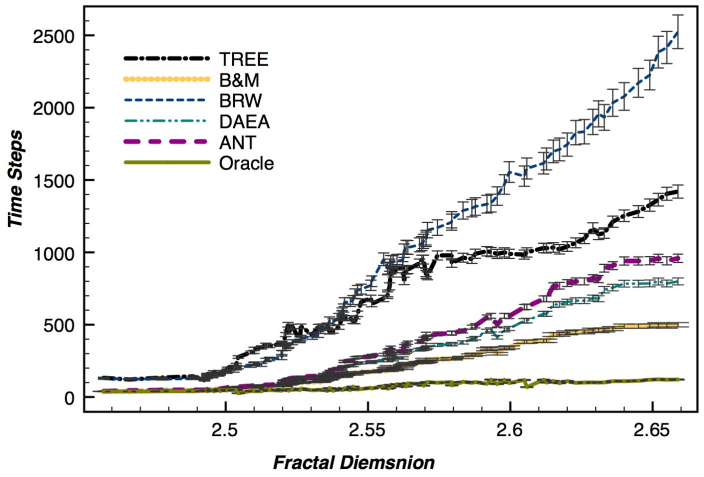

Figure 5.1: Agent's time step to find a single goal, one time step=36 seconds

In figure 5.1, where we compare a goal discovery time of 6 different search techniques, B&M and DAEA algorithms perform very closely to our oracle algorithm, i.e. our optimum search. This graph is plotted with the corresponding standard deviation bar. Without considering any transition error B&M agent has the best performance among all other 5 strategies. However, by adding odometry errors (i.e. 6.6 % odometry error) *the Brick & Mortar approach only manages to complete its execution for* 17 *out of* 100 *search fields*. This is tested by an experiment, which all the free passable cells inside a search field should be visited at least once by the B&M agent. If all available free cells are visited, the coverage is known as 100% complete. This incomplete performance is the result of the virtual wall being bricked wrongly by the agent and therefore erroneously blocking an area. Therefore, in the next section we apply *Hybrid B&M algorithm* (i.e. combination of ANT and B&M algorithms) to deal with agent positional uncertainty.

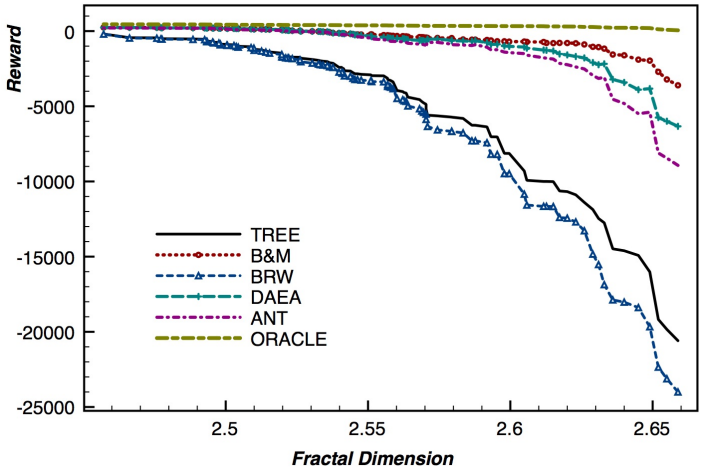

Figure 5.2: Estimated reward, earned by a single agent

Algorithm 6 *performDAEA*

Input: $Cr[]$:*number of times a cell is revisited;* ogm:*occupancy grid memory; A:agent*

Output: ogm:*occupancy grid memory,* Ns, Nr;Cr;

while *true* **do**
  A:$pos\_start(0,0)$; $P = 0.83$; $n = 7$;
  for *each cell inside the window* **do**
    Calculate the U:*effectiveness of coverage, [equ: 4.9];*
    R=*compute*(U, n),*[equ: 5.1];*
  end for
  if $R \neq -1$ **then**
    move toward cell with maximum R.
    if *there are more than one cell* **then**
      select one randomly
    end if
  else
    move toward cell with maximum P^{Cr}
    if *there are more than one cell* **then**
      select one randomly
    end if
  end if
  if *the selected cell is not revisited* **then**
    Fill the ogm
    $Nr++$;
    $Cr++$;
  end if
  if ogm *is completed* **then**
    return ogm
  else
    Ns++
    return Ns, Nr, Cr
  end if
end while

Furthermore, in figure 5.2 the earned rewards, by an agent, are illustrated. Each time step and reward point are the average of running the algorithms 100 times for BRW and 2000 (for all other algorithms we change the position of a goal 20 times) for other algorithms. This figure demonstrates that, the rewards earned by B&M (unaware of its reward function), when $FDP > 2.6$, is larger than rewards earned by DAEA agent and all other exploration agents. This is because B&M agent is highly aware of its explored and visited cells and handles these cells differently.

The results of the two last graphs indicates the correlation between goal discovery time and performance awareness. It is possible to achieve performance, which approaches to our optimal goal discovery time, through a function reward. DAEA discovery time reduction is statistically significant for complex environments (p-value < 0.008) compared to BRW (unaware algorithm). By applying an on–line metric, prior to selecting the next cell, the agent is able to decrease its goal discovery time significantly relative to a less aware search techniques (ANT, TREE) or completely unaware technique (BRW).

Algorithm 7 *performBrick & Mortar*

Input: *A: agent ; V[ ]: visited cells; Nr; E[ ]:explored cells; ogm:occupancy grid memory*

Output: *ogm:occupancy grid memory, Ns, Nr;*

Agent is released inside the search field at (0,0);

```
1: while true do
2:    if A.chosenNextMovement ==“EAST” OR A.chosenNextMovement ==“NORTH” then
3:       if A.scanSouthWall==“TRUE”AND A.scanWestWall==“TRUE” then
4:          V[Nr]=currentcell
5:       else
6:          E[Nr]=currentcell
7:       end if
8:    end if
9:    if A.chosenNextMovement ==“WEST” OR A.chosenNextMovement ==“SOUTH” then
10:      if A.scanNorthWall ==“TRUE”AND A.scanEastWall==“TRUE” then
11:         V[Nr]=currentcell
12:      else
13:         E[Nr]=currentcell
14:      end if
15:   end if
16:   if the selected cell is not revisited then
17:      Fill the ogm
18:      Nr++;
19:   else
20:      n++;
21:   end if
22:   if ogm is completed then
23:      return ogm
24:   else
25:      Ns++
26:      return Ns, Nr, n
27:   end if
28: end while
```

In our next step we introduce an extended DAEA (EDAEA) agent. This agent is aware of the amount of transition uncertainty that will face while exploring a search field (i.e. in addition to its action utility awareness).

5.4 Exploration With Positional Uncertainty

We have presented a time efficient algorithm that returns an almost optimal exploration strategy, among competing algorithms, through a dual–mode controller. One of the problems with this frontier–based agent is the use of the effectiveness of coverage function that assumes a stationary and noise free world throughout the whole exploration procedure. This assumption is not rational for a learning agent that its performance awareness is based on its odometry recorded data.

In order to develop a *risk–sensitive* exploration strategy we need a mechanism that allows the agent to take into consideration the expected revision of its *belief* when computing the expected effectiveness of coverage. Such a mechanism requires a method for representing the agents uncertainty regarding to its noisy sensor data. The binary-occupied or not, representation of the explored environment by noisy sensors is insufficient to represent the uncertainty the robot may have regarding to its state cells, thus we address this uncertainty through a *coverage grid memory*.

The agent can move in 8 directions and looks for the presence of a cell with maximum reward. We consider that, inputs from the move actions are imperfect while the *IR sensor* inputs are perfect at detecting empty and occupied cells inside its eight segmented window vision $\mathcal{O}(s', a, z) = 1$. We assume that the robot no longer necessarily will move in the intended direction. Rather, there is a 15% probability that it will move into one of the neighbouring directions if they are free. For instance if the robot is supposed to move north, it could move either north–east or north–west with probability of 0.15 provided they are free. From these data we generate the transition vectors $\mathcal{T}(s', a, s)$, at each time step.

The values Nr and Cr recorded by the agent are now derived from the transition vector. The resulting values $Cr[Nr] \times \mathcal{T}(s', a, s)$, and $Nr \times \mathcal{T}(s', a, s)$, so the number of visited a cell is no longer a whole number. Nor, it is no longer, the actual number of visited unique cells since it is multiplied by the probability of robot visiting it.

5.4.1 Belief System by Bayes Theorem

The belief, $Bel(s') = Pr(s'|z, a, s)$, is the probability that the robot has at time step t when it is in a state s', after taking action a and making observation z. Figure 5.3 describes a POMDP while future observations and beliefs are *conditionally independent* from the past (Markov assumption is applied). In another words we consider a POMDP as a belief state MDP, that is, an MDP with a state space defined by set of beliefs [Roy03].

The probability distribution can be updated for each time step using a Bayes' filter:

$$Pr(s'|z, a, s) = \frac{Pr(z|s', a)Pr(s'|a, s)}{Pr(z|a, s)} \tag{5.3}$$

$$= \frac{Pr(z|s', a)\sum_{s \in Wv_t} Pr(s'|a, s)Pr(s)}{Pr(z|a, s)} \tag{5.4}$$

$$= \alpha\mathcal{O}(s', a, z) \sum_{s \in Wv_t} \mathcal{T}(s', a, s)\ Pr(s) \tag{5.5}$$

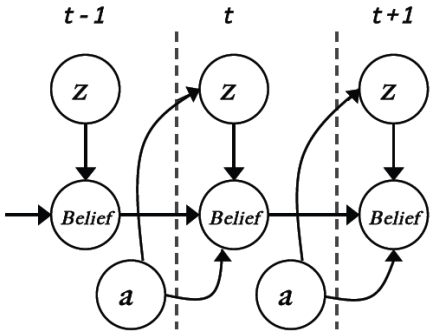

Figure 5.3: The graphical model of the POMDP, Markov assumption is applied on its belief system

Where α is the normalisation constant and Wv_t is the window vision at time step t. We simply refer to $Pr(s^{t-1})$ as the distribution over states at time $t-1$. Thus we define it as $Bel(s)$, the previous belief inside the previous window Wv_{t-1}. An initial belief is assumed to be known.

The next step is to define a threshold that allows the agent to switch between control modes, whenever it deems this to be necessary. This is usually a free parameter selected by the user that should prevent problematic uncertainty, and let the agent act sub–optimally when its uncertainty is ineffective to its performance. In this regard, we define Λ as the probability of selecting a free cell inside the current window when it has been revisited:

$$\Lambda = \sum_{s,s' \in Wv_t} Cr \times \mathcal{T}(s', a, s) \tag{5.6}$$

We later apply Λ as our controller mode.

5.4.2 Dual–mode Controller

All the cells with values of $\Lambda \leqslant n'$ are categorised as a valid evidence, $n' = n \times \mathcal{T}(s', a, s)$. However, when $\Lambda > n'$, the defined evidence becomes invalid and our agent is no longer able to guide itself through an effective exploration coverage (i.e. to maximise its reward). We therefore need to discuss how the controller is able to deal with its confusion by minimising the *Belief Entropy*.

Action entropy is an example of a probabilistic which can act to reduce the uncertainty [CKK96]. It switches between two distinct modes: seeking reward and seeking information. When the entropy of the belief–optimality distribution is above a given threshold, and the agent therefore is uncertain which action to take, the action entropy algorithm selects the action that will best reduce its belief uncertainty over a *one set of actions*.

Our dual–mode controller alternates between pursuing expected reward and gathering information to refine the belief state. In this section we address how to trade off between maximising the expected reward and minimising the belief entropy. The belief entropy distribution is given by:

$$H(b) = -\sum Bel(s') * logBel(s') \tag{5.7}$$

where $Bel(s') * logBel(s') = 0$ when $Bel(s') = 0$

The lower is the $H(b)$ value, the more certain is the agent about its information distribution. When the robot is confused, the entropy is high, and it will be reasonable to take actions to reduce the entropy of the belief distribution. Furthermore we define the expected entropy of a belief state $Bel(s')$ and action 'a' as:

$$EE(a,b) = H(b')\mathcal{T}(s',a,s) \tag{5.8}$$

Estimating action entropy for a long sequence of actions might enable the agent to have the optimal behaviour (i.e. a behaviour of an omniscient agent) . However, it quickly becomes quite expensive to compute. In this thesis we have tested various length of action sequences and the best number of actions for each sequence is selected, as we will discuss later.

The expected gained reward (V) is estimated by $V = \mathbf{E}\left[\sum_{t=1}^{\infty}\gamma^t R_t\right]$, where γ is set to 0.95 (as discussed in Chapter 2). The dual–mode controller is given by two sub–optimal action selection ($n' = n \times \mathcal{T}(s',a,s)$) :

$$\text{Action selection} : \begin{cases} a^* = argmax[V(s')], & \text{if } \Lambda \leqslant n'; \\ a^* = argmin E[E(a,b)], & \text{otherwise} \end{cases} \tag{5.9}$$

When Λ is above a threshold, the agent is in a state of maximum confusion (invalid evidence). However, the controller is able to direct the agent in a direction that will *minimise the belief entropy*. In this regard the agent will be directed toward respectively: 1- unvisited areas; 2- areas with larger belief distribution.

We have tested three different lengths of actions: $2, 3, 4$ action sequences. These are respectively 24, 48, and 80 belief points around an agent. Larger number of the belief points give more possibility to the agent to reach its desire endpoint, i.e. frontiers. However, as discussed before the computation will become very expensive. For instance for 80 states POMDP agent has to estimate 192 possibilities of performing a sequence of 4 actions. This is causing a noticeable delay (approximately an average of 160 seconds) for our agent at each step to select its next movement. We describe a 24 states POMDP with details in an example in the next section. However, in all our experiments we have applied 48 belief points, with a sequence of 4 actions. *This sequence length setting enables the agent to reach to its frontiers in* $\%77.6$ *of* 2000 *run times, when tested on* 100 *different search fields.* The average delay caused by this length is less than a minute, i.e. 49 seconds.

5.4.3 24 States Approximate–POMDP

We describe how to solve a 24 states approximate POMDP by an example as follows:

Selecting the required belief points– For each neighbouring cells, we select the previously collected belief points as the collected transition probability, equation 5.5. Since we assume that the observation is complete therefore, we approximate our belief points as: $Bel(s_t) \approx \mathcal{T}(s',a,s)$. Furthermore, in addition to the agent's observation window, we define a frame size of 16 neighbour cells surrounding the agent's window vision.

e.g. Collected belief points (T) :

0.85	1.50	2.00	0.85	0.70
0.70	0.70	1.50	0.85	0.85
0.70	0.70	**-S-**	2.00	2.00
0.70	0.70	0.70	0.85	0.85
0.15	0.70	0.70	0.70	0.70

There are a set of rules defined for our agents collecting transition probabilities (belief points), while exploring a field, as follows:

- in the reward and information seeking procedures (as discussed in DAEA algorithm) for each cell, no matter how many time it has been revisited, we store the largest number of its transition probability, and we refer to it as (T).
- in information seeking procedure for each cell with minimum entropy that is revisited by our agent, its transition probability is changed to 1.5, $H(1.5) > 1$, while for all $b < 1$, $H(b < 1) < 1$. Thus, agent will avoid to revisit them, as much as possible.
- transition probabilities for observed occupied cells are set as 2 and this has the largest action entropy and it is always avoided by the agent ($H(2) > H(1.5)$).

Estimating the transition function for all actions– To estimate the expected entropy for each neighbouring cell, , $EE(a, b) = H(b')\mathcal{T}'(s', a, s)$, we investigate the transition probability for each action (e.g. SW) inside agent's observation window as follows:

$T' : SW$	$T' : S$	$T' : SE$	$T' : W$
0 .00 0.00 0.00	0 .00 0.00 0.00	0 .00 0.00 0.00	0.15 0.00 0.00
0.15 0.00 0.00	0.00 0.00 0.00	0.00 0.00 0.00	0.70 0.00 0.00
0.70 0.15 0.00	0.15 0.70 0.15	0.00 0.15 0.85	0.15 0.00 0.00
$T' : NW$	$T' : N$	$T' : NE$	$T : E$
0 .70 0.00 0.00	0.15 0.7 0.15	0.00 0.15 0.85	0.00 0.00 0.00
0.15 0.00 2.00	0.00 0.00 0.00	0.00 0.00 0.00	0.00 0.00 2.00
0.70 0.00 0.00	0.00 0.00 0.00	0.00 0.00 0.00	0.00 0.00 0.00

Estimating the action entropy– E(entropy)$< an >:< startpoint >:< endpoint >: H(b)$

The procedure described is general for all the available sequence of actions that enable the agent to reach to its defined end points. our endpoints are, as discussed in Chapter 2:{s9, s10, ..., s24}.

e.g.

E:SW and SW : s0 : s20 : $H(T_{s0}, T_{s20})$

E:SW and S : s0 : s21 : $H(T_{s0}, T_{s21})$

E:SW and W : s0 : s19 : $H(T_{s0}, T_{s19})$

E:S and SW : s0 : s21 : $H(T_{s1}, T_{s21})$

E:S and S : s0 : s22 : $H(T_{s1}, T_{s22})$

E:S and SE : s0 : s23 : $H(T_{s1}, T_{s23})$

In this regard we have 27 different possibilities, the pomdp–solver will select the minimum expected entropy among all available possibilities. Thus, we follow that sequence of actions, which lead us to the expected minimum entropy.

We estimate the entropy of the belief distribution as: $H(b) = \sum |Bel(s') * logBel(s')|$

e.g. $H(T_{s0}, T_{s21}) = |T_{s0} log(T_{s0})| + |T_{s20} log(T_{s20})|$

$= |0.7(log 0.7)| + |0.15(log 0.15)| = 0.108 + 0.123 = 0.231$

and finally the expected action entropy as: $E = H(T_{s0}, T_{s21}) \times T'(a_{SW})$

$= 0.231 \times 0.7 = 0.162$

5.4.4 Exploration Evaluation

As discussed before, odometry errors may cause confusion in real scenarios. Our next step is therefore to set up an experiment, identical to experiment in section 5.3, to test the vulnerability of all the above exploration techniques to odometry errors. We do this by randomly selecting a cell (i.e. error cell) in the frontier regions, that might be an occupied cell, after every 15 steps and add it to the list of visited and explored cells. These error cells have not visited by the agent before but they have at least a neighbour cell that has been visited.

Through this technique we try to model the effect of a 6.6% odometry error on our agents.

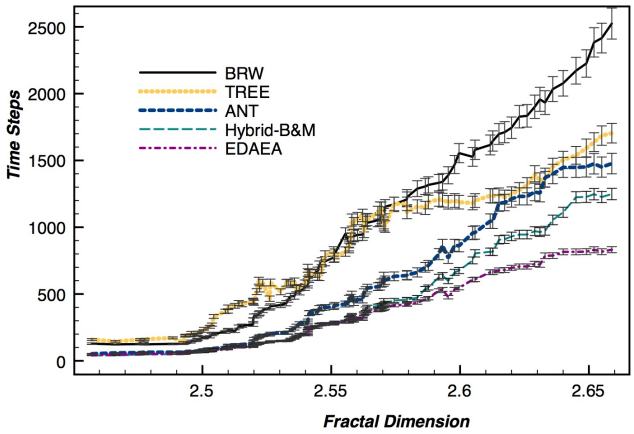

Figure 5.4: victim discovery time steps with 6.6% odometry error, 30×30 cells

As figure 5.4 is indicating that: *i*- transition uncertainty have no effect on the behaviour of BRW algorithm, unaware performance; *ii*- the discovery time for all performance aware algorithms has increased due to transition uncertainties (negative effect of transition uncertainty on the discovery time); *iii*- Hybrid B&M algorithm cannot deal with agent's positional uncertainty as efficient as EDAEA.

Hybrid B&M agent does not behave to overcome the uncertainty, instead it acts as an ANT agent to keep off from being trapped. All other performance aware exploration algorithms in some point will stuck in local–minima when facing positional uncertainty, since in their technique positional uncertainty are not taken into account. However, *EDAEA agent* will measure its uncertainty against its threshold, whenever it deems necessarily it performs sub–optimal actions in order to acquire a better model compared to the competing algorithms and this will generate a better utility in a long run.

EDAEA *switches between the highest expected reward and the lowest expected entropy*:

- if the *uncertainty* of taking an action is lower than a predefined threshold ($\Lambda \leqslant n'$) it selects an action with the highest *expected reward*.
- if the *uncertainty* of taking an action is higher than a predefined threshold ($\Lambda > n'$) the agent selects an action with the lowest *expected entropy*.

Thus, EDAEA technique is able to present an exploration strategy, which holds a practical solution among all other competing strategies, for our complex and unknown environments that are required to be explored in a short time–frame. In Chapter 6 we will investigate the EDAEA search field 'full coverage' , and how to fill a memory grid, which is defined in advance for all agents, as accurate as possible.

Algorithm 8 *performEDAEA*

Input: *Λ[ ]:number of times a cell is revisited; ogm:occupancy grid memory; A:agent;*
Pr: probability of selecting free cell; $\mathcal{T}$:transition vector; *R*:reward function ; *U*:effectiveness coverage;
Output: *ogm:occupancy grid memory, Ns, Nr;Λ;*

while *true* **do**
  $\mathcal{O} = 1$; $\gamma = 0.95$; $n = 7$; $P = 0.83$; $b_0 = 1$;
  for *each cell inside the window vision* **do**
    b:compute($\mathcal{T}$, $\mathcal{O}$, P, Λ))[equ: 5.5]
    V:compute(γ, R);(expected reward)
    E:compute(H(b), $\mathcal{T}$);[equ: 5.8]
  end for
  if $\Lambda \leqslant n$ **then**
    move toward cell with maximum V.
    if *there are more than one cell* **then**
      select one randomly
    end if
  else
    move toward cell with minimum E
    if *there are more than one cell* **then**
      select one randomly
    end if
  end if
  if *the selected cell is not revisited* **then**
    Fill the ogm
    $Nr++$;
  else
    $\Lambda = \mathcal{T}(s', a, s) + \Lambda$;
  end if
  if *ogm is completed* **then**
    return *ogm*
  else
    $Ns++$
    return *Ns, Nr, Λ*
  end if
end while

Note that, the basic frontier–based technique (or even DAEA), is not able to complete its task (exploration with odometry errors) in most of the above tests, like B&M algorithm. Therefore, we did not mange to compare its performance to other algorithms. However, just like Hybrid-B&M, ANT algorithm can help this technique to deal with its positional uncertainty and complete its exploration.

5.5 Frame Size Setting

Whenever our agent fills its occupancy grid memory, it switches from *exploration mode* to *exploitation mode*, and when this happens the agent must return to its mother and report back its grid memory. It therefore selects an angle that will direct it toward the start point of its occupancy grid memory and from there it tracks a path to the wall that will direct it to its mother (backward Wall–Following). Since all the grid cells visited by the agent are recorded, moving backward is just following the tracks. In Chapter 7 we introduce EANT algorithm that is able to direct the agent optimally toward a wall.

To investigate the best frame size for agent's grid memory, while performing EDAEA algorithm to find a single goal, we increase the size of the search field to 50×50 and proceed to compare the performance when a single robot, four robots and eight robots are employed (with odometry error of $\%6.6$). *In this experiment we want to learn the best frame size for our agents to fill in during their exploration procedure.* After filling this predefined memory they will terminate and return to their mother.

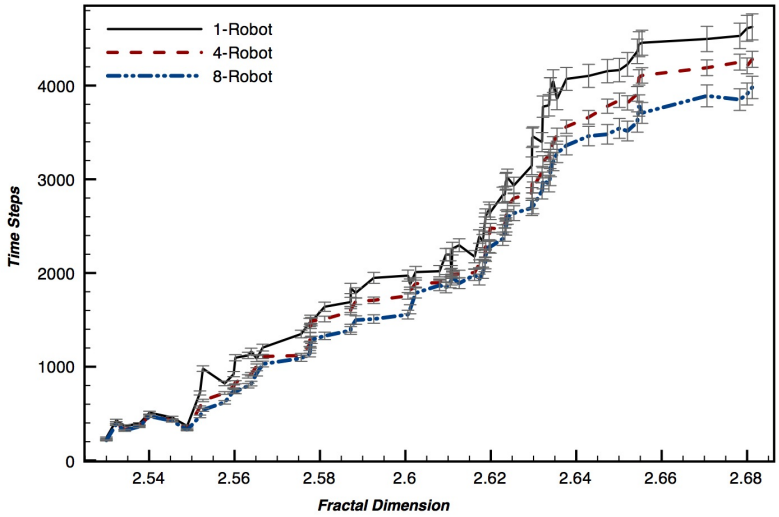

Figure 5.5: Average coverage time steps for Multi–agent EDAEA, 50×50 cells

As indicated in figure 5.5, the performance improvement resulting from an increase in the number of agents is small. *This is because there is no collaboration among the agents.* To improve the performance, we will therefore introduce local grid memories to our agents (smaller than 50×50), and apply a technique aimed at spreading the agents inside the search field in order to fill their local grid memories.

For our multi–agent exploration system (to be discussed further in Chapter 6) we divide the search field to subspaces, and distribute them among the agents. Each subspace, which is normally connected to the wall that was followed by the multi–agent team, forms a local grid memory of an agent, as indicated in figure 5.6.

5.6 Initial Congestion

Teams of robots offer a promising new way to perform search and rescue in environments where it is dangerous or difficult for human rescuers to enter. These robots will typically enter a building at a single

or a small number of points and spread out from there. This can potentially lead to significant initial congestion for the robots, especially if there are only a small number of exits from the room to the rest of the building. However, strategies that allow teams of robots quickly overcome this initial congestion and spread out sufficiently to begin their mission have received little attention in literature. This section will give a brief overview of possible ways to solve traffic congestion in multi–agent systems.

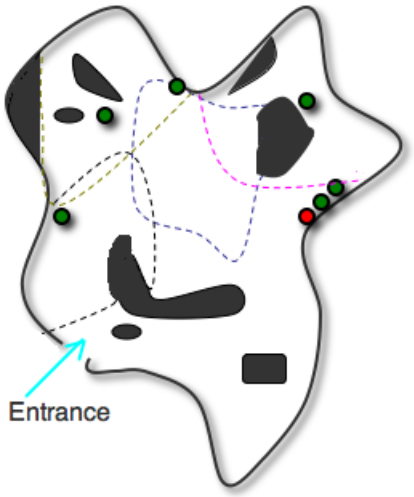

Figure 5.6: Generating a local grid memory of subspaces

In *MARTHA* [AFH+98] a token–based approach was applied, that allowed a very small number, of robots to move one at a time, therefore slowing the overall efficiency. Some robotic swarm approaches appear relevant to this problem, but are usually focused on keeping the swarm together, not spreading it out [KDFA+04]. In general, swarms are often relatively inefficient around tight openings, like doorways. Potential fields and other physic inspired models initially seem to show some promise but, in a congested room, spreading out might not be possible until some of the robots manage to leave the packed room. Thus, the potential field approaches are generally ineffective until the initial congestion is cleared. Note also that clearing the congestion will be very difficult and time consuming for even the most skilled human operators, because the robots have localised viewpoints that provide poor information of their surroundings. Thus, while we do not believe this problem represents a fundamental road block to robot teams, we believe that there is an absence of good solutions to the problem and that a good solution will be the key to improve the practical effectiveness of robot teams for urban search and rescue. In this activity we introduce a technique that prevents the team movement from creating any congestion by forcing them to move in a formation. Our solution is a chain formation, inspired by army ants in the nature.

A species of army ant, Ection burcelli, creates very long trails, which consist of a central path used by ants to bring food back to their nest and two parallel outer paths for the ants that are fanning out into the forest looking for prey. The three–stream traffic system is the result of some basic principles in the ant–biology. Each of the ants in this species leaves a trail of pheromones behind as it moves, attracting other ants and eventually marking the path. Large numbers of ants will naturally follow the existing

route to store food, and they will in turn reinforce the scent of that. If an ant is about to collide with the ant ahead, it moves slightly to the side and walks more slowly.

The ants typically move very quickly, guided by their acute sensitivity to the pheromones. If one ant stopped abruptly, it would hinder the progress of the ants marching behind it. According to biologist 'Iain Couzin' [Cou09], ants turn off the path every time they are about to collide. Outbound ants automatically turn far more than ants returning to the nest with food. This reduces congestion on the trails and may also be a way of giving priority to those bringing food to the colony.

This idea can be applied on multi robot USAR systems. Here robots move inside a confined area searching for trapped victims or hazards and when they locate any they return along, follow their own path to report their findings. Other robots that are in danger of colliding with and hinder the return of robots and their important findings, should automatically move out of the way to give priority to the returning robots.

5.7 Kidnap Recovery

The kidnap robot problem or re–localisation problem (e.g. catastrophic localisation failures) occurs when the robot has generated a false belief of its most likely location, which must then be identified before it can re–localise. The kidnapped robot has significant issues with its localisation system, and only a subset of localisation algorithms can successfully deal with the uncertainty created. We assume this problem arises when the robot is located in an environment of which it has a partial or complete occupancy grid memory. For instance, if the kidnapped robot moves from point A to point B with changes in its direction, at a given moment and based on sensory information, it has to decide where it is located within its occupancy grid memory. In the kidnapped robot case, the robot thinks it is lost because its sensor readings temporarily do not seem to fit with its notion of where it is and it therefore has to update a sharply peaked distribution.

A basic technique to deal with this problem in robotic is to let the kidnapped robot wander until it encounters a familiar scene or a landmark. A lot of effort has been invested in solving this problem during the last few years by using different tools such as Monte Carlo simulation [FBDT99], graph theory [BNRDW00], search in interpretation trees [CT00], RANSAC [NT03], or string matching technique. The last technique takes panoramic pictures of the environment by means of a mobile robot, and then extract features that are converted into strings [GBC04] and topological representations [CN01].

The DAEA algorithm applied on our agents inside life safe voids does not consider any land marking since the dark voids with their maze of obstacles makes this approach impractical. Instead we use the agents belief vector. It will continue to compare the belief vector, which is its recorded position including the uncertainties, to the new observed position and try to match them. The introduction of new samples from the transition function means that the process eventually should converge. Unfortunately, the time it takes for such convergence to materialise may be unacceptably long.

Since our agent has no image camera, the most practical solution is to leave some tracks as landmarks. Therefore we consider an agent that is programmed to follow the internal walls of a search field with a predefined step limit of Peb_{steps}. When they reach the Peb_{steps} limit, they sit next to the wall

that they were following. The kidnap agent can now find its way to the wall according to its belief, try to reach to one of the static agents and re–localise. We will evaluate the recovery procedure in Chapter 6.

Wake–up robot problem is also a special case of the kidnapped robot problem. The wake–up robot knows that, it has no idea where it is because its clock has just started and this knowledge is reflected in a uniform distribution over locations, which provides a good start point for localising.

5.8 Discussion

In this chapter we have introduced an advanced frontier–based exploration technique for individual agents. As the results indicate this technique performs closely to the oracle technique inside small spaces with a frame size of 33×33 cells.

The *EDAEA exploration method* is later developed to handle the problem of being stuck in local–minima that affects traditional frontier–based algorithms when facing positional uncertainties. An *EDAEA agent* performs sub–optimal actions in order to acquire a better model of the opponent and this will generate a better utility in the long run. However, in a large number of state cells, e.g. 50×50 cells, EDAEA agent will also lead to a poor state where even *optimal play* generates low utility, i.e. gaining knowledge become too expensive.

In this regard, for larger search fields, we divide the area among multi agents and apply the EDAEA technique for each subspace. The method used to divide a large search area among a team of multi–agent will discuss in chapter 6. Figure 5.6 is an overview of our multi–agent system. It shows how a team of agents is sent in, forming a chain, to explore a large search field. In pre–defined steps the last agent in the chain is released from the chain and proceeds to explore the local subspace by itself. This individual search technique is the basis for our EDAEA algorithm. When the subspace is explored and a local grid memory is generated, the robot returns to its mother, located at the start point, and report its recorded memory.

Chapter 6

Architectural Design of Multi Agent System

The field of single autonomous robot has now reached a point where practical solutions become available. Mathematically well–defined and experimentally proven solutions can be found to the problems of mapping [TBF00, EP03], exploration [BMS02, BMSS05], and localisation [FBKT00]. Quite a lot of experiments have been carried out in these fields and well–elaborated exploration algorithms for exact and detailed comparison exist. However, in the field of Multi–agent Systems (MAS), the situation is somewhat different even though MAS exploration performance might be highly superior to single agent performance in most of the robotic applications.

The improved flexibility and robustness resulting from the inherent parallelism of multi–robot systems gives them potential advantages over single–robot systems and the additional payload capability that becomes available when heterogeneous systems are introduced can significantly increase their ability to handle situations where the resource requirements are too substantial to be met by a single robot. However, the use of multi–robot also pose additional challenges that must be addressed if multi–robot technology is to present a viable and effective alternative to single–robot methodology. The most important challenges for MAS are:

1. The system complexity increase is associated with the use of several robots.
2. Most of the single robot strategies cannot simply be scaled to MAS.
3. Access to larger real world MAS or adequate simulation is difficult.
4. MAS are far more dynamic, simply because changes potentially can occur simultaneously in so many locations.
5. MAS can be self–defeating if agents collide or block the progress of each other.
6. Team collaboration and negotiation requires a reliable communication among the team members.

These challenges explain why only a limited amount of research deals with systematic experiments in the field of MAS, especially real world applications. In this chapter we introduce our collaborative multi–agent system. It consists of a team of agents that has distinct advantages over single robots, as well as other available MAS, with respect to sensing actuation. One of the important aspects in our introduced multi–agent exploration is that the team members are able to speed up the operation by *breaking the exploration area up into subspaces and distribute these among themselves.* A policy of formation progress is also defined for our MAS. This enables the system to keep its team members away

from each other and should result in the reduction of exploration redundancies.

In general, a team of collaborative agents can perceive their environment from multiple disparate viewpoints. In such a system, a single robot does not complete a task but instead by a team of collaborative robots. A single robot, on the other hand, can only sense its environment from a single viewpoint even when it is equipped with a large array of different sensing modalities. For robotic missions in an unknown environment, mapping, exploration, and movement are a coordinated effort (as discussed in Chapter 2). As our robots move, they collect information about the environment that surrounds them and from this information they will construct composite maps. The incrementally constructed maps emerging from this collaboration will in turn provide clues about the most promising areas of exploration and this will aid further path planning. The information will enable the robots to spread out inside the search field in order to speed up the exploration while using less energy. It will also help them to keep in touch with each other (directly or indirectly) in order to maintain their exchange of local memory grids.

As discussed in Chapter 2 our agents have a direct visual interaction. For instance, when they are performing wall following they have implicit visual interaction among the team. When there is no implicit communication among the agents, agents are not able to interact with each other at all time. However, since they are in the same environment, they can have explicit communication every time they are within sensor range of each other. Both types of communication techniques (`IMPLICIT-COM` or `EXPLICIT-COM`) allow an agent, whenever it is deemed necessary, to influence the behaviour of another agent or be influenced by it.

In this chapter we also introduce our *TeSLiSMA* exploration algorithm for self–organising MAS, which is a combination of several exploration tasks. Agents are able to change their behaviour by switching among these tasks. A set of logical conditions allows the agent to determine when it is advisable to switch to what behaviour. All the agents are fully autonomous and decentralised at the task levels. They receive task assignments, which they transform into sequences of behaviour using their predefined interpretation and planning capabilities. Tasks are therefore a temporal and logical composition of elementary behaviour able to achieve an objective in a context dependant way. It is also possible for our MAS robots to switch to a predefined corrective behaviour in a case of unsuccessful execution. The process of merging local, often overlapping, memory grids into a single global memory grid that provides a more comprehensive view of the environment is an important task for our MAS. Building an occupancy grid memory using a Bayesian update rule performs this task. [Mor88]. The process we propose, to create both the local memory grid and the global memory grid, is based solely on the direct communication among agents. It involves no communication with their mother. The system is therefore completely decentralised and does not require human intervention, which is why it has been introduced as self–organised exploration MAS.

6.1 Emergent and Self–Organised System

Generally speaking, a MAS is a complex system consisting of several locally interacting agents performing very simple tasks. Their interactions can nevertheless lead to a macroscopic, a *system–level* behaviour that has not explicitly been designed into the behaviour of the individual agents. Such a col-

lective conduct is referred to as *emergent behaviours* or *phenomena* [MM01]. These emergent properties cannot be directly inferred from the local behaviours of individual agents, due to complex interactions among the agents. Thus our agents are simply following the local rules that guide their movement inside search fields. However, as the collective behaviours of individual agents, the system will exhibit a complex behaviour, which will considerably reduce the time it takes for covering a maze–like complex search field.

We will here refer to the global, system–level properties as Macro–Level and the individual, agent–level behaviours and performances as Micro–level. A MAS consisting of simple and limited–function individuals can only accomplish complex tasks autonomously if it is able to exhibit *emergent behaviour* [WH05]. Steels [Ste95] pointed out two advantages of emergent behaviour for autonomous multi–agents when compared against directly programmed behaviour:

- No additional structure is needed inside the whole system to get additional capabilities. Therefore, there is no need for any centralised decision making and additional explanations for a special behaviour.
- Emergent behaviour tends to be more robust because firstly it is less dependent on accurate sensing or action and secondly it makes fewer assumptions about its environment.

Emergence without self–organisation is certainly possible for most of the systems. In some systems the interaction among individuals are sufficient to exhibit emergent properties and through this behaviour the system is directed towards its goal. However, if we allow our agents a large number of states and also allow them constantly to switch between these states, the result can be chaos. Therefore, it is important to make sure the system is able to organise itself and develop coherent behaviour at the macro–Level and promotes the required functionalities.

A self–organising system [SWGC04] is defined as a system with members that are able to organise themselves *without external direction*, or control. Collaboration and group formation are key factors when creating such systems. Additionally, it maintains *functional structure* through its individuals. A well known example of such self–organised systems is the ad–hoc networks where individual members of the system collaborates to create and maintain its own network.

The term ‘functional structure’ is referring to the division of tasks in an organisation, which is grouped by the main functions that need to be performed within their organisation. Also, the term without external direction, refers to the absence of any direction, manipulation, interference, pressures or involvement from the outside of the system. Thus external inputs are ignored, as long as they are not macroscopic control instructions.

6.2 Tethered Search of Limited–Sensing Multi–agent (TeSLiSMA)

The self–organised and emergent multi–agent exploration technique introduced in this thesis is the **Te**thered **S**earch of **Li**mited–**S**ensing **M**ulti–**A**gent **(TeSLiSMA)** algorithm. This exploration algorithm consists of macro–level and micro–level actions for MAS. The behaviour performed by the MAS using the TeSLiSMA exploration algorithm (appendix E) are:

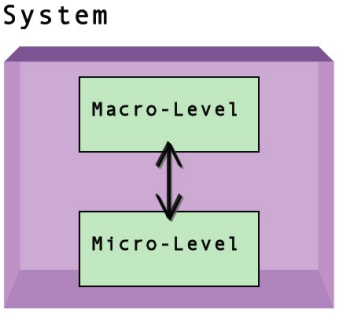

Figure 6.1: Self–organised and Emergent System

performHeadBehaviour Al. 11, *performFollowerBehaviour* Al. 12, *performTailBehaviour* Al. 13, *performIdleBehaviour* Al. 14, *brokenChain* Al. 16, *noChain* Al. 15, *moveForwardNextToWall*, *stuck* Al. 17, and *performEDAEA* Al. 6.

In order to explain the TeSLiSMA algorithm more intelligible, we arrange it into Macro–Level and Micro–Level behaviour:

i) Macro–Level behaviour: Chain–Formation, Wall–Following, and Implicit–Connection.

ii) Micro–Level behaviour: Obstacle–Avoidance, Stay–Away, EDAEA–Performance, and Return to–Mother, as illustrated clearly in figure 6.2.

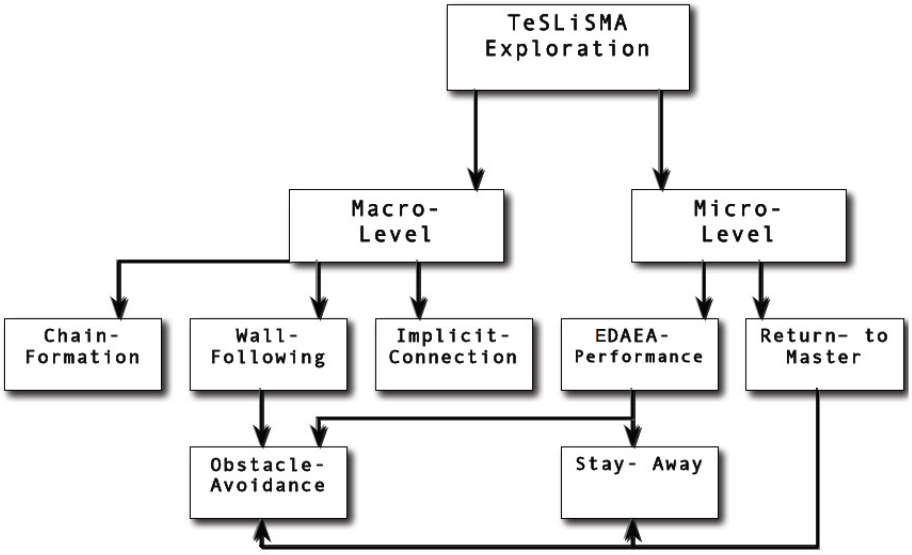

Figure 6.2: TeSLiSMA performance taxonomy

6.2.1 Macro–Level Behaviours

As discussed before the Macro level behaviours are:

a) *Wall–Following* (Algorithms: *moveForwardNextToWall*, *stuck* Al. 17): During this task agents should sit next to a wall and follow the wall to explore the search environment, they should look for trapped victims, while checking their *Implicit–Connection*.

b) *Chain–Formation* (Algorithms: *performFollowerBehaviour* Al. 12, *performIdleBehaviour* Al. 14, *noChain* Al. 15): Agents should form a chain while they follow the wall. This enables the team to spread easily inside the search field without blocking the paths of other agents (congestion avoidance).

c) *Implicit–Connection* (Algorithms: *performHeadBehaviour* Al. 11, *performTailBehaviour* Al. 13, *brokenChain* Al. 16): This task maintains the connectivity of the chain. The agents at the Head and Tail should be in line of sight of at least one other agent, while the rest of the chain members should have two agents in their line of sight. If not, they should try to reconnect the chain. In this task the agent that realise the chain is broken will attempt to reconnect it by moving to its previous position. It should inform the connected agents to follow it in this action. If the broken chain is reconnected during that one step, the agents will switch to the Wall–Following algorithm, otherwise they will either apply the EDAEA search algorithm (less than 3 agents) or they create a new chain and perform the wall–following behaviour.

During the wall–following procedure, a chain of multi–agents are moving next to a wall or pile of obstacles. The first agent to enter the search area is designated as the *Head* and will be located at the head of the chain. The head moves forward, and waits till it observes another agent in its line of sight. Followers, perform the same movement constantly and check to see if of two agents are available and the Tail checks for an agent in its line of sight. The key factor in the wall–following behaviour is to maintain the implicit connection among all members of the chain. The head continues to move close to a wall, all through our generated confined and cluttered internal walls till it reaches a victim or reaches back to the entrance (where the mother is situated). By the wall–following, the head is able to estimate the internal size of the search terrain and report it back to its mother (For mini robots with small `ENERGY-LIM`, performing wall–following we select a maximum frame size, i.e. safe frame). This enables the rescue team to automatically aware of their full coverage. Additionally, agents that are performing wall–following avoid obstacles by moving next to them, exactly like they move next to a wall and they avoid collision with other team members (all should be defined as their *low–level* behaviour).

The line formation is very useful when the team is passing through corridors or passing to some confined places [BS07]. Normally the head is chosen as the team commences the search. However, whenever it is deemed necessary, our chain is able to select another agent as the head of the formation. That means the head does not control the team, since the formation is decentralised. The agents are able to perform this procedure because they have the capability to interact visually and convey there states to other agents via their RGB LEDs. The association between the LED colours and states is indicated in table 8.1.2.

Figure 6.3, an example that demonstrates how agents squeeze inside a search field through a narrow entrance. Because it is possible for the head to get stuck in a narrow dead end, other agents (Followers and Tail) must have the option to reconfigure in order to redirect the chain out of the dead end. An example of this situation is shown in figure 6.4. Here, the head initially realises that it is stuck in a dead end and therefore turns its LED to purple (State=Idle). Consequently, the LED of the agent immediately

Table 6.1: Study of various states for team members by their defined states

State	Definition	*RGB LED*
Head	The first agent on the head of chain	RED
Tail	The agent at the end of the chain	YELLOW
Follower	All the agents on the chain following the Loop-Closer	GREEN
Lost	The agent that lost `IMPLICIT-COM` within its chain	BLUE
Filler	The agent performing DAEA exploration algorithm	OFF
Idle	The Head that has been stuck	PURPLE

next to it in the chain becomes RED as it takes over the job of head. It sees the previous head as an obstacle that should be avoided. Now, if it has a free cell available, it starts to move the chain in the usual manner. However, if there are several agents trapped this process, the job as head is passed along the line until it reaches an agent that is able to move. Then, as the agents trapped in the dead end start to move out, the *idle agent* will find a free space in its line of sight and follow the chain as its tail.

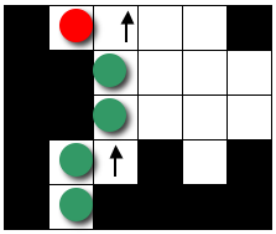

Figure 6.3: Wall–following behaviour inside a search field

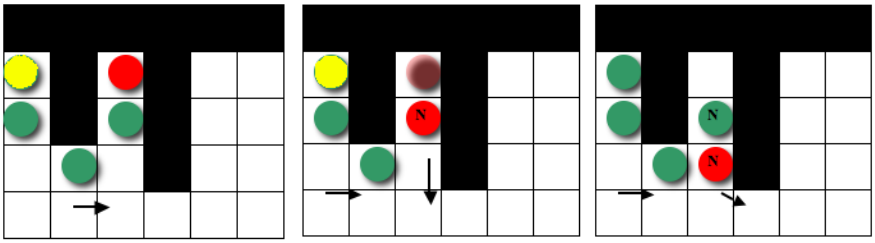

Figure 6.4: A Head is stuck while following a wall (left to right)

6.2.2 Micro–Level Behaviours

performDAEA: After reaching a predefined number of steps, the tail of the chain will sever its connection with the chain and change its state to “filler”. Another will immediately replace its status as tail agent. Under the DAEA search algorithm, Fillers should fill their predefined local occupancy grid memory. They are able to increase the frame size of the local memory when for instance they are in an area with a high density of obstacles. This is part of the provided predefined corrective behaviour that can be launched in a case of unsuccessful execution.

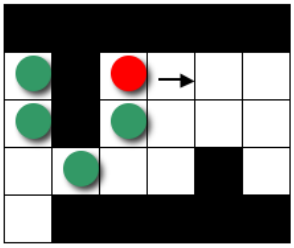

Figure 6.5: Wall–following behaviour while keeping the IMPLICIT-COM

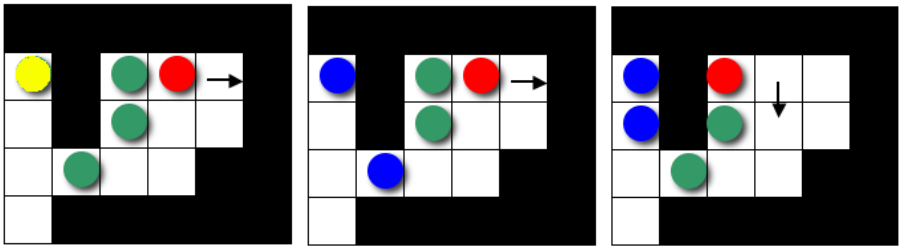

Figure 6.6: A Follower loses its connection with its chain (Left to right)

As illustrated in 6.7 , a follower should turn its LED off whenever it changes its state to filler. Thus the agent ahead of the filler immediately recognise that it should replace the tail, and not report the chain has broken. Other essential micro–level behaviour for all kind of mobile agents like collision avoidance, both obstacles and other agents, are also developed for all agents.

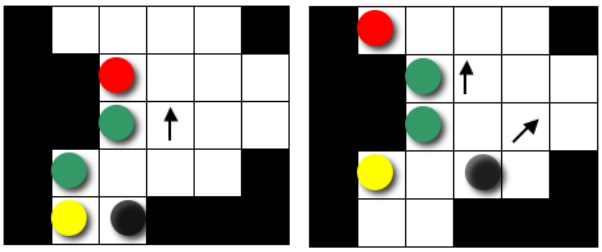

Figure 6.7: Filler leaving its chain (Left to right)

In the next section we define how a chain of agents, following a wall, is able to create a local memory grid of the subspace inside a search field.

6.2.3 Local Memory Grid Generation

Teams of agents can divide an area into subspaces and assign each subspace to two or more agents. In this regard each agent has two matrices to fill in. One when it performs wall–following and the other when it performs DAEA algorithm:

- A $(M \times N)$ matrix grid memory, which is expected to fill with information as to whether cells are occupied by obstacles or not according to its sensor readings (we refer to $M \times N$ grid memory as the search terrain interior memory). If the area is larger than its grid memory, the head creates a virtual wall and avoids going beyond it. However, inside its interior grid memory it will distinguish

the real walls from virtual walls.

- A $(M' \times M')$ matrix grid memory (where $M' < M, N$), as agent's predefined local memory for each subspace.

As discussed in Chapter 5, in occupancy grid memory the environment is divided into homogeneous cells, we set a probability of occupancy for each grid cell. This probability is stored in the matrices by each agent, at each step. An occupancy value of zero corresponds to a free cell and a value of one corresponds to a cell occupied by an obstacle. Initially all the available cells inside a predefined matrix memory grid, for all agents, are assigned a value of 0.5, i.e. equally likely to be occupied or free.

The agent assigns its local memory grid as: $X_{robot\ position} = X_f - x_0$ and $Y_{robot\ position} = Y_f - y_0$, figure 6.8. Where $(X_{robot\ position}, Y_{robot\ position})$ is the start point for our agent as the 'Filler'. x_0, y_0 should be set for the robot, in all our experiments both are equal to 6.

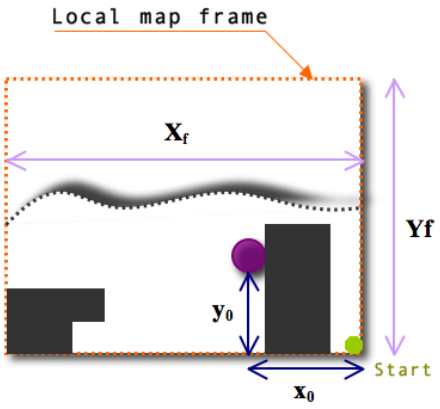

Figure 6.8: A *local memory grid* (X_f, Y_f) assign to an agent to explore a subspace

We now go through the memory grid creation process in more details. Every 'nns' steps 'nnr' of agents (known as *Fillers*) are released from the end of the chain and dispatched for extended exploration. They start to fill their own internal memory grid frame, by exploring the search field according to the EDAEA exploration strategy. Both the Head and the Fillers mark the release point, as the merging point inside their grid cell memory and when the filler has filled its grid memory it returns to the extended exploration point (start point) through the recorded paths. It should report back its local memory grid to the base station (mother robot) at the start point. Since we assume that all mini agents are released by their mother at the entrance to the void, as discussed in Chapter 2, the loop will be closed when the head meets its mother and, after unloading its internal memory grid frame, it changes its state to filler. As filler, it continues to perform EDAEA searches to fill its local grid memory with additional information.

There is a possibility that a head may not be able to go back to its start point and thereby close the loop. One example of this is shown in figure 6.9 where the narrow entrance leads the chain of agents to a dead end. Under such circumstances the head should inform the chain that it must reverse back to the start point. To accomplish this it changes the colour of its LED to blue and rotates its heading 180 degree. When the neighbouring agent sees LED colour change in the head, it changes its own LED to

red and reverse the direction of its heading the same way, waits ω seconds and then changes it's LED to blue. This process continues down the chain until the last agent, which has only one agent in its line of sight. The LED of this agent remains red and it now starts to follow the wall back to the starting point.

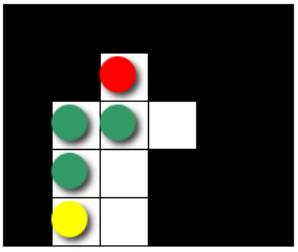

Figure 6.9: An example of a chain stuck inside a narrow pathway

In order to deal with odometry errors in real scenarios, we need to add an extra feature to our present filler behaviour. Figure 6.10(a) illustrates an agent that is filling its local memory grid. The correct occupancy memory grid for the agent is shown in figure 6.10(b). However, if the agent has odometry errors due to slippage, it could fill its memory grid as shown in figure 6.10(c) and therefore end up completing a false memory and missing parts of the search area. To prevent this from happening, the agent should perform a wall following procedure before it resorts to *automatic filling*. Automatic filling happens when the agent closes a wall inside its local memory and consequently starts to fill the sealed area with data indicating it is occupied (Agent changes behaviour from performEDAEA to automatic filling whenever it meets a wall or virtual wall). This makes the agent think its memory grid is completely filled and that it therefore should terminate the exploration. Therefore, when the agent has closed an area inside its memory grid it should perform a wall following procedure around it to make sure that no entrance remains (a very similar technique to loop–closing in SLAM [EP03]). If the area actually is sealed, the agent will fill it with 1s, otherwise it will enter the sub–void for further exploration.

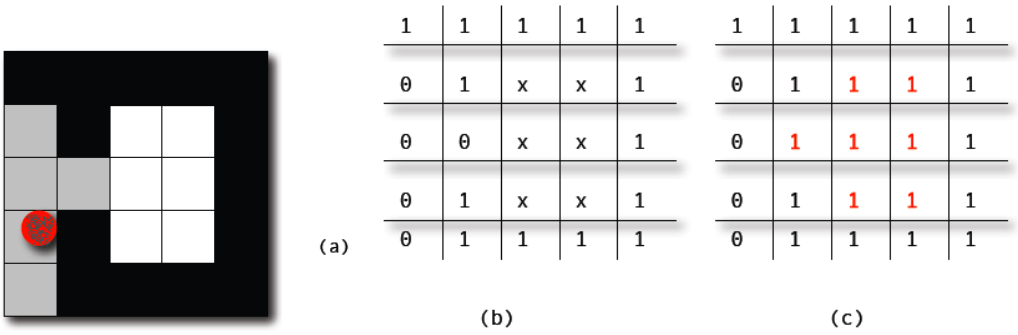

Figure 6.10: (a) : local memory grid, (b) : true local memory grid , (c) : false local memory grid

When the agent starts to search its sub–region, it assumes that a virtual wall indicates a border around it and cannot go beyond that. As illustrated in figure 6.11(a, b) these borders can potentially create confusion for robots because they are unable to apply wall following around it and fill the sealed area. The agents therefore fill such cells with the value 0.75 as shown in figure 6.11(c). This indicates

that the probability that these cells are occupied is 75%. However, the mother robot still needs to decide if such cells are occupied or free when it merges local memory grids through Bayesian update. Extra data must therefore be provided from another agent covering the area on the other side of the virtual wall.

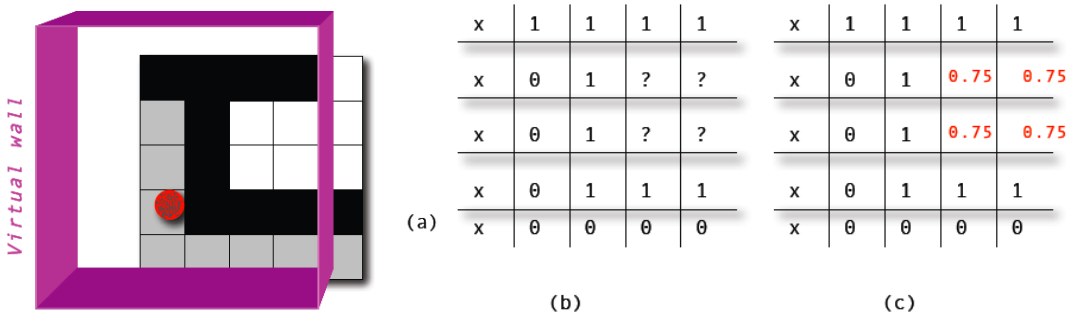

Figure 6.11: (a) : local memory grid filling, (b) : cell status are unknown , (c) : filled by 0.75s

6.2.4 Multi–Agent Spreading Techniques

Filler agents start to fill their $M' \times M'$ local grid memory. The start point for the filler is the merging point where it is disconnected from the chain. The mother should merge the memory grids provided by the agents into a global memory grid. After merging the local memory grids, it should select attractive cells for additional investigation and, after a battery recharge, dispatch the agents back into the area for further investigation with the global memory grid of the explored environment loaded into their memory. This will allow the new team to continue the search towards full coverage using a topological memory grid of the environment built on top of the grid based, occupancy memory. Such a topological map will guide the team towards the attractive points much faster, and from there it can continue to fill in more local memory grids.

Table 6.2: Memory Grid Filler parameters

Variable	Definition
NF_{max}	the maximum length of the frame size of the search field
tna	Total number of agents that has been sent inside the search field
nns	Number of steps taken by the Tail in the chain
nnr	Number of agents from the chain has been sent inside as a filler
ana	Number of available agents in the chain of robots
M, N	Interior memory grid lengths, i.e. frame size of $(M \times N)$ matrix
M'	Local memory grid length, i.e. frame size of $(M' \times M')$ matrix i.e. total number of the cells that is occupied.

Two techniques are used in this thesis to spread filler agents:

Technique 1- dispatches two agents after nns time steps. One of these is a Filler and the other

one the Head. We always assume that $NF_{max} \geq 50$ and $tna < 100$, so $2 \times NF_{max} > tna$ and also $nns < M'$, thus we have:

$$2 \times NF_{max} \times M' > tna \times nns \tag{6.1}$$

$$2 \times NF_{max} \times M' = k \times tna \times nns \tag{6.2}$$

Experimental results indicate the best k is selected as $k = 12$.

$$nns = \frac{NF_{max} \times M'}{6 \times tna} \tag{6.3}$$

We define a minimum number of steps for our multi–agent system when $M' = 33$, i.e. $nns = 20$, to avoid any congestion inside a search field. If '$20 < nns < 33$', thus :

$$\frac{NF_{max} \times 33}{120} > tna > \frac{NF_{max}}{60} \tag{6.4}$$

Technique 2- let the head release nnr agents every time it reaches a point beyond its local memory length (M'). One of the released agents becomes a head and the rest will remain followers. They create a new chain of agents for further exploration.

$$nnr = \frac{ana}{2} \tag{6.5}$$

6.2.5 Local Memory Merging

When fillers report back their local memory grid, their mother merges the information from the provided memory grids. Since all the local memories were built using an occupancy grid, the mother can apply a simple *Bayesian update rule* to merge several memory grids. The mother applies Bayesian update rule [BP02], [Sta06]on all provided occupancy grid memory.

$$\mathbf{Pr_{Occ}}(cell|Se^{(1)}, \ldots, Se^{(t)}) = \tag{6.6}$$

$$1 - \left(1 + \frac{\mathbf{Pr_{Occ}}(cell|Se^{(t)})}{1 - \mathbf{Pr_{Occ}}(cell|Se^{(t)})} \times \frac{\mathbf{Pr_{Occ}}(cell|Se^{(1)}, \ldots, Se^{(t-1)})}{1 - \mathbf{Pr_{Occ}}(cell|Se^{(1)}, \ldots, Se^{(t-1)})}\right)^{-1}$$

Equation 6.6 [Mor88] updates the occupancy probability for each cell, $\mathbf{Pr_{Occ}}(cell|Se^{(1)}, \ldots, Se^{(t)})$, this is based on the current sensor reading, $\mathbf{Pr_{Occ}}(cell|Se^{(t)})$, and the a priori probability, $\mathbf{Pr_{Occ}}(cell|Se^{(1)}, \ldots, Se^{(t-1)})$, where $Se^{(t)}$ is the sensor readings at time t (current time). If we can convert the output from a sensor into the probability that a particular cell is occupied or not, it is therefore possible to merge this information into our global memory grid. Through this technique, mother is then able to generate a composite memory of a search field.

When a mother is merging the local memory grids there is possibility to find a memory grid unreported. That is because the filler is stuck inside its subspace and not able to report to its mother.

However, the mother is able to identify which agent in which subspace has been stuck. In this regard more sophisticated agents are required to send in, for that subspace, for further investigation (e.g. agents with on–line image camera).

Thrun and *Bucken* [TB96] have presented a technique that enables us to extract a topological map from the global memory grid that we have generated. They use thresholding, voroni diagram, critical points, critical lines, and finally generating a topological graph. The key idea of the above technique is that the free space of an occupancy grid memory grid can be partitioned by critical lines in to small number of regions. Critical lines correspond to narrow passages such as doorways. The partitioned map is then mapped into an isomorphic graph. Since all our local memory grids have already assigned values of 0 and 1, we can skip the thresholding and continue to Voronoi diagrams. The technique is illustrated by the example in figure 6.12. Using the resulting topological graph map the mother is able to estimate the optimal path the mini robots should follow to reach the selected attractive point. For instance, we can implement a standard search, such as *Dijkstra's Algorithm* [ZN98], to find a 'best' path, which is a subset of the Voronoi diagram. The method generates a route that gives the robot a relatively safe path along which to travel to reach its goal.

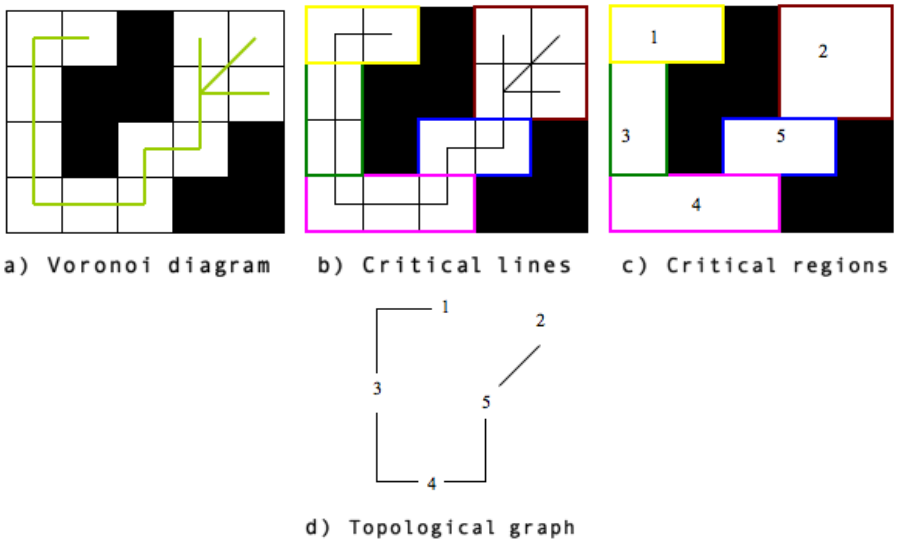

Figure 6.12: Extracting the topological graph from an example occupancy grid map

6.2.6 Spreading Techniques Evaluation

In this section we evaluate the performance of our MAS in relation to full coverage of the search field. In this experiment we select 72 different search fields with frame size of 50×50 cells. We assume that the indoor search field is sized from outside, so we have a rough estimation of the `MAX-SIZE` of the search field. We set up two experiments, first by applying *Technique 1* to send fillers into a search field. Second we apply *Technique 2* and compare their results. Furthermore for each experiment we test the performance of our multi–agent system with two different numbers of agents within their team.

As discussed before, all the filler agents have a local memory grid with fixed frame size to explore. Since we want to test the MAS performance in one experiment we add an extra feature for these agents.

During the exploration an agent may have been allocated a subspace that has only a very limited amount of free space to explore. This would lower its performance and we therefore require that every agent fills more than 50% of the available cells in its predefined local memory before it returns to its mother (automatic filling is excluded). While the number of explored cells remains less than half the total number of cells inside the predefined memory grid, it should instead expand its local memory grid by adding 10 cells to both the width and height of its frame size. It then continues to explore this extended subspace. This increase in frame size can be performed several times until it manages to meet the minimum requirement. This feature enables the team to attain full coverage faster. The rules can in practice be implemented as the number of grid cells explored before agents can return to their base station to inform their local memory grid. Therefore they are able to perform the second stage of the exploration only in those areas that require further investigation.

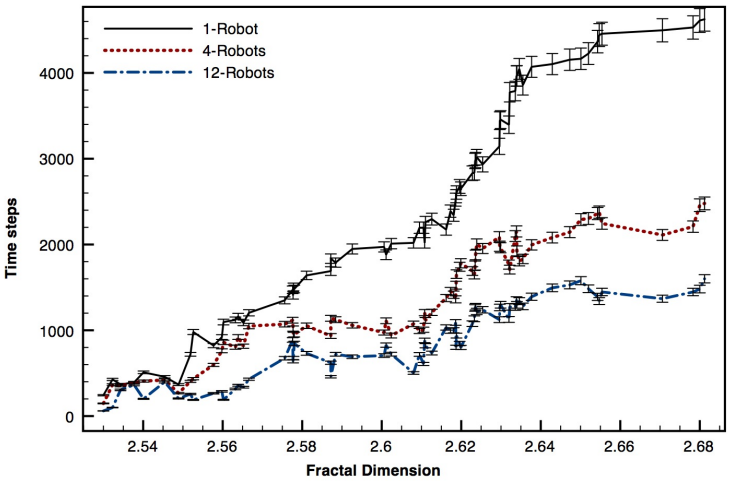

Figure 6.13: Average coverage time steps for TeSLiSMA exploration Al.-Tech. 1, 50×50 cells

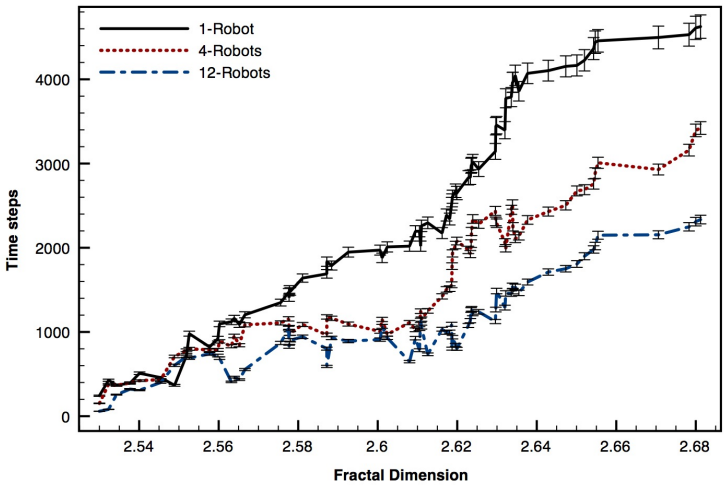

Figure 6.14: Average coverage time steps for TeSLiSMA exploration Al.-Tech. 2, 50×50 cells

Figures 6.13, 6.14, and 6.15 are plotted with the corresponding standard deviation bar. In figure 6.13 we compare the performance of a single agent using only the EDAEA algorithm to MAS teams of 4 and 12 agents using the TeSLiSMA algorithm. The agents are using technique 1 as discussed above. The results from a similar set of experiments using technique 2 are shown in figure 6.14, where we observe very similar results. However, in more complex environments, technique 1 is able to cover the search field faster than technique 2. If the mother that releases its mini agents is not able to estimate the approximate size of the search field from outside the spreading technique is restricted to technique 2 since this technique only requires knowledge of the number of mini robots inside a chain. Otherwise, it should always use technique 1.

There is one more approach, *Technique* 3, for spreading agents inside a search field. If $\frac{\text{'External area'}}{ana \times M'^2 \times 10^{-2}} \geq 1$, where the *External area* is the approximate size (m^2) of the field from outside. we can apply Technique 3 for our system. This involves 'nnr' fillers (from technique 2) being released from the end of the chain every 'nns' (from technique 1) steps.

6.3 Exploring a Large Area with One Team

In the next experimental set up we investigate how well our MAS is able to cover an area as large as $100m^2$. This is an area 10000 larger than the size of our deployed agent.

In this experiment all the defaults value are as described in previous experiment, except for the search field frame size ,which is now 100×100 grid cells. In contrast to previous experiment, we have not here compared the MAS performance to that of a single agent since one agent takes more than 18000 time steps to cover areas with complexity of $FD > 2.67$. From the results of this experiment, as indicted in figure 6.15, we estimate the average time will take 30 agents to search an area of $100m^2$ to be approximately 12 hours. However, in practice, when each agent fills its local memory grid, it will return to report to the mother instead of extending the frame size. Thus, if further teams of agents are required, they will be dispatched for further investigation, according to the generated global memory grid.

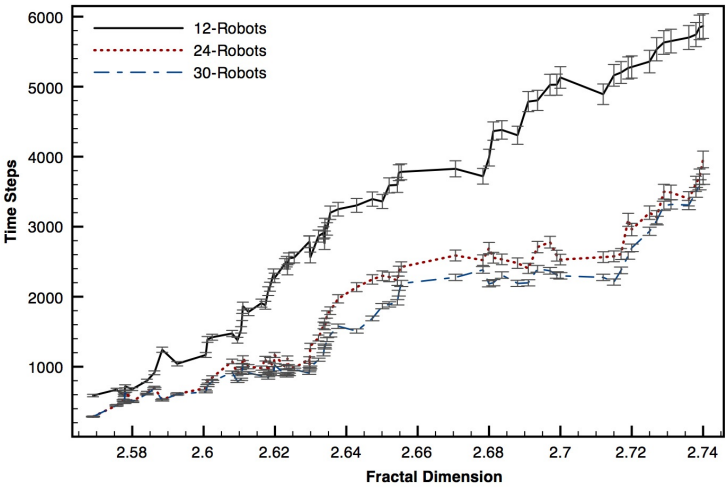

Figure 6.15: Average coverage time steps for TeSLiSMA exploration Al.-Tech. 1, 100×100 cells

The operation may be made faster if the rescue team could find another entrance for agents on other sides of the debris, send in additional teams from those entrances and merge all the local memory grids. This would enable the rescue team to speed up their operation and find better routes for the rescue team to dig for trapped victims. In this regard, to explore an area such as a large warehouse, e.g. $1000m^2$ (100000 times larger than our mini robot), it is not possible to use only one entry and apply the TeSLiSMA algorithm, since the mini robots have very limited battery power and some of these agents will have no battery power left to continue when they reach their start point as the filler. Thus, we need either to search for multiple entrances from different directions of a pancake area or the rescue team need to start digging and create access every $80m^2$. This would enable our mini robots to continue their search operation. Moreover, the potential located victim(s) will be rescued much faster than if they were to wait for the full coverage of a large area.

6.3.1 RF Communication

As discussed in Chapter 2, *RF* communication inside the structural collapses is not a reliable technique for agents to interact. However, we can assume that every time this communication link is available the agents can take advantage and update their local memory grid according to data collected by other agents. This significantly speeds up the exploration operation for MAS (especially inside a large and very complex search area) when a sufficient number of mobile or fixed agents is employed [FT09], [FTL09], [KL09].

A mobile agent will here send state queries to agents in its communication range, and from their respond it is able to update its local memory grid according to the Bayesian update rule every time the RF communication link becomes available. In addition we are able to define a Negotiation Algorithm that will allow the agent with more power source to be selected to continue for further investigation. For instance when more than two fillers meet, which it can establish a RF communication link, the agents change their state to Link. In this state one agent continues its exploration as filler (the one with less battery power) and the other two agents one as the Head (the smaller ID) and the other as the follower, thus they are able to create a chain.

6.3.2 Pebble Tracking

We have introduced Pebble–Tracks, in Chapter 5, as agents that act as a static node inside a search field. This enables kidnapped robots to re–localise themselves when they reach these tracks. For every Peb_{steps}(equ: 6.7), a Pebble–Track will be left next to the wall, while the chain of agents continues its wall–following progress. Thus a kidnapped agent can always use its belief to find its way toward a wall and subsequently move along the wall until it reaches the Pebble–Track and is able to localise itself according to the pebble. In addition, if the kidnapped robot meets another member of its team and they are able to establish RF communication, it can localise itself according to the fellow team member since they are from a homogeneous set and both have the same start point. The mother that has released the agents at the entrance consider that 20% (i.e. set as the best number according to experimental results on the search fields equal and smaller than 100×100 cells) of the agents inside the chain will be used as Pebble–Tracks number of agents, $pna = 20\% \times tna$ and will set their state to 'Pebble'.

$$Peb_{steps} = \frac{3 \times Ns}{pna} \tag{6.7}$$

We set up an experiment with 20 different search fields of frame size of 50×50 and $FDP > 2.58$. Randomly we select a time limit t_{limit} (i.e. $\{15, 25, 35, ..., 145\}$) for our search agents within a team (Pebble-Tracks are excluded). When the agent reaches that time limit, automatically 10 last recorded data are deleted from its memory. The kidnap agent should localise itself by the help of available information distribution and Pebble–Tracks.

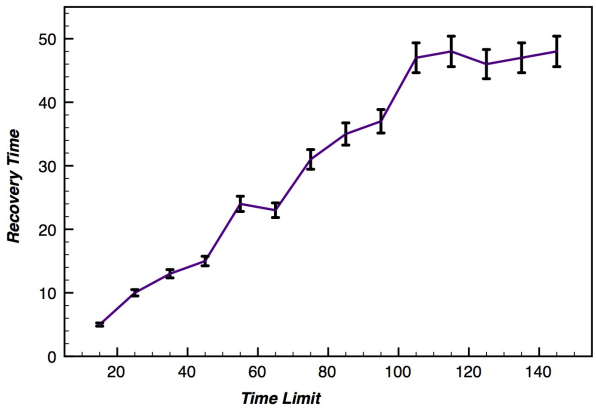

Figure 6.16: Recovery time steps for a kidnap agent, search fields 50×50

Figure 6.16 is illustrating the average time for a kidnapped agent to re–localise. The recovery time is larger for the filler agent (filling its local grid memory). However, after increasing the time limit (125), where almost all the agents are spread inside the search field an performing as the filler, the average recovery time remains unchanged (an average of 47 time steps (28 minutes)). From the result the maximum time step for an agent to re–localise is 54 steps that is equal to 32 minutes, which is counted as an acceptable time.

6.4 Area Percentage Coverage

Although complete coverage is considered an important metric for exploration procedures, performance measures such as percentage coverage are often neglected in search and rescue operations. Through this metric we are able to see how well the team of robots is able to spread its agents inside the search field.

Here we first consider a frame size of 50×50 and three environments with three levels of complexity, $FD_{min} = 2.53 < FD_{av} = 2.61 < FD_{max} = 2.681$. We then test the coverage percentage of a team of 8 agents. Secondly, we consider a frame size of 100×100 and three environments with three level of complexity, $FD_{min} = 2.56 < FD_{av} = 2.64 < FD_{max} = 2.73$ and test the coverage percentage of our team of 26 agents.

60% of all the search fields are covered in less than 530 time steps, which correspond to a search time of less than 6 hours for our millibots. This is a very desirable timing for a limited number of very small agents, i.e. $100mm$, to explore areas ranging from $25m^2$ (which approximates average size for a

room in a residential area) to $100m^2$. This proves how well our team is able to spread its agents inside a search field.

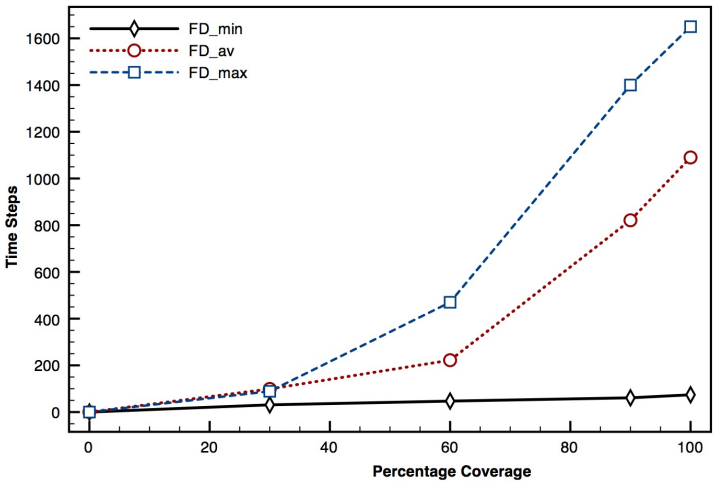

Figure 6.17: Percentage Coverage of three level of complexity for search fields, 50×50

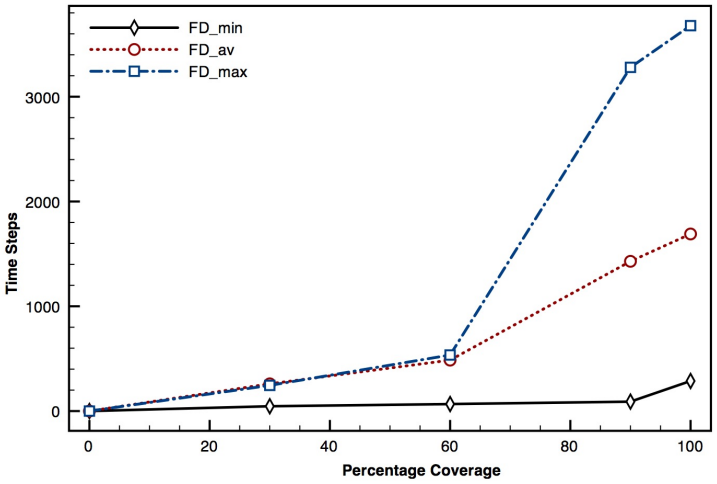

Figure 6.18: Percentage Coverage of three level of complexity for search fields, 100×100

6.5 Exploration Efficiency Coverage

Another important metric is the exploration efficiency coverage. We here create an experiment to compare exploration algorithm efficiency of the TeSLiSMA against that of the TREE exploration algorithm. This experiment indicates how well is our collaborative MAS performing when compared to a well–known collaborative multi–agent methodology. As discussed in Chapter 4 we define exploration efficiency coverage as $Q_{EE} = \frac{A_{Ac}}{\sum_{i=1}^{tna} d_i}$. In this experiment we approximate this value as $Q_{EE} = \frac{A_{Ac}}{tna \times rt}$. Where rt stands for required time to cover the search field completely, 100% coverage. Furthermore, we estimate the system coverage rate defined as $R_{SC} = tna \times Q_{EE}$ as well. Experimental defaults are as

follows:

– A frame size of 100×100 with average level of complexity, $FD_{av} = 2.65$ and total number of agents send inside the search field '$tna = 24$'.

Table 6.3: Experiment data, frame size:100×100

Exploration Algorithm	Exploration Efficiency Coverage Rate	System Average Coverage Rate
TeSLiSMA	$Q_{EE} = 2.48\ m^2\backslash hour$	$R_{SC} = 59.52\ m^2\backslash hour$
TREE	$Q_{EE} = 1.37\ m^2\backslash hour$	$R_{SC} = 32.88\ m^2\backslash hour$

– A frame size of 50×50 with average level of complexity, $FD_{av} = 2.61$ and '$tna = 8$'.

Table 6.4: Experiment data, frame size: 50×50

Exploration Algorithm	Exploration Efficiency Coverage Rate	System Average Coverage Rate
TeSLiSMA	$Q_{EE} = 2.61\ m^2\backslash hour$	$R_{SC} = 20.88\ m^2\backslash hour$
TREE	$Q_{EE} = 1.24\ m^2\backslash hour$	$R_{SC} = 9.92\ m^2\backslash hour$

As the results in the above tables indicate, our MAS is far more efficient than the traditional collaborating TREE system [MT04].

6.6 Discussion

After the mother has merged all the local memory grids that has received from its agents and created a composite memory grid of the search field, the rescue team should check the merged global memory grid. They need to compare the size of this composite memory grid to the size of the search field frame size. If the memory grid points to passable borders and the main frame size is larger than the composite memory grid, which happens when less than 95% of the passable search environment is explored, more agents should be sent inside the exploration field to continue the exploration procedure. This is when $0.95 \times \texttt{MAX-SIZE} \leq Size_{\text{Global Memory Grid}}$. However, the agents used for further explorations will receive a topological global map of the explored environment. Thus they are able to find their way through the search field much faster and reach the points of interest from where they continue their exploration as discussed in this chapter. Therefore, when these explorer agents are dispatched, we are able to send a team of agents for further exploration with one used to mark the start point and the others as a chain of agents.

.

Chapter 7

Multi Agent System Evaluation

In this chapter we will describe an experiment aimed at testing how well our team of agents perform compared to teams using other multi–agent techniques. We will compare the performance of several competing algorithms such as ANT, TREE, EANT, Hybrid B&M, and TeSLiSMA when they are applied to limited sensing multi–agent systems. After focusing on the advantages and disadvantages of our TeSLiSMA algorithm, we will present some thoughts regarding future work. The introduction of an intelligent monitoring technique is urgently required to test and monitor the exploration algorithms. The ideal monitoring in this regard is using the same kind of agents, same size and capability, as well as to study the behaviour emergent from the exploration agents as they perform their search task. They are able to recognise undesirable emergent behaviours much faster and easier than human tester. It is also possible that they could end up adding previously unrecognised emergent behaviour and thus improve the MAS performance.

According to Eric Berger, co–director of personal robotic program at Willow Garage [Tea], if we want to develop the robotic system equivalent of Twitter, i.e., available for all web users, we do not start by constructing a computer: "We build the thing that is new". Therefore, although we have developed our own simulation environment to expedite our testing and comparison process, we should be able to apply our novel exploration technique on existed robots as well as, well known, available software interfaces. At the end of this chapter we attempt to prepare TeSLiSMA so it can be applied to real robots through the Player interface. We suggest future work that could improve the search performance of TeSLiSMA, especially when searching very complex fields.

7.1 Multi–Agent Exploration Comparison

The default values in our experiments are: a frame size of 50×50 squared grid cells with 70 level of complexity. We also send in 10 agents to perform different exploration algorithms. In this experiment we compare the discovery time for TeSLiSMA against the discovery times for other competing algorithms that can continue with their performance with the presence of odometry errors . In this experiment 6.6% odometry error is included.

The ANT algorithm used in this thesis suffers several limitations. In addition to its redundancy, the ANT agents are not automatically aware if they have managed to cover the entire search field. In

practice they are therefore forced to continue the exploration phase until they run out of energy and this makes them unsuitable for time critical operations like USAR unless there are equipped with extra functionalities like records of explored cells. We therefore introduce the Extended ANT (EANT) exploration algorithm, which differs from the ANT algorithm by using a weight factor for each vision segment. An exploration agent, using the EANT algorithm, calculates a *vision segment weight factor* at each time step for each of the eight segments around it and, using Algorithm 9, it selects among the free segments the one with minimum weight. In an attempt to avoid revisiting explored areas as much as possible, the weight of the most recent selected angles is increased by applying our weight function, as defined in the equation 7.4.

Table 7.1: Weight Factor parameters for agent segmented vision

Variable	Definition
$angle_t$	agent's angle $\in \{-180, ..., +180\}$ at time step t
$angle_p$	agent's predefined angle $\in \{-22, ..., 0, ..., +113\}$
N_{θ_i}	number of times that segment i is visited by the agent
W_{t_i}	weight factor at time t for segment i
NP	Number of steps that agent follows $angle_p$
Ns	Number of steps taken by the agent
i	$i = \{1, 2, 3, 4, 5, 6, 7, 8\}$ respectively represent $\{NE, N, NW, E, SE, S, SW, W\}$

$$W_{i_t} = \frac{N_{\theta_{i_t}}}{Ns} \tag{7.1}$$

The Exponentially Weighted Moving Average (EWMA) [Wika] is a form of average that weights data and decreases the applied weight as the time pass. The formula for calculating the EWMA at time $t > 2$ is:

$$EWMA = \lambda L + (1 - \lambda) W_{i_{t-1}} \tag{7.2}$$

where L is an observed value of the variable measured at a given point at time t, and λ is a constant between 0 and 1 that determines the inertia of the EWMA. The closer λ is to one, the more weight is given to a current observation.

EANT– The EANT exploration algorithm uses the EWMA function *to reduce the ANT algorithm redundancy. It scatters the agents inside the search field.* Assigning a default angle selected from the set $\theta_i \in \{N, NE, E\}$ to each agent leads to the initial spreading of agents. The agent follows the default angle ($angle_p$), to which it has been assigned for a number of predefined steps. If it encounters an obstacle before it reaches this limit, it tries to avoid it by using the BRW algorithm, where its time limit is $\omega = PRE\_DEFINED\_STEPS \times 36$. Once the obstacle has been avoided it continues to follow the default angle assigned to it until it reaches the pre–define step limit (NP). It then switches to the EANT

Table 7.2: Function:*opposite()*

$angle_t$	$opposite(angle_t)$
$angle_t \in \theta_{NW}$	θ_{SE}
$angle_t \in \theta_{SE}$	θ_{NW}
$angle_t \in \theta_N$	θ_S
$angle_t \in \theta_S$	θ_N
$angle_t \in \theta_W$	θ_E
$angle_t \in \theta_E$	θ_W
$angle_t \in \theta_{NE}$	θ_{SW}
$angle_t \in \theta_{SW}$	θ_{NE}

algorithm and randomly selects another angle, i.e. $angle_t \neq opposite(angle_p)$. Even if the agent is able to move in any direction, the default angle will still affect its trajectory because the EANT algorithm prevents the agent from moving in the opposite direction of its default angle as much as possible. We introduce the opposite angle as $\theta_n = opposite(angle_p)$, and if the agent has to select the opposite angle (e.g. stuck in a narrow dead end) it should weight it with the weight given in equation 7.3:

$$W_{i_t} = \frac{NP + N_{\theta_{i_t}}}{Ns} \tag{7.3}$$

Then we apply EWMA function 7.2 to weight the direction of EANT agent as follow (applied, in algorithm 9) :

$$\text{EWMA function} : \begin{cases} \mathcal{W}_{s_t} = \lambda \times \frac{N_{\theta_i}}{Ns} + (1-\lambda) W_{i_{t-1}}, & \text{if } \theta_n \neq opposite(angel_p); \\ \mathcal{W}_{o_t} = \lambda \times \frac{NP+N_{\theta_n}}{Ns} + (1-\lambda) W_{i_{t-1}}, & \text{if } \theta_n = opposite(angel_p); \end{cases} \tag{7.4}$$

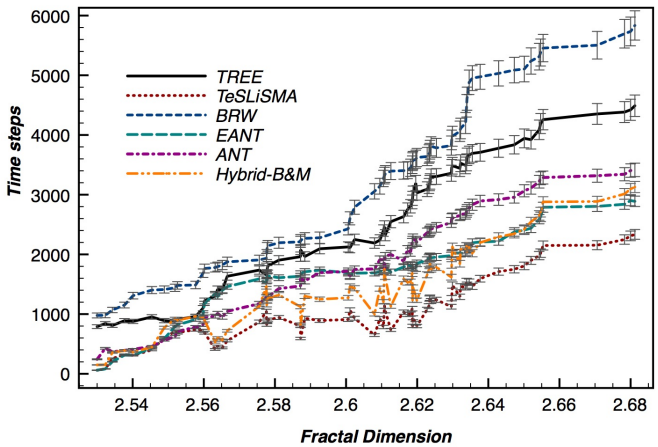

Figure 7.1: Discovery time of multi–agent exploration algorithms, (one time step = 36 sec.)

We set up another experiment. As we expected from the results in Chapter 6, figure 7.1 shows that the performance of the TeSLiSMA multi–agent algorithm is superior to that of the other multi–

agent algorithms (ANT, EANT, Hybrid-B&M and TREE). EANT performs very closely to Hybrid B&M team of agents. Nevertheless, for environments with complexity larger than 2.65 ($FDP > 2.65$) the performance of the TeSLiSMA team is still not desirable in the context of search and rescue operations with an elapsed time of more than 20 hours. A group of agents using the TeSLiSMA algorithm can achieve goals that are well beyond what can be expected if we just add up their individual abilities. However, we sometimes find that their emergent behaviour can turn out to be undesirable. The detection of unwanted emergent behaviour of the system in this activity was the sole responsibility of human tester who was monitoring it with very limited tool support. We believe a more intelligent way is strongly desired as part of the testing and monitoring

A potential solution in this regard is to use testing and monitoring techniques from soft computing. An interesting approach in this direction has been put forward by *Denzinger* and *Kidney* [DK06, KD06], who propose to use evolutionary learning methods to search for unwanted emergent behaviours in MAS has put an interesting approach in this direction forward. The general idea is to employ another group of cooperating agents (testers) that interact with our multi–agent system. Testers use learning methods to adapt their behaviour based upon observations made of the MAS. This process will continue until the testers recognise the unwanted behaviour. Therefore, by removing the undesired emergent behaviour we are able to reduce the exploration time and improve our search technique. In the future we have to concentrate more on learning how to detect and intelligently monitor these unwanted emergent behaviour, through a tester team, and use this information to improve the performance of the TeSLiSMA algorithm for the very complex search fields.

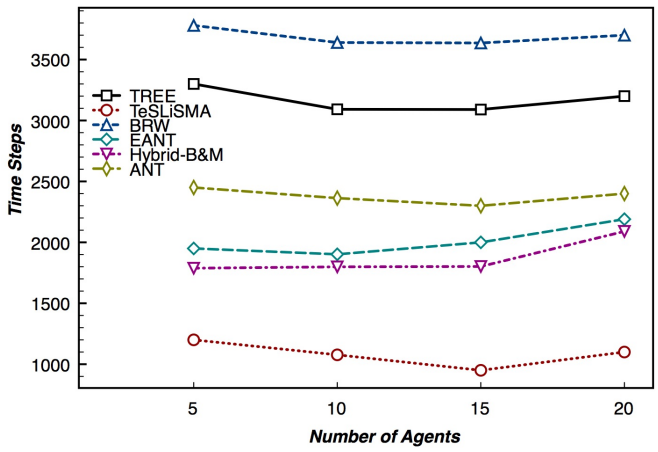

Figure 7.2: Discovery time of multi–agent exploration algorithms, (one time step = 36 sec.)

We select a search field, with the complexity of $FDP = 2.622$, and we vary the number of agents for the team of agents from 5 to 20. As figure 7.2 is indicating all algorithms except Hybrid-B&M (with no information sharing among the team) and EANT algorithms have a very little changes in their performances when we increase the number of agents. For other algorithms, their performances have improves when we change the number of agents from 5 to 10. However, for BRW, TREE, and ANT after increasing the number of agents, from 10 to 20, their average performances remains unchanged

(or a very little changes). For TeSLiSMA technique when the number of agents are increased till 15 we see improvement in team performances. However, when we increase the number of agents to 20, their discovery time increase (disimprovement in the performance). This is also increased for the hybrid-B&M for the 20 number of agents. This is because of the traffic congestion caused by the large number of agents inside a search field. That is why it is very important to estimate the maximum number of agents which are sent inside a search field according to the frame size (from the outside).

Algorithm 9 *performEANT*

Inputs: $angle_p; NP; N_{\theta_i}; W_{i_t}; angle_t$*; ogm:occupancy grid memory; A:agent*

Outputs: $N_{\theta_i}; W_{i_t}; angle_t$*; ogm:occupancy grid memory; NP*

$W_{min} = 1$*;* $W_0 = 0$*;*$\lambda = 0.7$;
while *true* **do**
  %if the agent is scatter inside the field, after moving NP steps
  if $NP >= PRE\_DEFINED\_STEPS$ **then**
    %scanning all 9 cells around our agent inside its window vision
    for $i = 1; i < 9; i++$ **do**
      %recognise the opposite direction, agent should avoid it
      $\theta_n = opposite(angel_p)$
      %if scanned cell is not the opposite direction and free then
      if $i \neq n$ AND $\theta_i =$ "$FREE''$ **then**
        $\mathcal{W}_o$[equ: 7.4]
        if $W_{min} > W_{i_t}$ **then**
          $W_{min} = W_{i_t}$
          $AN = i$
        end if
        %else if scanned cell is the opposite direction and free then
      else if $i == n$ AND $\theta_i =$ "$FREE''$ **then**
        $\mathcal{W}_s$[equ: 7.4]
        if $W_{min} > W_{i_t}$ **then**
          $W_{min} = W_{i_t}$
          $AN = i$
        end if
      end if
    end for
    for $i = 1; i < 9; i++$ **do**
      if $W_{min} == W_{i_t}$ **then**
        Fill the ogm
        $N_{\theta_i}++$
        $angle_t = \theta_{AN}$
        $NP++$
        return *NP,* N_{θ_i}*,* $angle_t$*, ogm*
      else
        return *ogm*
      end if
    end for
  else
    $\omega = PRE\_DEFINED\_STEPS \times 36$
    performBrownian Random Walk algorithm
  end if
end while

7.2 Multi–agent Performance Improvement

Furthermore this chapter is an introduction to our future work, applying Soft computing (i.e. fuzzy systems, neural networks, and genetic algorithms) to controlling robots [KR95]. There are some attempts in the literature to integrate soft computing methods in order to improve the overall performance. This means the advantages of some components compensate the disadvantages of the others.

The motivation to apply *fuzzy neural* network is to develop more efficient methodology for general goal–directed navigation in generic indoor environments. In this approach a navigator is proposed that should travel to a pre–defined task goal safely and efficiently without any prior map of the environment. There are core techniques, for directing autonomous agents, such as *neural competition* [HR97] and *fuzzy logic* [SM89]. These rule–based goal–directed control techniques are popular computational intelligences in the field of robot control.

From the viewpoint of goal–directed systems, neural networks are superior to fuzzy systems, since they are more flexible in their handling of input–output relation. However, a practical weakness is that it can be hard to train neural networks for complex control problems, particularly if evolutionary techniques have not been employed to optimise the process. Usually, autonomous exploration robots are not required to obtain complete and exact information for moving in their unknown environment. Therefore, fuzzy logic based controllers for autonomous robots present a more promising approach, but the challenging problem is to improve the fuzzy system via learning.

A recently proposed approach is to build a model with fuzzy operation through the learning of a neural network (Fuzzy Neural Network, FNN). For this model, if $X = x_i$ and $Y = y_j$ represent the input and output of the system, the weights W = w_{ij} describes the relationship between X and Y. Based on fuzzy inference in figure 7.3 the definition is $Y = X \circ W$, where ($\circ$) is a compositional operator and generally should be defined in the form of a transform function. In FNN a neural network architecture finds a solution to a fuzzy relation equation. Furthermore A is defined as the fuzzy set of the FNN system and estimated by training the system with various sensor samples inside the environment.

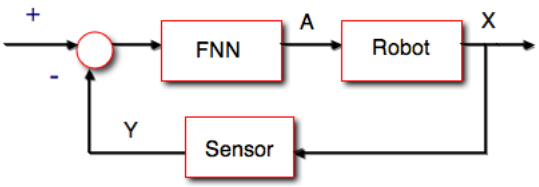

Figure 7.3: The control block diagram of the autonomous mobile robot

Genetic algorithms (GA) have also used to find near global values of the neural fuzzy logic controller (NFLC) [IFSA95]. There is a new approach to evolutionary algorithms called *LEGA* [SL01] that optimises the controller algorithm during the learning period. LEGA combines the standard genetic technique with numerical optimum seeking for a limited number on elite individuals in each generation. What makes this technique superior to other available techniques in robot control is that it can lead the robot to a global optimum in only a few generations.

7.3 Toward Real Robot Testing

Our presented simulations cannot *completely* reflect the complexity and noise available in the real world. Simulated sensor sets, even if degraded in order to avoid being unrealistically ideal, do not represent the type of input that a system will encounter from a sensor in the real world. In general, it is almost impossible to capture the level of detail of an environment and the physical interactions that occur due to the modelling efforts and the resulting strain on the processing power of the computer running the simulation. Similarly, the actuation of the robot cannot be completely modelled. Lags in response time, backlash in gears, and other physical symptoms, are difficult to characterise and accurately model. Therefore, we should be aware of the limits of the simulation in validating and verifying the robotic system performances. Although simulation techniques provide an important and effective first step, more testing will be required before a real system can be built.

Urban Search and Rescue Simulation (USARSim) [Bal06] was developed under a National Science Foundation grant to study Robot, Agent, and Person Teams in Urban Search and Rescue. It has since been adopted, modified and extended by researchers around the world. This simulation package builds on a set of modifications to the Unreal Tournament game engine that enables actors in the game to be controlled through a TCP/IP socket. USARSims interface is designed to mimic the low–level robotic interfaces that are commonly found in USAR and Explosive Ordnance Disposal (EOD) robotics system. This provides the robotic developers with the embodiment needed for the development and testing of autonomous systems. For debugging and development of USARsim a test control interface is provided to control the robot(s). Robot control programs can be written using the *GameBot* interface [GSaf], *MOAST* System [MSf], and *Player* interface [PSs].

Player is available on a wide range of platforms and is used throughout the world. It was an international group of robotics researchers that developed the platform and continue to improve it. Player runs on Linux, Solaris, BSD and Mac OS X. Some users have worked with Player using Cygwin, a Linux environment running under Windows. The official language interfaces include C, C++ and Python, while third parties support Java. Player defines the robot device interface. It operates at the network level allowing robot remote control over a sockets interface, and is designed to work with a variety of robot and sensor hardware. Player uses a client/server model so each side can be written in any programming language. The system supports multiple, concurrent client connections. Application programs are linked to Player client libraries that provide a connection to a Player server. The Player server normally resides on the target robot and is linked to Player device drivers. The device drivers map to logical, abstract drivers that a client normally sees. There are sufficient simulated devices and robots for use with Stage and Gazebo so a user can easily experiment with Player.

Writing device drivers for a new robot and its peripherals is relatively straightforward. As with many OS device drivers, a single driver can handle many numbers of physical devices. For example, a standard driver is available for the SICK LMS200 laser range finder that matches the abstract "laser" interface. An actual connection might be specified as 'localhost:6665:laser:0' for port 6665, logical laser interface device 0. In addition to the Player server and interface standards, the Player project includes a

number of modules (e.g. Playernav is a multi–robot localisation, and navigation GUI).

7.3.1 Player/Stage

Here we deploy the *Player* interface and test it on the $2D$ *Stage server* prior to Unreal Tournament ($3D$ simulation environment). We propose the use of Player/Stage, a middle-ware commonly used as a standard by the robotics community, and a backbone of a heterogeneous ubiquitous system. Stage supports $2D$ sensing with various sensor models including sonar, scanning laser range finder, and pan-tilt-zoom cameras with colour blob detection. It is able to support simulated robots that utilise the Player framework. It has visual tracking capabilities that can be handy when viewing the simulation. The implementation is relatively simplistic but very useful. The initial environment is setup via a text-based world file with details such as:

position

(

name “millibot1”

pose [1 1 90][1]

sonar()

range_return 1

)

The entity definition describes the properties of the entity. This includes the interfaces that will be supported. Control is in the client that you supply or you can use one of the standard Player user interfaces. Robots can also be free running as well. The Stage interface is relatively simple. You can pan, zoom and examine entities. It is possible to view almost all data related to the system.

The configuration file (below) is relatively simple but it shows the various interfaces for the PC–Bot [Roba].

driver

(

name “wbr914”

provides [“position2d:0” “ir:0” “aio:0” “dio:0”]

port “/dev/ttyUSB0”

)

It is just a matter of starting up the Player server and then Playerv to control the system. Player runs on the PC–Bot but Playerv uses port 6665 and can be on a networked PC. It is then a matter of selecting the interfaces that will be controlled just as when using Stage. The client side consists of a collection of C++ class definitions that essentially match all the standard interfaces provided by Player. This includes things like *LaserProxy*, *GripperProxy* and *SpeechProxy*. Movement is done using the *PositionProxy* that has methods such as *SetSpeed* and *SetMotorSpeed* as well as *GetPosX* and *GetYaw*.

The Player Project provides the basics needed for multiple robot control and simulation. It is easy to incorporate new devices and, therefore, new robots, because the system just deals with the basics. It

[1] [x, y, $angle$ (degree)]

is not as ambitious as Microsoft Robotics Studio although the two are comparable. Microsoft Robotics Studio has a much more sophisticated underpinning though and the might of Microsoft behind it. Still, there is much to be said for open source and simplicity. The Player Project may be just what you need for robotic control. In the next section we corroborate TeSLiSMA superior performance by deploying an experiment on Player/Stage.

7.3.2 Applying TeSLiSMA al. on Player/Stage

Owen [Owe09] explains the *Player/Stage tool*, process of setting up a new simulation environment and how to deploy the simulation environment and control for robotic systems. Player, is a Hardware Abstraction Layer, which means that it talks to the bits of hardware on the robot (like a claw or a camera) and lets the user control them with their own code. Stage is a plug–in to Player, which listens to what Player is telling it to do and turns these instructions into a robot simulation. It also simulates sensor data and sends this to Player which in turn makes the sensor data available to your code.

We have tested the TeSLiSMA exploration algorithm performance on Player/Stage to enable us to challenge other exploration techniques in Robocup–Rescue Simulation Competition (through USARsim). Player can also be run on real robots in addition to the Stage simulation system. , e.g. running Player on the White box Robotics PC–Bot. In future we are able to move to next stage, which is testing our multi–robot exploration technique on real robots. Figures 7.4 and 7.5 show a comparison between Player/Stage and our tool, which is based on the CLOWN formalism. The environments generated by our search field are used as the bit maps applied on Player/Stage. However, to make it more realistic we altered a little, for instance we remove some sharp edges and instead add some smooth obstacles. Below is an example of loading a bitmap environment inside World file:

map

(

bitmap “bitmaps/BIG28.png”

size [16 16][2]

name “BIG28”

)

The area of the search field frame used with our tool is $25m^2$(i.e. $(50 \times 50)0.01m^2$) and that used with the Player/Stage system is $25.6m^2$(i.e. $(16 \times 16)0.1m^2$).

In the Player/Stage setup we are able to define the LED colours for the robots and therefore the Followers have a “LED=green” and the Leader has a “LED=red”. Also all the agent’s positions should be defined in advance. We are therefore not able to send them inside one by one, as in our tool. There are three robots in both tests. Similar to the agents inside the CLOWN formalism, each robot in Player/Stage is equipped with a camera “range_max 0.38” and eight IR sensors “$sview$ [0.001, 0.24, 45]” (For each IR sensor we define [$range\_min(m)$, $range\_max(m)$, $view\_angle$(degree)]). The observation view of these sensors is clearly indicated Player/Stage figures 7.4 and 7.5, where the robot size is $0.11m \times 0.11m$.

In the Player/ Stage model we are furthermore able to add odometry errors as well as errors for the

[2] $[x, y]$

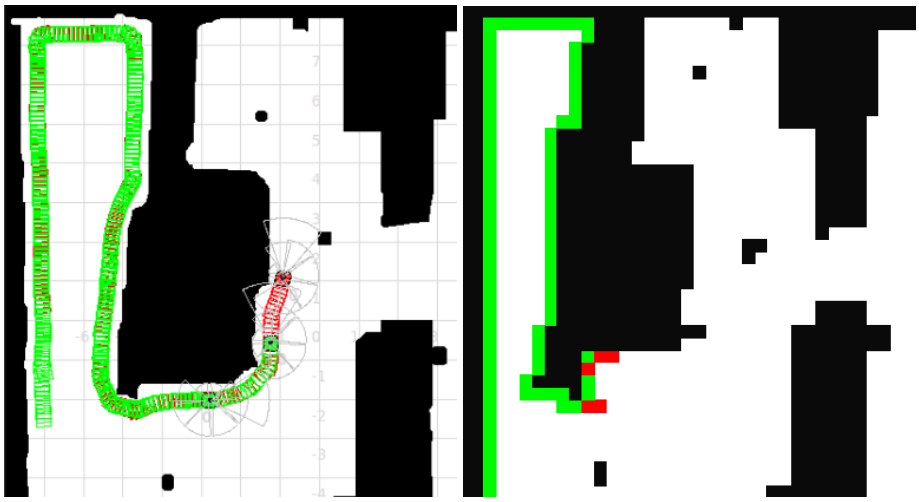

Figure 7.4: Wall–following multi–agent by Player/Stage (right) and CLOWN formalism (left)

sensor readings. In contrast to our tool where the cells are the same size as the agents and the agents move one cell at a time, the steps of the Player/Stage robot are not as big as its size. This makes the comparison between these simulation tools harder. In this regard we record the time it takes our team to locate an object, discovery time, inside both tools. We change the position of the object 10 times, randomly, to avoid any sensitivity to the position of the object.

Figure 7.5: TeSLiSMA algorithm by Player/Stage (right) and CLOWN formalism (left)

In figure 7.6 we compare the elapsed time from three different random environments (each run 100 times). We test the performance of three agents through the Player/Stage simulation with the same bit maps applied by our tool. The comparison is not exact. The robot movements are quite realistic in the Player/Stage system where the robot can get stuck unless it has sufficient space to turn and we consequently have to make small changes to some narrow pathways. However, the results indicate that the performance is quite similar. We believe that, our technique therefore seems very promising if applied on real robots in future.

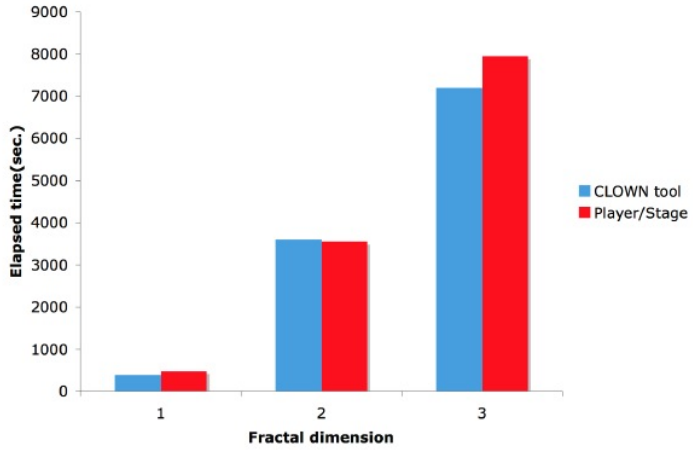

Figure 7.6: Performance Comparison

7.4 Conclusion & Future work

As we were expecting further traffic congestion might occur inside our search terrain, because of the narrow pathways and several number of agents. Therefore, we should set a maximum number of agents, in addition to the minimum number of agents, that can be sent inside a search terrain according to the search terrain external frame size. In all experiments TeSLiSMA algorithm proves its significant performance compared to other competing algorithms. Furthermore, to make this algorithm ready for real experiments we have tested its performance with Player interface

We summarise the USARSim tool motivations as:

1) it is an interesting tool for academic and industrial prototyping. New robotics algorithms and approaches can be safely and cost efficiently tested before carrying them over to real world systems.

2) it allows performance evaluations of systems as well as algorithms and procedures. Standardised scenarios with known parameter settings are especially suitable as the basis for a fair and reproducible comparison. Also, important nontrivial comparison bases are available in USARSim for quantitative performance analysis.

3) simulation environment like USARSim are very helpful for training purposes. End–users typically have too limited time scales and financial budgets to engage in training activities with real systems.

Independently from how sophisticated simulation frameworks have became, the human tester is not able to monitor and visualise all the weaknesses of exploration techniques. Furthermore setting up various experiments can only verify if we reach our goal or not, it does not clarify the problem. Therefore, it is important that we develop an additional feature for our simulation tool, known as a multi–agent tester group. The tester group is an intelligent tester inside the simulation environment that should recognise unwanted emergent behaviour in MAS.

Chapter 8

Conclusion

This thesis was inspired by the need to discover trapped victims rapidly inside pancake collapses of a building, mainly in earthquake affected areas. The aim has been to establish an advanced, autonomous exploration strategy that will enable rescue workers to perform fast but thorough searches of cluttered and confined regions during USAR operations. There are three factors expected for an advanced exploration strategy in this thesis: Efficiency (e.g. minimising victim discovery time), Thoroughness, Robustness and Fairness.

To evaluate the performance of such a complex system, it was necessary to create a simulation environment that would enable us to test their behaviour in a variety of controlled operations. The simulation allowed us to study of the system dynamics and limitations of the millibot team including aspects like team collaboration and interaction. The proposed architecture is based on two different types of robots: mother as a base station and its minis. Each is designed to fulfil a special task in the search and rescue scenario. The single mother allows high level and centralised control whenever this is deemed necessary. It also transports its minis to the narrow entrance of areas that are inaccessible to the mother agent due to size constraints. It is responsible for providing power to its millibots as well as computational tasks like the merging the local memory grids to a global representation of the search area. Meanwhile, the mini robots are intended to penetrate inside the narrow pathways of the maze–like environment to explore the operation area looking for trapped victims. They should be able to deal with their positional uncertainty and traffic congestion inside unexpected long narrow pathways.

8.1 Critical Evaluation

We fulfil our thesis objectives, introduced in Chapter 1, as follows:

1- Frontier–based exploration agent moves toward boundaries or frontiers between known and unknown environment. In this technique agent is seeking to visit new areas without considering any transition error in its collected data. This technique is efficient, in terms of time, and relatively easy to integrate on any kind of robot and independent from the search terrain. However, long term uncertainty will cause dramatic errors in the agents understanding of its position. This prevents the agent to reach to its frontiers (in most cases the agent have to switch to ANT behaviour to find its frontiers). We developed an advanced frontier–based exploration strategy that is able to reduce agent's positional uncertainty. We

evaluated that the redundancy of revisiting of unnecessary visited regions is reduced significantly, in this regard:

1-1- We introduced DAEA, a mathematical model for frontier-based algorithm. This model will enable our agent to: *1-1-1-* apply the effectiveness of coverage (our on-line metric) to asset it toward frontiers; *1-1-2-* recognise problematic uncertainties, through a predefined threshold; *1-1-3-* apply approximate–POMDP techniques to reduce positional uncertainties (because of transition errors).

1-2- We added approximate POMDP-solver to deal with agent's positional uncertainty, whenever it stops the agent from performing efficiently. For EDAEA approach we define a *dual mode controller*: *1-2-1-* a mathematical model of frontier-based that directs the agent toward frontiers (as discussed above); *1-2-2-* to reduce positional uncertainty we applied approximate POMDP technique.

2- Sending several agents inside a confined and cluttered search terrain, leads the system to initial congestion. We introduced TeSLiSMA a technique that enables its agents to squeeze inside a search terrain while moving in a chain formation and able to divide the area among themselves. Each agent released from the chain of agents performs EDAEA inside its subspace.

3- Further traffic congestion might occur inside our search terrain, because of narrow long pathways. Therefore we should estimate a maximum number of agents, in addition to the minimum number of agents, which are allowed to send inside a search terrain according to the search terrain external frame size. A formula is defined and discussed in this thesis to enable a search group prevent any further congestion while performing exploration.

In this activity we manage to develop an efficient, thorough, robust, and fair search strategy for a team of limited sensing simulated robots. We analyse the experimental results for this development considering both its merits and its shortcomings while comparing it against the most competing available algorithm, Hybrid-B&M algorithm.

8.1.1 Efficiency

TeSLiSMA efficiency overcomes the performances of other strategies inside complex maze–like environments. We have estimated the average discovery time of Hybrid-B&M algorithm (second best algorithm) and TeSLiSMA algorithm (the best algorithm) in three levels of complexities.

	Level 1	**Level 2**	**Level 3**
Search Field Complexity	$2.53 < FDP \leq 2.58$	$2.58 < FDP \leq 2.63$	$2.63 < FDP \leq 2.68$
P-Value	$<$0.2	$<$0.023	$<$0.042

For Level 2 and Level 3 the estimated p-value is less that 5% (statistical significance), which indicates the significant efficient performance of our novel search technique for search fields with complexity of $FDP > 2.58$.

8.1.2 Thoroughness

Agent performing TeSLiSMA is able to cover large environments completely in an acceptable time for the USAR operation. From the experimental results, the average time to cover a complete search area

with 144 various level of difficulties is less than 13 hours.

Summary of the results:

- When it is 10000 times larger than a mini robot (i.e. frame size of 100×100),

Strategy	$Average_{Time}$	No. of Agents	MAX_{Time}	MIN_{Time}
TeSLiSMA	12 and 380 seconds	24	23 hours	72 seconds

- When it is 2500 times larger than a mini robot (i.e. frame size of 50×50),

Strategy	$Average_{Time}$	No. of Agents	MAX_{Time}	MIN_{Time}
TeSLiSMA	7 and 174 second	8	12 hours and 779 seconds	83 seconds

Hybrid-B&M agents failed to cover the complete search area in more than $\%40$ of the tests, because of odometry errors. In this regard, Hybrid-B&M agents believed that they filled their occupancy grid memory completely, however they filled their memory with incorrect data that misled them to a wrong conclusion. This problem might be solved by adding a technique such as loop–closing in SLAM [EP03].

8.1.3 Fairness

To evaluate the fairness of our developed strategy we sent in various number of agents inside different search fields with a range of complexities.

- From the experimental results, 60% of all the search fields, with 245 various difficulties and three frame sizes, are covered in less than '6' hours.
- By changing the place of a single goal randomly inside the search field the average standard deviation of goal discovery time is estimated as 2 hours (*MAX* 7 hours and 131 seconds, *MIN* 72 seconds) .

8.1.4 Robustness

To evaluate the robustness of TeSLiSMA multi agent system we investigated: losing a team mate (e.g. replacing the leader); the immediate agent that detect the lost team mate informs the team by changing its LED colour; kidnapped robot (e.g. the average recovery time for our agents was 28 minutes), and for Leader stuck inside a dead end there is an emergent behaviour defined for the team members.

In all these circumstances our agents manage to perform the desirable emergent behaviour and capable of continue to their exploration without any failure.

8.2 Practical Limitation

TeSLiSMA exploration algorithm will face some limitations, when is applied in practice, as follows:

- TeSLiSMA has only tested inside static environments, since this technique is based on agent's recorded data. To deal with dynamic environments extra behaviours should be included. For instance the agent should be able to distinguish odometry errors from the environment changes (e.g. further collapses);

- TeSLiSMA agents cannot provide an accurate topological map of its search field. It only creates an approximate occupancy grid memory of its search field. However, through this limited information by the small millibots (platform of $100 \times 100\ mm^2$) the rescue team are able to identify victim's location approximately, and estimating the position of points of interest (e.g. a narrow pathway that might lead the agent to a life safe void) for further exploration ;

- We have tested the performance of the TeSLiSMA agent inside $2D$ simulation environment. For real scenarios (working in $3D$ exploration) TeSLiSMA agent requires extra emergent rules and behaviours, for instance it requires a behaviour to enable the agent to perform wall following if there are pathways not only on the ground level but also higher inside the wall;

- Form the experimental results TeSLiSMA agent performances inside the search fields with complexity larger than 2.65 , $FDP > 2.65$, are undesirable. This is in the context of search and rescue operations with an elapsed time of more than 20 hours for a search field as large as $100m^2$. Note that the discovery time is not decreasing by adding more agents, it might also increase because of the traffic congestion inside the cluttered and confine search area.

8.3 Future Work

To improve the performance of our self–organised exploration system we have identified solutions for the future work both in the hardware and software of the robots.

• *Adding extra hardware feature*– Swarm–bots or 'S–bots' [Wikb], has been created in a project headed by the Artificial Intelligence Laboratory at the Free University of Brussels, Belgium. In this project they are a group of mini robots that are able to connect to one another through their gripper arms. Therefore, each s–bot is not only equipped with essential sensors, and limited computational (as our millibot), it is also equipped with a gripper arm that can connect it to another s–bot. S–bots follow a few basic rules: If you see another robot with red light, connect to it. This connection can help the system to pass over gaps and steps where individual bot would have fail. No central control commands the whole swarm. The goal of s–bot project is that they can someday be used autonomously to explore at sea, in space, or infectious environment.

We are able to apply the gripper arm on our millibots in practice, for instance while they are performing wall following they can easily create a connected chain by connecting to each other. In this case we are able to speed up the operation even more, since there is no need for a robot to check the availability of other robot(s) in its line of sight. This enables us to reduce the number of LED colours applied to the multi–agents. Thus, it can reduce the chance of failure in colour readings for the MAS. Additionally the agents in the middle use less battery, therefore they can spare battery for filling their occupancy grid map.

• *Improving Software techniques*– As discussed in chapter 7, we might be able to improve our multi–agent system by applying soft computing techniques, such as LEGA or FNN. The idea is that to enable our system to learn, through limited sensing agents, new essential emergent behaviours, in a short time, for different situations to optimise the system. In this regard we should develop an additional

feature for our simulation tool, known as tester multi–agent group. The intelligent tester should learn the unwanted emergent behaviours while learning new essential behaviours for our MAS.

In the literature *Denzinger* and *Kidney* [DK06, KD06], used evolutionary learning methods to search for unwanted emergent behaviours. The general idea is to employ another group of co-operating agents (testers) that interact with our multi–agent system. Testers use learning methods to adjust their behaviour based upon observations made of the MAS. This process will continue until the testers recognise the unwanted behaviour. Therefore, by removing the undesired emergent behaviour we are able to reduce the exploration time and improve our search technique. In the future we have to concentrate more on learning how to detect and intelligently monitor these unwanted emergent behaviour, through a tester team, and use this information to improve the performance of the TeSLiSMA algorithm for the very complex search fields.

Additional features, as indicated through the thesis, can be added to the Simulation tool (e.g. $3D$ cubic cell, discussed in Chapter 3), as well as exploration algorithms (e.g. RF communications to upload other agents local memories, discussed in Chapter 6).

Appendix A

Sensor Selection

If robots are to detect victims trapped under the rubble of a collapsed building, they need to be outfitted with sensors. There is a great variety of sensors available and this appendix involves the selection, circuit design and implementation of sensors needed to accomplish a search and rescue mission. Each sensor has certain capabilities and limitations, thus to choose the correct sensor, one must first properly define the application. The important criteria to select sensors for this project are:

Low cost

Small size

Lightweight

Easy to interface

Robustness

Low power consumption

USAR explain that video cameras are essential for their robot missions because they permit the workers to navigate and see the site via teleportation. However for the purpose of victim identification, digital thermal cameras appear a much better choice. FLIR (Forward-Looking Infrared camera) for human heat detection has therefore been chosen for this project. In real scenarios Sonar might also be very useful for obstacle avoidance. Microphones, speakers, and sensors for motion detection might be added to this model later in the project.

A.1 Robot perception

Robots gain information about their environment by perception. Once information has been perceived, it can be processed and analysed to allow appropriate actions to be taken. What a robot can perceive is defined by its set of sensors and USAR missions deal with a robot type that is equipped with infrared range finders, an inertial measurement unit, wheel encoders, a camera and a localisation unit. Robot perception is critical when a robot has to act autonomously, considering that every action is based on the information perceived. Therefore, autonomous mobile robots are a great challenge, even more when operating in complex environments.

A.2 Sensors for Victim Detection

A.2.1 Vision

Vision sensors are the most common type of sensors used for human detection so far. There are different forms of vision sensors available in the industry: linear cameras, colour cameras, stereovision, and infrared cameras

Linear Camera: The linear camera is the cheapest vision sensor, but it is not very effectual to detect people, since more than one line of pixels is required for detection.

Colour Camera: Colour camera like low cost USB cameras (webcam) or even more expensive cameras with CCD (Change Coupled Device) sensors (good optics used in professional systems) might be very useful. The disadvantage of these types of cameras is that an operator must be employ to find victims, or an advance image processing software should applied (very time consuming). However, if robots are communicating with full colour LED, it is desirable to have 'VGA camera' [NGD+05]. For instance *DCSG* is a monocular VGA camera for machine vision tasks for scenes in motion. This is an ideal camera for our project since it has a size of: 38 mm high$\times$66 mm long $\times$25 mm deep, low weight, and low power consumption.

Stereo Vision: This is one of the most expensive device for this purpose, because two colour cameras must be deployed to take images of the scene. The difference between the two images provides the depth information. This method is known as disparity and enables the operator to evaluate the distance of the object from the camera.

Infrared Camera: The most efficient sensor for detecting humans is the infrared camera. This device gives a picture of the environment heat. Normally Pan and Tilt capabilities are added. In this case two servos are required for this new system operation; panning servo and tilt servo is a nice compact sub-miniature servo. Pan, tilt, and zoom cameras are some of the most versatile cameras in the market. They can pan (move left and right), tilt (move up and down), and zoom in or out.

A.2.2 Heat sensor

Heat sensors like Thermopiles and Pyroelectric detectors can easily sense a live human.

Pyroelectricity the ability of certain materials to generate an electrical potential when they are heated or cooled. Very small changes in temperature can produce an electric potential due to the materials Pyroelectric properties. Motion detection devices are often designed around Pyroelectric materials, as the heat of a human or animal from several feet away is sufficient to generate a difference in charge. Any material that has a natural charge separation (even in the absence of an electric field) is called a polar material. All polar crystals are Pyroelectric. The infrared Pyroelectric system has the potential optical power (that is received) to an electrical output signal. Though they provide adequately high output signals to permit an easy interface with a measurement or control circuits [HBSS95].

Thermopiles are a widely used type of remote , contact–less temperature sensing and can also be used as a way to convert thermal potential difference into electric potential difference. They are cheap, interchangeable, use standard connectors, and can measure a wide range of temperatures. Perkin–Elmer offers thermopile detectorz equipped with two or four channels and infrared spectral band pass filters.

These types of thermopiles are used for gas detection via IR absorption. Prominent gases to be detected are CO_2, hydrocarbons and CO [Ele]. Therefore human detection can be done either by temperature or even from their exhaled gasses.

A.2.3 Microphone

Inevitably survivors under the rubble will call for help. Therefore sound is a very accessible parameter to detect. A microphone can be attached on the robot to find victims who have been buried alive. A filtering system is highly recommended, to remove sounds outside human speech frequencies. They allow an operator or even more practical an intelligent program to alert the search team to the existence of a survivor under the rubble. Microphones are very low cost but additional data and signal processing is required.

A.2.4 CO_2 Sensor

Medical sensors like carbon dioxide sensors might help rescuers determine if a victim is still alive (by estimating human breathing cycle) and in need of rescue, therefore it improves the rescue operation. Most of these sensors are bulky (their monitoring equipment like screen and case) because they are usually used in hospitals. However, NDIR (non-dispersive infrared absorbance) sensors are able to monitor CO_2 emissions. It determines the amount of CO_2 absorbance; while CO_2 absorbance band is unique, and highly selective [Rsa]. CO_2 sensor chip is low cost and highly integrated with MEMS device [SCs].

A.2.5 SpO_2 Sensor

SpO_2 (Spot Oxygen saturation) [Mgs] sensor is a medical sensor that it senses blood oxygen content. It needs direct contact with the persons body (bare skin). Although this sensor has the extra ability to separate live humans from other living animals (because of the unique blood oxygen) it is a bulky system and thus is not suitable for this scenario.

A.3 Sensors for Navigation

A.3.1 Range Finders

Ultrasound is a sound with a frequency greater than the upper limit of human hearing ($16 - 20KHZ$). A common use of ultrasound is in range finding; this use is also called SONAR (Sound Navigation and Ranging). For robot movement, its microcontroller sends a signal to activate the sonar. Sonar emits a mostly inaudible sound and detects the return echo. By keeping track of the delay it can then calculate the distance of the detected objects. Sharp IR Ranger Finder is probably the best technology to deploy for this scenario. This method works by the process of triangulation. A pulse of light is emitted and then reflected back (or not reflected at all). When the light returns it comes back at an angle that depends on the distance of the reflecting object. Triangulation works by detecting this reflected beam angle. Therefore, by knowing the angle, the distance can be determined. These methods can only provide the distance between obstacles and the robot but cannot make the distinction between a human or non-human presence. Thus their most important application is 2D mapping. The ARC (Angle Range Compensation)

laser range finders designed by *Bushnell Scout*, not only give an accurate distance reading, but also displays the angle to the detected target. This compact and lightweight device can provide the search robots with both gyroscope and range finder.

Precise and accurate range-to-target information is readily provided by the laser range finders. However, currently available laser range finders are bulky, heavy, expensive and very difficult to mount. The US Army CECOM RDEC NVESD is addressing these laser range finder issues in the development of a *Micro-Laser Range Finder* (mLRF) [NBSL99]. Their motivation to build such a range finder is to mount it on the weapons for the soldiers. However these laser range finders are still very expensive to apply on search and rescue multi-robot systems.

A.3.2 Radar

Radar is appropriate for motion detection. Millimeter –wave radar is efficient for long distance motion measurement. This type of radar works on very high frequency $(5 - 24GHZ)$ and are able to operate through smoke, dust, fog or rain. These small and low power radars can detect motion up to 6 meters. However, they are very expensive to use on multi robots project [Bur04]. Laser Radar (LADAR) or LIDAR (Light Detection And Ranging) can be deployed under circumstances similar to millimeter wave radar, but uses laser beams to scans and process the signal echoed from targets. A Laser Radar (LADAR) seeker can detect objects and identify specific features with very high definition of up to 15cm resolution from a distance of 1,000 meters.

A.3.3 Electrical Compass

The vehicles heading contain the most significant navigation parameters (x, y, θ) in relation to the impact on accumulated deadreckoning errors. Therefore, sensors that are able to measure absolute headings accurately become extremely important in supporting the navigation needs of autonomous platforms. However, the magnetic field of the earth can be distorted near power lines or steel structures. This makes the straightforward use of geomagnetic sensors difficult for indoor applications (instead we deploy gyroscopes as equipped in IMU). For outdoor and large scale environments an electrical compass is a very suitable device to find directions. There are different sensor systems available to use:

- Mechanical magnetic compasses
- Fluxgate compasses
- Hall-effect compasses
- Magnetoresistive compasses
- Magnetoelastic compasses

However, the best suited compass for mobile robot application is the *fluxgate compass*. This electronic sensor measures the magnetic field of the earth directly. Therefore problems like spin, swirl, overshoot and sluggishness are eliminated. Fluxgate compass has these features: low power consumption, no moving parts, intolerance to shock and vibration, rapid start-up, and relatively low cost. If the

robot with fluxgate compass is expected to operate over uneven terrain, the sensor coil should be gimbal–mounted and mechanically dampened to prevent serious errors introduced by the vertical component of the geomagnetic field [OB00].

A.3.4 Gyroscope

For robots that have to crawl inside the rubble or even climb a ramp or steps indoor, a gyro sensor is crucial. This sensor is used to determine the angle of the main body of the analogue robot with respect to the horizon. It is a piezoelectric sensor, which detects speed of rotation and sends out an analogue voltage that is about $0.67mV/deg/sec$. In this case, by taking a base angle for robots main body, the angle of the main body with respect to horizon can be determined at every moment. By deploying two gyro sensors for each robot the stability of the robot can easily be determined [AAE$^+$05]. Currently, search robots usually use only rate gyros for correcting rotation and simplified schemes for gyro-compensation. Future configurations may integrate inertial measurement units (IMUs), tiltroll, and triaxial inertial sensors to measure and compensate for the many real-world conditions that confound the odometry, especially on hilly and rough outdoor terrain. The advanced algorithms provided by Kalman filtering, which extract an erratic signal from noise, may also be used to refine and fuse the various mobility-related sensor inputs into a better odometry.

A.3.5 3DM

3DM uses three orthogonal MEMS accelerometers and three orthogonal magnetoresistive magnetometers. It measures orientation over 360 degrees about any axis, but it is suitable for static and quasi-static environments only (roll over 360 degrees over one axis), because the accelerometers cannot accurately measure inclination in the face of inertial influences. Heavy low pass filtering can be used to alleviate this problem but it introduces a lag in the response to angular change [Sen]. However, there are more advanced devices in MicroStrain like 3DM–G that has a 9 total sensors and allows a 360 degrees range over all three axes (pitch, roll, and yaw). The combination of multiple MEMS sensors with proprietary software algorithms (running on tiny microprocessors) allows this company to enhance performance and simplify their sensors outputs while keeping things small. Bear in mind that 3DM and compasses are complementary and working together enables them to determine the direction of the robot.

A.3.6 GPS

By using the global positioning system (GPS, a process used to establish a position on the globe) the following two values can be determined anywhere on earth:

1) Ones exact location accurate to within a range of $20m$ to approx 1mm

2) The precise time accurate to within a range of $60ns$ to approx $5ns$

GPS receivers are used for positioning, locating, navigating, surveying and determining the time and are employed both in civil and military fields. A GPS system has been implemented for global positioning on the search robots for outdoor navigations. This system might prevent large faults in the localisation. Thus, if the position detected by other sensors differs considerably from the GPS position, robots location can be updated via GPS data. To convert the longitude and latitude data from GPS to

Cartesian coordinate, a microcontroller is required. This method has the most effect on localisation on outdoor and large-scale environments. A minimum of four satellites has to be detected by the receiver to give an accurate position estimate. Errors in an uncorrected GPS signal come in many forms and arise from a variety of different sources. These errors are high frequency noise and long-term drift. High frequency noise or spikes are easily identifiable on a 2D plot of the GPS track recorded from a moving platform. Although, no attempt has been made to formulate an explicit definition of what constitutes noise. For instance one of the methods to recognise these noises are epoch jumps in the GPS position. An epoch is one GPS cycle (milliseconds). Therefore if the new positions received form GPS maintained for more than approximately 30 seconds, then it is no longer considered noise but lies in the grey area between the two categories. The difficulty arises from the fact that in some instances the position can jump several meters and then either jump back on the next epoch or maintain that new position for a few seconds or indefinitely. Experiences have shown that the two main causes of GPS noise are satellites coming in and out of the view of the GPS receiver and multi–path effects. The magnitude of these errors varies from a few feet to hundreds of feet. Long term drifts are much more difficult to see on a track plot, since they change over a period of hours on the contrary to noise that occurs over a period of seconds. It is also difficult to determine the exact cause of these types of errors, but they are typically attributed to atmospheric effects in the ionosphere, troposphere and satellite geometry. The magnitude of these errors can vary from no error at all to thirty feet or more. The most common solution for solving GPS noise problem is to deploy other sensors and Kalman filter. An *inertial sensor* is an ideal companion for the GPS. Also by using Kalman filtering almost all the spikes in GPS seems to be smoothed out. Unfortunately, it does not do any good for the long-term drift errors. Here other techniques, like dead reckoning, may provide better solutions.

A.3.7 Inertial System

An Inertial System is a sensor package that measures the inertial forces generated by the movement of an object. The most frequent applications for an inertial system are navigation, attitude measurement and platform stabilisation. The system mostly measures accelerations and rotations about the three primary axes X, Y, and Z. Crossbow offers a wide range of inertial products based on MEMS and FOG (Fiber Optic Gyro) technologies [Tec]. All of its inertial systems are designed to operate without the need for external aiding. However, some of them do accept aiding signals such as GPS or Air Data to offer an even higher level of performance. When Inertial Navigation System is combined with GPS, it provides high quality measurements, including obscured-sky speed measurements, in environments where GPS alone struggles.

A.4 Noisy sensor measurements

Sensing devices do not operate perfectly, i.e., the measurements they produce do not always reflect the real state of the world, due to noise in the environment and to certain characteristics of the sensing technology. Therefore if we are modelling these sensors inside a simulated environment we should be aware of this and try to add a set of realistic errors as well.

Incorrect measurements from an active range sensor (e.g. IR) can be caused by reflective surfaces or ambient lighting conditions and can also depend on the distance to an object [Jon04]. The real world will not deliver perfect measurements as a simulation might do. Therefore, for a realistic scenario, it is important to integrate noise and error sources into the simulation.

There are different ways to simulate noise. The most simplistic way is to base noise simply on random numbers. But usually, when noise cannot be classified, a specific noise distribution is assumed. For a situation with no additional information, a Gaussian distribution (normal distribution) is the common choice. Only looking at the data sheet error specification of a specific robots perception devices, supports the application of a Gaussian error distribution. This only characterises the sensor error itself and the noisy environment still needs to be accounted for. Thrun et al. [SBD05], however, suggest a model for range finders, which takes the existence of four different types of measurement errors into account:

- small measurement noise
- errors cause by unexpected objects
- errors based on the failure to detect objects
- random noise

They create a probability distribution for each of the mentioned situations and mix those distributions, to finally reproduce the most likely measurement. This means that the measurement model is based on four different densities. So instead of using a pure Gaussian distribution, this offers a better representation of world noise and sensor errors. In order to produce a noisy measurement, the process takes an actual measurement (from the simulation) to create the appropriate mixed distribution and sample from it. At the end this sample becomes the final measurement.

Appendix B

Robot–Robot Interaction

There are several advantages associated with using a group of robots instead of a single one. An immediate advantage is that a robot group is more fault-tolerant, because one robot can repair or replace another in case of failure. Furthermore, a group of robots can overcome the limitations of a single robot and solve tasks that are too complex for a single robot to solve. The coordination among the robots is achieved, in this thesis, through self–organised behaviour within the system. This self–organisation is the result of local interactions among the robots, as well as between the robots and their environment. One such cooperative and collective robot group is known as a swarm. In the swarm each robot is controlled by simple strategies, and complex behaviour is achieved only at the colony level. Exploration swarm robotic applications have been inspired by social insects like ants. Ants collaborate by leaving a pheromone trail that attracts other ants toward a food source. This idea has been used in the development of robot search teams to allow individuals to interact and direct the team toward the localized victim. According to the reports from earthquake sites, quake survivors are frequently trapped in very close proximity to each other. Thus when a robot finds a survivor it is highly probable that there will be more survivors in close proximity to the first located victim, but it is only possible to take advantage of such occurrences if robots are able to inform each other of their findings.

Imagine a convoy of robots comprising a leader followed by several relay robots. Each relay robot is programmed to follow the one immediately in front of it. However to perform this movement they need to follow some sort of a trail. This trail can be a communication link or even a path guide. Having a reliable communications link between members of the robot team consequently plays a vital role In hazardous environments. Analogue radios have limited bandwidth and suffer greatly from multi–path fading. Wireless broadband digital communication, on the other hand, has proven to be much more robust and versatile. However, due to their high operating frequency, they need to maintain line of sight (LOS) contact with the base station and this is a requirement that is extremely difficult to achieve, especially in urban environments.

Pezeshkian et al. [PNB06] proposed a solution to this by using relay systems. Their proposed system includes several relay radios, known as Relay Bricks, that are carried by the mobile robots as payload. These Relay Bricks are plug–and–playable, and can be attached to many unmanned vehicles requiring long–range and non–LOS operational capability. Steele et al. [FS07] also devised a Stigmergic

search and rescue system that has been inspired by food foraging behaviour of ants. The robots in their system generate a stigmergic system by communicating directly or indirectly with each other. Among insects we find similar direct and indirect interactions as well. Direct interactions among the insects are the obvious interactions like, the exchange of food or liquid (trophallaxis), mandibular contact, visual contact, chemical contact, etc. [BDT99]. Two individuals interact indirectly when one of them modifies the environment and later the other insect responds to the new environment. In an ant colony, each ant will leave a chemical pheromone trail toward the food source, which is traceable to the other ants. This indirect interaction explains stigmergy, a method of communication in which each individual affects the emergent behaviour of its group. The same idea has been employed for stigmergic search system. Therefore each robot informs other robots of its achievement and vice versa. This positive feedback allows the system to improve its performance. Osherovich et al. [OBY06] also presented an ant inspired algorithm for covering continues domains by primitive robots that are only able to mark visited areas with pheromone and sense pheromone in their neighbourhood. They show that despite the simplicity of the robots in their team, they are able to cover their search environment efficiently.

B.1 Tether

The tether can act as a conduit for any subset of the following: power, data communication, gases or fluids supply [KJ97]. The tether can also be used as a safety line in cases such as a power failure, mechanical malfunctioning, or when a robot is deployed through small and narrow vertical openings [R.M04]. Tethers can be categorised as active and passive. Passive tethered robots have a tendency to drag and catch on obstacles. Many tethered robots are stopped before they complete their task by having their tether caught on an obstacle. Tethers also limit the depth to which the robot can be deployed because a tether has a finite length. Furthermore, in search and rescue robot team applications, robots have been known to cross paths and tangle in each other tethers. However, active tethers [YVLP09] a new research area for tethered robot systems. It provides a continuously distributed form of actuation that results from the water hammer effect. It reduces the friction of the tether by the jerks that are created by the water hammer effects.

Figure B.1: Active tether for a robot , University of Denver

B.2 Guide path following

In this section different methods of indirect interaction among robots are introduced. They provide the ability to mark visited places and produce a path guide to interact with each other. Apart from this, robots have no means of communication. They have no global information such as a domain map, global positioning, and coverage percentage.

B.2.1 U/V Stimulated Emission

The U/V guide path stripe is composed of fluorescent particles suspended in water–based hardener that permeates the floor surface for extended durability, and dries in approximately eight hours. An active ultraviolet light source irradiates the guide path, causing the embedded fluorescent particles to reradiate with a fairly narrow spectral output in the blue–green visible spectrum. The lateral position sensor that detects this stimulated response is insensitive to ultraviolet wavelengths, and thus sees the fluorescent stripe as a path with a fairly high signal-to-noise ratio relative to the surrounding floor surface. The Bell and Howell Mailmobile robot uses three discrete photo-detectors to track the glowing guide path [BH]. However, one of the disadvantages of this method is the potential interference from direct sunlight, which makes this path guide suitable only for indoor search applications. The guide path technology also requires a fairly smooth floor to ensure reasonable performance and is therefore not a practical solution in search scenarios involving significant amounts of rubble.

B.2.2 Heat and Odour paths

Early work in this area was based on a short–lived heat trail laid down by a lead robot equipped with a quartz–halogen projector bulb configured to raise the floor temperature beneath the unit during the course of transit [KR93]. Other robots would simply follow the temporary trail of the leader in convoy fashion. A standard Pyroelectric sensor as is commonly used for passive motion detection in security applications was modified to sense the residual thermal energy imparted to the floor. However due to interference problems associated with hot water and heating pipes embedded in the floor, as well as localized hot spots created by shafts of sunlight, this method was not really practical. In addition the 70 watt halogen heater placed considerable energy demands on the limited storage capacity of the onboard battery, which encouraged the scientists to improve this method.

Deveza et al [DRTMS94] devised a solution using a wide trail of camphor for the slave (follower robot) to detect and follow. The camphor is dissolved in alcohol and applied with a felt–tip applicator. The alcohol evaporates in seconds, leaving a trail of camphor particles on the floor. The choice of camphor was based on its ease of detection and the fact that it is inoffensive to humans, slowly subliming over a period of several hours into a harmless vapour. Camphor sensors should be equipped on the follower track team. These sensors are based on the gravimetric microbalance technique, employing a quartz crystal coated with silicone. Camphor molecules are absorbed into the crystal coating, adding to the effective mass of the crystal and thus lowering the resonant frequency in direct proportion to the odour concentration. So far discussions of both of these methods as path guides among robots are centralized because they have leader that provides them with the path. To make sure their system

provides a comprehensive search of the search field we require an operator to direct the leader and the system therefore becomes semi–autonomous. In a known environment (for instance one that has been mapped by other exploration robots before) the leader could guide its team to their localized goal for further assistance.

B.2.3 Environment Electrical Tagging

In this indirect interaction method, robots are able to communicate only by reading and updating the state of their local cell. In this method of communication the search environment should be instrumented with uniformly distributed miniature devices (e.g. motes or RFIDs). These devices have a memory and they are therefore able to store information regarding the environment. Robots visit the unvisited cells and tag them. This causes the cells to change states. For instance, when they are visited they will change their state from 0 to 1. This kind of environment, known as Intelligent Space (iSpace), has ubiquitous distributed sensory intelligence [LH02].

The iSpace propagates mobile robots and they will in turn change the state of the space. The intelligent iSpace has the capability of performing situation evaluation inside the space. There are two main process on the evaluated situation in iSpace. One is the sense–think–act loop, where the evaluated behaviors are applied directly by the mobile agent control system to produce intelligent response to the instantaneous situation. The other is the sense–learn–conclude loop, where new behaviors are learned and given to the control system. The sense-think-act loop of the iSpace is updated with behaviors what is derived from the sense-learn-conclude loop [SH04]. For instance Ferranti et al. [FTL07] present a system where all the robots within the system are able to read the tags (equipped by tag readers) and therefore able to understand the state of each cell and act accordingly. However, this method cannot provide a practical solution for robot–robot interaction in our search and rescue scenario. The method requires the environment to be definitely charted and for each cell an electrical device should be installed. This makes the technology suitable only for certain types of indoor environments, making it a specific rather than general solution.

B.3 Communication link

The approach to multiple robots denotes the advantage of the distribution of the risk, and the simplicity of the design. However when we choose to use multiple robots instead of a single robot a new problem might occur, communication. When multiple robots must cooperate and coordinate in order to complete a task, communication plays an important role. Some tasks may not even be able to be completed without communication. For example if robot R1 knows which task needs to be completed and can not complete the task by itself this robot needs to find a way to communicate this information to the other robots (e.g. R2 or R3) in order for the task to be accomplished. Here in this section direct method of interaction is discussed. On contrary to the indirect method, the search field remains untouched all the way through the search operation.

B.3.1 RF communication

Examples of conventional communication architectures are broadcast communication and point-to-point communication. Additionally, in complex systems, a hierarchical architecture is required to allow the team of robots to communicate effectively. The most important characteristics of suitable communication architectures are: adaptability, response, extensibility, and fault tolerance [TFU94]. These characteristics have a great effect on the abilities of the multiple robotic systems. Robots can easily carry miniature wireless devices that in turn can be used to create wireless, mobile, ad-hoc networks (MANET) using hop-by-hop routing along shortest paths to reach the controller from many sensor sources.

Wireless networks can be classified into two categories: Infrastructure–based wireless and Ad–Hoc wireless networks. In Infrastructure–based wireless networks there are usually some fixed nodes (base station) that are connected to wired backbone. These nodes provide communication between the wireless nodes and moreover provide gateways to other networks. Typical examples of this kind of communication are GSM networks, WiMax, and building wireless networks. While in wireless ad-hoc networks, nodes communicate directly with each other by means of a short–range wireless medium (e.g. WiFi, Bluetooth, Zigbee, etc.) [KL09]. Between the infrastructure–based mobile networks and ad–hoc networks, there are the hybrid networks where fixed infrastructure is used where it is available and ad–hoc communication where it is not. In Mobile Ad–hoc networks, nodes are free to move however they will disconnect and reconnect elsewhere. The disconnection is in the most cases unpredictable and lead to intermittently mobile connected ad–hoc networks. Therefore here, a new category of wireless networks has introduced which as Opportunistic Networks (ON). Because of the unreliability and short range of wireless communications in indoor environments, it is preferable for the system to use an additional interaction method. While inside a collapsed building the RF link (e.g. 2.45 GHz Zigbee) range often drops to a few meters or even blocked. Pan and Lowe [PL07] introduced RF communication via power cable transmission line. They used power line network of a building as an auxiliary signal transmission medium, which guide that weak signals from inside to get to the rescue team outside.

B.3.2 Visual communication

Lack of interoperability in radio communications is an urgent problem that affects every level of search operation. To avoid any disruption among the agents as the result of communication faulty two types of communication strategies can be implemented as introduced by other authors [VPnLO92]. Therefore one of the communication methods is an auxiliary method that might be used in cases of any faulty in communication system during the operation. For instance in some cases the second auxiliary method is that each robot emits a signal to indicate its position to other robots [GJ92, DFF92]. Visual communication among robots in an autonomous system is consistently via camera. Therefore robots should be marked and distinguished from other objects within the search field.

Dorigo et al [DN06] equipped their robots (s-bot) with RGB-LED ring and VGA camera. The ring emits a colour, each colour indicates robots mode. Other robots can distinguish these colours via their developed program, thus they can interact with other robots with different colours. For instance when a robot reaches an obstacle it changes its colour from green (wandering mode) to blue, which informs

other robots of its observation. This kind of visualisation is with active–beacon emitters. Visualisation method can also be done via passive retroreflective targets. Therefore robots can be coded with unique binary identifications. These robots are fitted with a retroreflective bar code. Bar codes are readable with LADAR, which each robot is equipped with this laser. Based on this bar code each robot obtains range, bearing, and orientation information of the robot in its laser range [PNB06].

Appendix C

Localisation

Inside a simulated environment, no localisation is necessary, because the exact position of the robot is known. However all real localisation techniques have errors, and therefore the simulation should have a realistic error added to the actual position. In this appendix we describe different passive localisation techniques:

C.1 Dead Reckoning

The dead reckoning (DR) method identifies robot positions by calculating the amount of travel from its starting point. It does this by integrating rotations of the right and left wheels, such as on wheel driven vehicles. This method is simple and is easily implemented. It can also identify robot positions in an uncharted environment because it only employs internal sensors.

agents are storing their pose (x_t, y_t, θ_t) at each time step. The simplest method for Dead Reckoning [Bor94, Rei91] in real application is to adjust a pair of encoders (drive wheels) on each nose wheel. The location of the agent is constantly updated using the following formula:

$$x_t = dis * sin(\theta) + x_{t-1} \quad \textit{and} \quad y_t = dis * cos(\theta) + y_{t-1} \tag{C.1}$$

The distance that robot has travelled since the last position calculation (*dis*), are calculated as (the current heading of the robot (θ) is estimated by compass) :

$$dis = (left_{encoder} + right_{encoder})/2.0; \tag{C.2}$$

WHEEL–BASE is the distance between the two differential drive wheels.

However this technique might have some serious errors while wheel slippage causes measurement errors, which is accumulated as the vehicle movements. Robots often use dead reckoning to estimate their position without a map, but its errors accumulate over time, and the dead reckoning position estimate becomes increasingly inaccurate. Therefore this popular method is not a reliable positioning method for long distances or uneven surfaces because of variations in wheel diameter and slippages.

To reduce the dead reckoning errors as much as possible, different additional methods have been devised. The search and rescue scenario causes the worst kind of localisation problems such as the kidnapping problem. A robot moving in such areas can experience unexpected and sudden position

changes. Odometry will be useless in such situations and only an inertial measurement system can help as an internal sensor. The internal position error correction scheme has been known as one of the clever ways to reduce errors [Rei91].

A smart encoder trailer is able to detect and quantify the instantaneous orientation errors. From Borenstein [Bor94] experimental results, rotary encoders (for error cancellation) improve DR performance in a perfect manner. However odometric drifts become more sever problem in certain situations when the robots environment contains a variety of ground surface types. Duckett et al. [Duc00] introduces self–localisation through dead reckoning while determining robots orientation using compass. This had the effect of removing the accumulated rotational drift affecting the robot's raw odometry. Feng–Ji et al. [FJHJA04] devise an efficient method for estimating long distance navigation of a mobile robot in an unknown terrain, by fusing dead reckoning, IR, and Sonar sensors.

Unfortunately, pure dead-reckoning methods are still prone to errors that grow without bound over time, so some additional method is necessary to periodically correct the robot position. It is common to combine the additional localization technique, such as triangulation from landmarks or map matching, with dead-reckoning using an extended Kalman filter to update the robot position probabilistically. In addition to deal with noisy odometric data and incomplete information, Partially Observable Markov Decision Process (POMDP) [LCK95] are often proposed to address the issue of goal pursuit in such environments.

C.2 Land marking

Landmark based navigation is intended to greatly improve robot position estimation over dead reckoning by tracking visual features in the environment and using them as landmarks. Therefore it uses the known or currently acquired knowledge about the robot 's environment (observed data) to improve the position estimation. Many other approaches reviewed in [BEF96] are limited in that they require modification of the environment. For instance some require artificial landmarks such as bar–code reflectors [eDWGS94], reflecting beacons, or visual patterns that are easy to recognise, such as black rectangles with white dots [Bor87]. Oussalah et al [OMB99] opened up a landmark–based method in which landmarks are composed of a set of infrared LEDs that help the mobile robots to accurately increase the accuracy of their estimation.

Some of the more advanced approaches use available landmarks- do not require modifications of the environment. Kortenkamp and Weymouth [KW94] and Mataric [Mat90] use gateways, doors, walls, and other vertical objects to determine the robot's position. The *Helpmate* robot uses ceiling lights to position itself [KW90]. Additionally Dark/Bright regions and vertical edges are used in [CC85, WFW95], and hallway and openings, and doors are used by the approaches described in [SK97, SK95]. The landmark method, which estimates current position relative to landmarks, cannot be used in an uncharted environment. Furthermore in landmark localisation predefined features are required, thus they are not as robust as necessitate in our research scenario. In literature landmark localisation is one of he most popular techniques for navigation inside explored or known environments [HGS00, SD98].

C.3 Cooperative positioning

In this context, most work on localisation has focused on the question *how to reduce the odometry error using a cooperative team of robots*. Kurazume et al. [KH96] introduced Cooperative Positioning System (CPS) by dividing robots in to two groups, A and B. Group A stands still and act as a landmark while group B moves. Later on, group B become landmark for moving group A. This dance is repeated until they reached a target position. In this positioning system they have proposed triangle multi–robot localisation [Eve95] method in which three or more robots form a triangle and each robot moves in turn while being located by the others. This positioning is a mixture of dead reckoning and landmark positioning techniques. It uses multiple robots, which each are equipped with sensors to measure their positions relative to one another[1] . CPS is able to reduce the odometry error in an uncharted environment or even in indoor environment and underground where GPS cannot be used. However, their technique is not able to recover from significant sensor errors.

Other researchers work in this area, CPS, and try to improve this technique [RDM97, Bor95]. None of these techniques incorporates environmental feedback into the estimation. Even if the initial position of robots are known , they ultimately will get lost. Navarro-Serment et al. [NSCPK99] have developed a novel method that combines aspects of GPS, land-mark based localization, and dead reckoning. The method uses synchronised ultrasound pulses to measure the distances between all the robots on a team and then determines the relative positions of the robots through *trilateration*- i.e. determination of the position based on distance measurements to known landmarks or beacons. Their leap–frogging algorithm allows the team of robots to move over large distances while maintaining accurate position estimates with respect to the initial reference frame. Simulation results indicate that this localisation system has the potential to be an order of magnitude more accurate than traditional dead reckoning systems. This technique is not a very suitable technique for avoiding obstacles and entering narrow pathways, furthermore it has no assurance for a fairness search. The problem arises when agents have to share their recorded data at each step, to create their coverage map. Thus it is not a very suitable technique for this project by itself, while in search and rescue operations we need a comprehensive search method and implicit communication among agents, inside voids, is a problematic issue.

C.4 Greedy Localisation

These kinds of localizations usually are designed for the agent centred search to restrict planning strategies. Agent centred search methods decide on the local search space (adjacent cells around the robot) and determine which moves to execute within it. Here robots with short–range sensors operate in a grid world (a chess board) with no knowledge about their terrain. Available equipments (sensors on board) provide the robots with information about their neighbouring cells. Occupied cells and defined borders (border cells) are all un-traversable. Robots are allowed to move one cell in any direction unless it is not traversable. In some case this method of localization determines how to move the robot until it knows its current position- according to its available knowledge about environment (i.e. map). In the

[1] for more details about relative localisation refer [Roe08]

literature similar self–localisation has been discussed [DRW95, Sch97]. However it might be known as a very specific landmark technique. Researchers have extended this technique by applying probabilistic localisation methods.

C.5 Probabilistic Localisation

To accommodate the noise and ambiguity arising in real–world domains probabilistic localisation methods have been developed to obtain robust estimate of the location of the robots. These methods generally incorporate some observation model that gives the probability of the sensor measurement given the location of the robot and the parameterisation of the environment map. Sometimes this parameter vector describes explicit properties of the environment, such as position of landmarks [FWS98] or occupancy values [KC99]. It might describes an implicit relation between sensor pattern and location, such as neural networks [Oor97], or look–up tables [FWS98].

There are several high level localization methods based on probabilistic modeling of pose uncertainty, motion, and perception. It is commonly assumed that the environment is Markovian (Defined in [SBF+98] as "past and future data are conditionally independent if the current state is known"). Due to the soundness in theory and ease of implementation, the Markov assumption is popular for research and practical applications. At any point in time, Markov localization maintains a probability density (belief) over the entire configuration space of the robot; however, it does not provide an answer to: how to control the robot's actuators?

A well–known implementation is Monte Carlo localisation which is based on particle filters [DFBT99]. One advantage of this approach is that, if global localisation of the robot is possible, then particles are allowed to be uniformly spread over the entire free space. Some variants were designed to track multiple targets [SBFC01] and multiple robots [FBKT00]. Kalman filters are also widely used due to their ability to optimally track the path in the presence of Gaussian noise [Rou00]. It was mentioned in [GF98] that according to experiments, general grid–based Markov localisation is more robust than Kalman filtering while the latter can be more accurate than the former. A fundamental weakness of Kalman filters is that the initial state must be known within a given error range.

Smith and Cheesman [SC86] proposed a stochastic map to model geometrical location errors. Their technique has been widely used in mobile robotics for processing geometrical information coming for a range of different sensors, odometry, laser range finder, sonar, and vision among others [MD06]. Additionally the standard approach to simultaneous localisation and mapping (SLAM) is initially based on their technique.

Appendix D

Robot Selection

Search–and–Rescue using robotic systems is a very hot research topic in both academia and industry. In this appendix different research teams, industry productions, and robot competitions and workshops are discussed.

The Science and Technology (S &T) directorate of the Department of Homeland Security (DHS) has initiated an effort with the National Institute of Standards and Technology (NIST) to develop comprehensive standards related to the development, testing, and certification of effective technologies for Urban Search and Rescue (USAR) robots. These USAR robotic performance standards cover sensing, mobility, navigation, planning, integration, and operation control in order to ensure that the robots can meet operational requirements under the extremely challenging conditions that rescuers are faced with, including long endurance missions. There will be annual workshops to monitor progress as well as several events that allow responders to work with emerging robotic equipment in realistic environments while helping to refine proposed test methods.

The USAR robotic standards effort also focuses on fostering collaboration between first responders, robot vendors, other government agencies, and technology developers to advance consensus standards for task specific robot capabilities and interoperability of components.

The National Institute of Advanced Industrial Science and Technology (the new AIST) is a newly formed research organization that is the result of an amalgamation of the 15 research institutes previously under the former Agency of Industrial Science and Technology (the former AIST) in the Ministry of International Trade and Industry and the Weights and Measures Training Institute. It becomes Japans largest public research organization with many research facilities. AIST (as same as NIST) engages in world level research and development using local technological resources. It also helps the local industrial technology by strengthening the cooperation among local industries, academia, and governments.

D.1 Standards for USAR robots

This catalogue allows the comparison of robot performances in competitions and at the same time, gives a brief insight into the motivations and real challenges of robotic search and rescue. The tasks of teams of robots involved in search and rescue defined by the National Institute of Standards and Technology as part of the Department of Homeland Security and relevant for this project are: negotiating compromised

and collapsed structures autonomously finding victims (and reporting their condition) produce a human readable (practical) map of victim locations identify hazards. The aspects mentioned here are mainly part of a pure search task, while the catalogue definitions are broader and cover the topic of physical rescue as well. The Department also stresses that currently performance metrics exist only for the above mentioned aspects, but future versions of the catalogue will incorporate metrics for rescue. The following list is an extract of the performance metrics list [oHS08] considered to be relevant for this project:

1) In the category of mobility and locomotion: Sustained speed, endurance and tumble recovery (considering none, self–righting and invertible continuous operations) are the main metrics. They are measured during the traversal of different terrain types, e.g. plane but obstructed areas or an inclined terrain.

2) For the design quality of human–system interaction: The time for initial training to control the robots and a proficiency education are measured in hours and years. Furthermore the usability (e.g. remotely controlling the robot) is a metric based on an effectiveness evaluation, but assistive elements such as automatic notifications or path tracing are simply tested on their existence.

3) Sensing metrics: Performance evaluation in this category concentrate on the fact, that the robot provides spatial modeling and allows or provides a waypoint annotation.

For performance evaluation, some of these aspects are applied to different types of environments or specific situations, e.g. measuring sustained speed while moving over a step or alternatively over rough, inclined terrain. The given evaluation gives the basic motivation and criteria to enhance, develop and measure robotic control and also suggest a basic operator interface. This project will not be able to extensively measure its approach against all the given criteria. Instead it will provide the simulation environment to do so.

D.2 University Research

Urban Search and Rescue robots work with rescuers (Policeman, Fire-fighters and medical assistance) in a disaster environment of man made structure, like collapsed buildings. There are several teams working on USAR robotics. In this regard Carnegie Mellon University is being founded by the National Science Foundation to investigate the use of semi–autonomous robots for urban search and rescue. These robots will assist firemen, police, and disaster agencies with exploration, site evaluation, and human detection. The goal of this research is to develop an interface and coordination module to support these tasks. There are three most advanced research teams, *CRASAR* University of South Florida, Utility Vehicle for Search *UVS* Kobe University of Japan, and Kohga University of Tokyo [Bur04].

The most recent academic projects in Europe that involves both hardware and software development for robotic research are:

-I–Swarm :The I–SWARM [IS] project was awarded in December 2003 under the European Union Information Society Technologies (IST) 6th framework programme. It is a follow–up of the MiCRoN project. The I–SWARM project involves ten leading academic organisations throughout Europe. There are many potential benefits of such a project including greater flexibility and adaptability of the system to the environment, robustness to failures, etc. Moreover, their collective behaviour opens up new

application fields that cannot be solved with todays tools. With a suitable sophisticated positioning system, possibly based on that used by insects and incorporating tactile sensors and a small but effective vision system, the individual agents will be able to communicate between themselves and thus enable and promote the desired swarm effect.

-FP6 :The European FP6 Advanced Robotics projects [Pro] has developed a swarm of autonomous robots that is able to adequately assist and safeguard fire fighters in the event of fire. These robots detect plume sources and reconstruct the map of the destroyed building. The aim of these robots is to conduct fire fighter safety assessments. Their Guardian is a Hi–tech miniature robot. It works in a large team of thirty robots. The swarm of guardians gleans information while searching for fires, human dangers and obstacles. Collected information will be report back to the fire fighters.

-SUAAVE :SUAAVE, stands for Sensing, Unmanned, Autonomous Aerial VEhicles, [SUA]. It is an EPSRC project funded for 3.5 years from 2008. The SUAAVE consortium focuses to investigate the principle underlying the control of clouds of networked resource–limited unmanned aerial vehicles(UAVs) acting as sensor platform. Its main focus is on search and rescue operation. The novelty of these mobile sensor systems is that their movement is controlled by fully autonomous tasking algorithms with two important objectives: i) to increase sensing coverage, ii) to maintain network connectivity to enable real–time communication.

-ISLab Intelligent System Lab The driving theme of the Intelligent Systems Laboratory is Research and Development of Decentralized Systems, including Multi-Robot, Multi-Agent and Management systems [Lab].

D.3 Industry Efforts

There are many companies that develop mobile robots. Some of their products might be useful for this project (e.g. mini Whegs). However there are only few companies that commercially designed robots for USAR. NASA and the United States Department of Defense also have ongoing efforts with various universities to develop intelligent robots with a variety of sensors. The most advanced projects in industry are listed below:

-iRobot :The US government finances iRobot company and they develop some robots that can be used in small and dangerous areas. One of the robots is called Packbot robot. This robot has different sensors like, cameras, microphone, laser range finders, sonar and IR sensors. They are also working on a project called Deployer. Deployer has a team of little robots that can be placed where it wants [imoGr].

-Mobilerobots :Mobilrobots is a designer and manufacturer of mobile robots since 1995. They have great variety of robots especially for localisation and mapping. They mainly produce autonomous robots for different autonomous operations [dmopr].

-Inuktum :There are originally designed for pipe and tank inspection. The Nanomag is designed to adhere magnetically to metal surfaces: horizontally, vertically, and even upside down. The system is extremely effective in areas that are too small or toxic for human inspection. Their robots have some applications that might be useful to deploy them in urban search and rescue field.

-Nasa :In the NASAs Jet Propulsion laboratory, some research on USAR field has been done. They have tested different sensors like stereo camera, IR, GPS... All these sensors were outfitted on different type of robots and their performance in different situations have been tested [Bur04].

-ISD :The Intelligent system Divisions research and development focuses on measurements, standards infrastructure for the intelligent systems and intelligent control. However its main aspect is distributed control among multiple autonomous robots rather than limited individual sensing or control model [ISD].

-K-team :K-team cooperation is a Swiss company that develops, manufactures and markets high quality mobile robots for use in advanced education and research. The Khepera Linecard is now a standard for academic research, while the Hemisson robot is designed for teachers and hobbyists. This company has produced a number of robots that are very suitable for autonomous robotic missions as well. This includes their latest product Khepera III, which is a very advanced robots in this regards. E-puck is also one of their simple robots that has been used for search and rescue operations many times before. Khepera has a simulator, which is a freeware package that allows developers to create controllers for the mobile robot using the C or C++ languages. It includes an environment editor and a graphical user interface. With that we are also able to switch very easily between the simulated robot and the real one [coo].

-SuperDroid robots :At SuperDroid robot they provide a wide range of robot kits [Robb]. Their Speciality is building custom robots and robot related projects, such as ATR(All terrain robots). The robots can house a variety of wheels, motors and frames. The different wheels will allow All-Terrain travel. The Omni wheels can be used for vectoring in any direction without turning. The motors can be regular size or super–sized for climbing over rocks.

D.4 IEEE International Workshops and Conferences

The International Workshop on Safety, Security, and Rescue Robotics (SSRR) is dedicated to identifying and solving the key issues necessary to field capable robots across a variety of challenging applications. The most interesting topics that are concerned with the Self-organized Multi–robot Systems project are the perception for navigation, hazard detection, victim identification, Human–robot interfaces for improved remote situational awareness. The Performance Metrics for Intelligent Systems (PerMIS) workshop is in a series targeted at defining measures and methodologies of evaluating performance of intelligent systems. In this workshop applications of performance measures and practical problems in commercial industrial, homeland security, and military are examined. These two workshops usually share plenary presentations, robot demonstrations, technology exhibits, and social events.

The IEEE Robotics and Automation Society [RSb] is interested in both applied and theoretical issues in robotics and automation (e.g. IROS, ICRA. and ROBIO). In this society Robots are defined as intelligent machines and systems used, for example, in space exploration, human services, or manufacturing; whereas automation includes the use of automated methods in various applications, for example, factories, offices, homes, laboratory automation, or transportation systems to improve performance and productivity.

D.5 Competitions and Events

Disasters that can serve as field settings for evaluating robot (and human-robot) performance are rare and unpredictable. Therefore, every year urban search and rescue competitions should be speed up the development of research, to learn from one another, and to forge connections to the search and rescue community. USAR Robocop competition promotes collaboration among the researchers,in both software and hardware.

The *DARPA Grand Challenge* [Cha] is a prize competition for driverless cars, sponsored by the Defense Advanced Research Projects Agency (DARPA), the most prominent research organization of the United States Department of Defense. Congress has authorized DARPA to award cash prizes to further DARPAs mission to sponsor revolutionary, high-payoff research that bridges the gap between fundamental discoveries and their use for national security. DARPA has technologies needed to create the first fully autonomous ground vehicles capable of completing a substantial off-road course within a limited time.

The *European Land-Robot Trial* (ELROB) [TERT] is a European event which demonstrates the abilities of modern robots. The ELROB is no competition, like for example the US DARPA Grand Challenge, but a pure demonstration of what European robotics is able to achieve today. The scenarios are designed to simulate real world missions, be it military or civilian ones. There are no artificial constraints set to these scenarios to ease the task for the robots like e.g. very visible road markings. These forces the participating teams and systems to fulfill the high requirements set by the real world scenarios.

Appendix E

TeSLiSMA Algortihm

In this appendix we will present the TeSLiSMA algorithm. First, we describe the required agent behaviour and then the algorithms implementing this behaviour.

E.1 Behaviours

Here we define three aspects of the behaviour:

-moveForwardNextToWall: If *followedwall=NULL*, the 'moveForwardNextToWall' behaviour causes the robot to move forward from its current configuration until one of its sensors detect a wall, then robot turns to align itself so that its 'wall–follow sensor' (one left side sensors that is define as wall–following sensor as well) faces the wall. That wall is named according to agent direction, for instance if the wall is in north direction followedwall= 'north'.

-moveDirection: The 'moveDirection' (e.g. *moveEast*) behaviour causes the robot to move forward, maintaining a range $r_0 \in [r_{min}, r_{max}]$ to the wall with its wall–follow sensor (e.g. followedwall= 'north'). Where $[r_{min}, r_{max}]$ is define as the *safe wall–following range* for each agent.

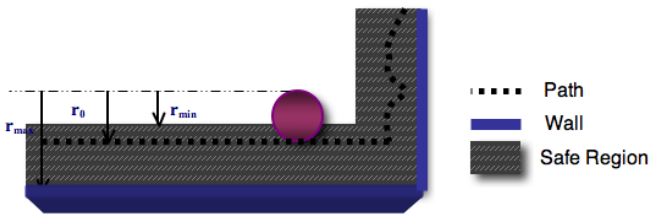

Figure E.1: The 'safe' Wall–following range

The wallfollowing terminates when the agent detects a welldefined corner. In this case a wall defined exterior occurs when the wall falls away. This happens when the robots distance from the wall is larger than $r_{max} - r_0$. Likewise, a welldefined interior corner occurs when the wall just inwards more than $r_0 - r_{min}$. In both cases the agent should fix its distance and then continue with moving forward next to the wall. The safe range will be adjusted for the agents according to their sensor range, agents size, and their environment.

We applied moveDirection behaviour only when applying on Player/Stage[1]. Additionally we defined 'turnSpeed'. This speed should be ajusted through several experiments inside different fields. It is the speed at which the robot can safely change direction to avoid collisions.

E.2 Algorithms

Algorithm 10 *performTeSLiSMA*

Input: *agents[ ]: an array of robots, ogm:occupancy grid memory; A:agent*

Output: *ogm:occupancy grid memory*

```
1: MR=0;                                              % MR=0, flag set for the Leader
2: for MR==0 do
3:     if A.state == "Head" then
4:         A.led.colour = "red"
5:         A.behaviour = performHeadBehaviour
6:         if A.masterReached( ) then
7:             MR=1;                                  % MR=1, flag set for the Filler
8:             A.behaviour = performEDAEA
9:             Break;
10:        end if
11:    else if A.state == "Follower" then
12:        A.led.colour = "green"
13:        A.behaviour = performFollowerBehaviour
14:    else if A.state == "Tail" then
15:        A.led.colour ="yellow"
16:        A.behaviour = performTailBehaviour
17:    else if A.state == "Filler" then
18:        A.led.colour ="off"
19:        A.behaviour = performEDAEA
20:    else if A.state == "Idle" then
21:        A.led.colour ="purple"
22:        A.behaviour = performIdleBehaviour
23:    else if A.state == "Link" then
24:        A.led.colour ="green"
25:        A.behaviour = performLinkBehaviour
26:    else if A.state == "Pebble" then
27:        A.led.colour ="white"
28:        A.behaviour = performPebbleBehaviour
29:    end if
30:    result = A.behaviour()
31:    if result == NO_CHAIN then
32:        result = noChain(A)
33:        if result == EDAEA_NEEDED then
34:            return result
35:        end if
36:    else if result == BROKEN_CHAIN then
37:        result = brokenChain(A)
38:    end if
39:    return ogm
40: end for
```

2

//

[1]It was not required for CLOWN formalism

[2]Algorithm 18 is a procedure

Algorithm 11 *performHeadBehaviour*

Input: *AH*: the head of the robot chain.

Output: *errorCode*

```
1: robots = AH.getRobotsInLineOfSight()          %agent checks the connection to its chain
2: for each A in robots do
3:     if robots.count == 0 then
4:         return NO_CHAIN
5:     else if A.led.colour == "red" then
6:         return MULIT_HEAD
7:     end if
8:     AH.moveForwardNextToWall(0)
9:     if AH.nextToMaster() then
10:        AH.masterReached()
11:    end if
12: end for
13: return SUCCESS
```

Algorithm 12 *performFollowerBehaviour*

Input: *A*: a follower robot.

Output: *errorCode*

```
1: robots = A.getRobotsInLineOfSight()
2: for each A in robots do
3:     if robots.count < 2 then
4:         return BROKEN_CHAIN
5:     else if A.led.colour == "purple" then
6:         A.state == "Head"
7:     end if
8:     moveForwardNextToWall(A)
9: end for
10: return SUCCESS
```

Algorithm 13 *performTailBehaviour*

Input: *A*: the tail robot.

Output: *errorCode*

```
1: robots = A.getRobotsInLineOfSight()
2: for each A in robots do
3:     if robots.count == 0 then
4:         return NO_CHAIN
5:     else if A.led.colour == "purple" then
6:         A.state == "Head"
7:     end if
8:     moveForwardNextToWall(A)
9: end for
10: return SUCCESS
```

Algorithm 14 *performIdleBehaviour*

Input: *A*: the tail robot.

Output: *errorCode*

```
1: robots = A.getRobotsInLineOfSight()
2: if robots.count == 0 then
3:     return NO_CHAIN
4: end if
5: moveForwardNextToWall(A)
6: return SUCCESS
```

Algorithm 15 *noChain*

Input: *A*: a robot in the chain.

Output: *errorCode*

```
1: A.led.colour = "blue"
2: A.moveBackward()
3: result = A.behaviour()
4: if result == NO_CHAIN then
5:     A.led.state = "off"
6:     return EDAEA_NEEDED
7: end if
8: return SUCCESS
```

Algorithm 16 *brokenChain*

Input: *A*: a robot in the chain.

Output: *errorCode*

```
1: A.led.colour ="blue"
2: A.moveBackward()
3: result = A.behaviour()
4: if result == BROKEN_CHAIN then
5:     if A.hasRobotInFront() then
6:         A.state ="Tail"
7:     else
8:         A.state = "Head"
9:     end if
10: end if
11: return SUCCESS
```

Algorithm 17 *stuck*

Input: *A*: a robot in the chain.

Output: *errorCode*

```
1: A.state = "Idle"
2: moveForwardNextToWall(A)
3: return SUCCESS
```

Algorithm 18 *moveForwardNextToWall*

Input: *A*: a robot in the chain, *counter* default = 0.

Output: *p: agent_next_position*

```
1: if A.followedWall = null then
2:     if A.scanWestWall() then
3:         A.followedWall = "west"
4:     else if A.scanSouthWall() then
5:         A.followedWall = "south"
6:     else
7:         throw AssertionError
8:     end if
9: end if
10: if counter > 3 then
11:     throw StuckException
12: end if
13: if A.followedWall == "west" then
14:     if A.scanNorthWall()  then
15:         A.followedWall = "north"
16:         A.moveForwardNextToWall( counter+1)
17:     else if A.scanEastNorthWall() then
18:         A.moveNorth()
19:     else
20:         A.moveNorthWest()
21:         A.followedWall = "south"
22:     end if
23: else if A.followedWall == "south" then
24:     if A.scanWestWall() then
25:         A.followedWall = "west"
26:         A.moveForwardNextToWall(counter+1)
27:     else if A.scanSouthWestWall() then
28:         A.moveWest()
29:     else
30:         A.moveSouthWest()
31:         A.followedWall = "east"
32:     end if
33: else if A.followedWall == "east" then
34:     if A.scanSouthWall() then
35:         A.followedWall = "south"
36:         A.moveForwardNextToWall(counter+1)
37:     else if A.scanSouthEastWall() then
38:         A.moveSouth()
39:     else
40:         A.moveSouthEast()
41:         A.followedWall = "north"
42:     end if
43: else if A.followedWall == "north" then
44:     if A.scanEastWall() then
45:         A.followedWall = "east"
46:         A.moveForwardNextToWall(counter+1)
47:     else if A.scanNorthEastWall() then
48:         A.moveEast()
49:     else
50:         A.moveNorthEast(counter+1)
51:         A.followedWall = "west"
52:     end if
53: end if
54: p=A.next_position()
55: return p
```

Functions: *A*.state*[the state of the agent]*, *A*.led.colour*[the LED colour of the agent]*, *A*.behaviour*[agent changes its behaviour to]*, *A*.masterReached*[Head agent reaches its master]*, *A*.moveForwardNextToWall*[agent in a chain should move next to a wall]*, *A*.followedWall*[which wall the agent is following]*, *A*.scanSouthWestWall*[agent should check for other available walls, while performing wall following, for instance when following East wall it should check if cells in its south, south east and east are occupied by an obstacle(a cell occupied by obstacle equals to wall)]*.

Bibliography

[AAE+05] H. Aria, F. Arjomandi, H. Eftekhary, F. Houshmand, A. Raessi, K. Rezaee, and H.Zibadel. Phoenix rescue team. *Robocup Rescue-Robot League, Osaka*, 2005.

[Adm96] United States Fire Administration. Technical rescue program development manual. Technical report, USA, 1996.

[AFH+98] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi robot cooperation in the martha project. In *proceeding of IEEE Robotics and Automation Magazine*, volume 5, 1998.

[AK01] Amin Atrash and Sven Koenig. Probabilistic planning for behavior-based robots. In *proceeding of the International FLAIRS conference (FLAIRS*, pages 531–535, 2001.

[AoT] Computer Science A.Torralba and Artificial Intelligence Laboratory Massachusetts Institute of Technology. `http://web.mit.edu/torralba/www/`.

[ARM] HEADQUARTERS DEPARTMENT OF THE US ARMY. Field manual no. 7-7j, department of the army, 1986.

[Bal98] T. Balch. *Behavioural Diversity in Learning Robot Teams*. PhD thesis, College of Computing, Georgia Institute of Technology, USA, 1998.

[Bal00] R.J. Balling. Pareto sets in decision-based design. *Engineering Valuation and Cost Analysis*, 3:189–198, 2000.

[Bal06] S. Balakirsky. Usarsim: Providing a framework for multi-robot performance evaluation. In *proceeding of PerMIS*, pages 98–102, 2006.

[BDT99] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, 1999.

[BEF96] J. Bornestein, H. R. Everett, and L. Feng. *Navigation mobile robots: System and Techniques*. Wellesly MA: A K Peters, Ltd., 1996.

[BH] Bell and Howell. `http://www.mailmobile.com/`.

[BH04] K. R. Beevers and W. H. Huang. Loop closing in topological maps. In *proceeding of the IEEE International Conference on Robotics and Automation (ICRA)*, New Orlean, April 2004.

[BM58] G. E. P. Box and M. E. Muller. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, 1958.

[BMS02] W. Burgard, M. Moors, and F.E. Schneider. Collaborative exploration of unknown environments with teams of mobile robots. *Advances in Plan-Based Control of Robotic Agents*, 4226, 2002.

[BMSS05] W. Burgard, M. Moors, C. Stachniss, and F. Schneider. Coordinated multi-robot exploration. In *proceeding of IIEEE Transactions on Robotics*, volume 21, pages 376–386, 2005.

[BNRDW00] T. Bailey, E.M. Nebot, J.K. Rosenblatt, and H.F. Durrant-Whyte. Data association for mobile robot navigation: a graph theoretic approach. In *proceeding of IEEE International Conference on Robotics and Automation (ICRA)*, pages 2512–2517, San Francisco, California, 2000.

[Bod07] M. Bodur. Control for hobby robotic system. Technical report, Computer eng. Eastern Mediterranean University, 2007.

[Bor87] J. Borenstein. *The nursing robot system*. PhD thesis, University of Haifa, Technion, Israel, 1987.

[Bor94] J. Borenstein. Internal correction of dead-reckoning errors with the smart encoder trailer. In *proceeding of the IEEE International Conference on Intelligent Robots and Systems, Munchen*, pages 2322–2327, 1994.

[Bor95] Johann Borenstein. Control and kinematic design of multi-degree-of-freedom mobile robots with compliant linkage. *proceeding of IEEE Transactions on Robotics and Automation*, 11:21–35, 1995.

[BP02] T. Balch and L.E. Parker. *Robot teams: From diversity to polymorphism*. Canada:AK Peters Ltd., 2002.

[BS95] A.L. Barabasi and H. E. Stanley. *Fractal Concepts in Surface Growth*. University of Cambridge, 1995.

[BS07] L. Bayindir and E. Sahin. A review of studies in swarm robotics. *Auton. Robots*, 15(2):115–147, 2007.

[BSS03] E. Bahceci, O. Soysal, and E. Sahin. A review: Pattern formation and adaptation in multi-robot systems. Technical report, Technical Report CMU-RI-TR-03-43, Carnegie University, USA, 2003.

[Bur88] P.A. Burrough. *Fractals and geochemistry: In The Fractal approach to the chemistry of distorted systems*. ed. By D. Avnir, Wiley and Sons, New York, 1988.

[Bur04] S. Burion. Human detection for robotic urban search and rescue. Technical report, Carnegie Mellon University USA, 2004.

[Cas] A. R. Cassandrai. `http://www.pomdp.org/pomdp/code/pomdp-solve-options-5.0.shtml`.

[CC85] T.S. Collet and B. A. Cartwright. Landmark learning in bees. *Journal of Comparitive Physiology*, pages 521–543, 1985.

[CFKM95] Y. U. Cao, A.S. Fukunaga, A.B. Khahng, and F. Meng. Cooperative mobile tobotics: Antecedents and directions. In *proceeding of IEEE/RSJ International conference on Intelligent Robots and systems(IROS), Pittsburgh*, volume 1, pages 226–234, 1995.

[CG07] T. Clarke and B. Goldiez. Collaboration on the edge of chaos. pages 122–128, 2007.

[Cha] DARPA Urban Challenge. `http://www.darpa.mil/grandchallenge/`.

[CKK96] A. Cassandra, L. Kaelbling, and J. Kurien. Acting under uncertainty: Discrete bayesian models for mobile robot navigation. In *proceeding International Conference on Intelligent Robots and Systems*, volume 2, pages 963– 972, 1996.

[CL04] Ch. K. Cheng and G. Leng. Cooperative search algorithm for distributed autonomous robots. In *proceeding of IEEE/RSJ International conference on Intelligent Robots and systems(IROS), Sendal, Japan*, volume 1, pages 394–399, 2004.

[Cle08] R.J. Cleary. Fundamentals of probability and statistics for engineers. *The American Statistician*, 62(1):90–91, February 2008.

[CM97a] D. Carmel and Sh. Markovitch. Exploration strategies for model-based learning in multi-agent systems. *Autonomous Agents and Multi-agent Systems, Kluwer academic Publishers*, 2:141–172, 1997.

[CM97b] D. Carmel and Sh. Markovitch. Model-based learning of interaction strategies in multi-agent systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 10:309–332, 1997.

[CM98] D. Carmel and Sh. Markovitch. How to explore your opponent's strategy (almost) optimally. In *proceeding of the Third International Conference on Multiagent Systems*, pages 64–71. IEEE Computer Society, 1998.

[CN01] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (slam): Toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17:125–137, 2001.

[coo] K-Team cooperation. http://www.k-team.com.

[Cou09] I. Couzin. The ants go marching – and manage to avoid traffic jams. *Journal of ScienceIllustrated*, pages 32–39, 2009.

[Cra09] J. Craighead. Using fractal dimension to assess robot operator search skill. In *proceeding of IEEE international workshop on Safety, Security, and Rescue Robotics(SSRR09)*, 2009.

[CT00] J.A. Castellanos and J.D. Tardos. Mobile robot localization and map building. a multisensor fusion approach. *Kluwer Academic Publishers, Boston, MA*, 2000.

[DB98] M. Dicke and P.A. Burrough. Using fractal dimension for characterizing tortuosity of animals. *Physiological Ecology, Physiol Entomol*, 13:393–398, 1998.

[DDDD98] H. Dickinson, H. Dickinson, M. M. Dodson, and M. M. Dodson. Extremal manifolds and hausdorff dimension, 1998.

[Dem67] A.P. Dempster. Upper and lower probabilities induced by multi valued mapping. *Annals of Mathematical Statistics*, pages 325–330, 1967.

[Dem68] A.P. Dempster. A generalisation of bayesian inference. *Journal of the Royal Statistical Society*, pages 205–247, 1968.

[DFBT99] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots, 1999.

[DFF92] A. Drogoul, J. Ferber, and A. Drogoul J. Ferber. From tom thumb to the dockers: Some experiments with foraging robots. In *proceeding of the Second International Conference on Simulation of Adaptive Behavior*, pages 451–459. MIT Press, 1992.

[Dil04] R. Dillmann. Benchmarks for robotic research, 2004.

[DK06] J. Denzinger and J. Kidney. Evaluating different genetic operators in the testing for unwanted emergent behavior using evolutionary learning of behavior. In *proceeding of the EEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT06)*, pages 23–29, 2006.

[dmopr] Mobilerobots In. designer and manufacturer of pioneer robots. `http://www.mobilerobots.com`.

[DN06] M. Dorigo and Sh. Nouyan. Chain based path formation in swarms of robots. In *proceeding of the 5th international workshop, Ant colony optimization and swarm intelligence (ANTS06)*, 2006.

[DRTMS94] R. Deveza, R.A. Russell, D. Thiel, and A. Mackay-Sim. Odour sensing for robot guidance. *International Journal of Robotics research*, 1994.

[DRW95] Gregory Dudek, Kathleen Romanik, and Sue Whitesides. Localizing a robot with minimum travel, 1995.

[Duc00] Tom Duckett. Learning globally consistent maps by relaxation. In *proceeding of the IEEE International Conference on Robotics and Automation*, pages 3841–3846, 2000.

[eDWGS94] H. R. everett, R. T. Laird D. W. Gage, G. A. Gilbreth, and R. P. Smurlo. Real–world issues in warehouse navigation. In *proceeding of SPIE conference on mobile robots, Bellingham*, volume 2352, pages 249–259, 1994.

[Ele] Opto Electronics. `http://optoelectronics.perkinelmer.com/contenet/reletelinks/ThermopileDetector`.

[EP94] T. Edlinger and E. V. PuttKamer. Exploration of an indoor-environmet by an autonomous mobile robot. In *proceeding in international conference on Intelligent Robots and Systems*, 1994.

[EP03] A. Eliazar and R. Parr. Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *proceeding 18th Int. Joint Conf. on Artificial Intelligence (IJCAI-03*, pages 1135–1142. Morgan Kaufmann, 2003.

[Eve95] H.R. Everett. *Sensors for Mobile Robots: Theory and Application*. Wellesly MA: A K Peters, Ltd., 1995.

[FBDT99] D. Fox, W. Burgard, F. Dallaert, and S. Thrun. Montecarlo localization: efficient position estimation for mobile robots. In *proceeding of national conference of artificial*, pages 343–349, Orlando, Florida, 1999.

[FBKT00] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization, 2000.

[FDS+02] R. Fierro, A. Das, J. Spletzer, J. Esposito, V. Kumar, J.P Ostrowski, G. Pappas, C.J. Taylor, Y. Hur, R. Alur, I. Lee, G. Grudic, and J. Southhall. A framework and architecture for multi-robot coordination. *International Journal of Robot Research*, 21(10):977–995, 2002.

[Fed] The Robocup Federation. `http://www.robocup.org/`.

[FJHJA04] Z. Feng-Ji, G. Hai-Jiao, and K.I. Abe. *Localization using combining sensors and dead-reckoning, Focus on computational neurobiology*. New york:Nova Science Publisher, 2004.

[FLS02] J.T. Feddema, C. Lewis, and D.A. Schoenwald. Decentralized control of cooperative robotic vehicles:theory and application. *IEEE Trans. on Robotics and Automation*, 18(5):852–864, 2002.

[FO05] L. Freda and G. Oriolo. Frontier-based probabilistic strategies for sensor-based exploration. In *proceeding of the IEEE/RSJ International Conference on the Intelligent Robots and Systems, IROS05*, pages 3892–3898, 2005.

[FS07] G. Thomas F. Steele. Directed stigmergy-based control for multi-robot systems. In *proceeding of the ACM/IEEE International Conference on Human-robot Interaction (HRI 07)*, pages 223–230, New York, NY, USA, 2007. ACM.

[FT05] A. Foka and P. Trahanias. Real-time hierarchical pomdps for autonomous robot navigation. In *proceeding IJCAI Workshop Reasoning with Uncertainty in Robotics*, 2005.

[FT09] E. Ferranti and N. Trigoni. The impact of localization errors on the performance of the ants exploration algorithm. In *proceeding of Workshop on Agent Technology for Sensor Networks (ATSN)*, 2009.

[FTL07] E. Ferranti, N. Trigoni, and M. Levene. Brick & mortar: an on-line multi-agent exploration algorithm. In *proceeding of IEEE International Conference on Robotics and Automation (ICRA)*, pages 761–767. IEEE, 2007.

[FTL08] E. Ferranti, N. Trigoni, and M. Levene. Hybridexploration: a distributed approach to terrain exploration using mobile and fixed sensor nodes. In *IEEE/RSJ 2008 International Conference on Intelligent RObots and Systems (IROS)*, 2008.

[FTL09] Ettore Ferranti, Niki Trigoni, and Mark Levene. Rapid exploration of unknown areas through dynamic deployment of mobile and stationary sensor nodes. *Journal of Autonomous Agents and Multi-Agent Systems*, 2009.

[FWS98] James Crowley Frank, Frank Wallner, and Bernt Schiele. Position estimation using principal components of range data. In *proceeding of the IEEE International Conference on Robotics and Automation*, pages 3121–3128, 1998.

[GBC04] C. Gonzalez-Buesa and J. Campos. Solving the mobile robot localization problem using string matching algorithms. In *proceeding of the International Conference on Intelligent Robots and Systems(IROS)*, volume 3, pages 2475– 2480, 2004.

[GBL02] H.H. Gonzalez-Banos and J.C. Latombe. Navigation strategies for exploring indoor environments. *International Journal of Robotics Research*, pages 829–848, 2002.

[GF98] J. Gutmann and D. Fox. An experimental comparison of localization methods continued. In *proceeding of IEEE International Conference Intel. Rob. Syst. (IROS)*, volume 1, pages 454–459, 1998.

[GJ92] S. Goss and J.L.Deneubourg. *Harvesting by a group of robots*. In Proceeding of the 1st European Conf. an Artificial Life, MIT Press, Cambridge, 1992.

[GMAM06] S. Garrido, L. Moreno, M. Abderrahim, and F. Martin. Path planning for mobile robot navigation using voronoi diagram and fast marching. In *proceeding of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2376–2381, 2006.

[GR01] Y. Gabriely and E. Rimon. Spanning–tree based coverage of continuous areas by a mobile robot. In *Annals of Mathematics and Artificial Intelligence*, volume 31, pages 77–98, Hingham, MA, USA, 2001. Kluwer Academic Publishers.

[Gro03] J Grob. *Linear Regression:Lecture notes in Statistics*. Springer, 2003.

[GSaf] A simple to use irc connectivity class Gamebot Sourceforge and a framework. `http://sourceforge.net/projects/zagamebot/`.

[Hau00a] M. Hauskrecht. Value-function approximations for partially observable markov decision processes. *International Journal of Artificial Intelligence Research*, 13, 2000.

[Hau00b] M. Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[HB07] D. E. Hogan and J. L. Burstein. *Disaster Medicine*. Lippincott Williams and Wilkins, 2007.

[HBSS95] T.M Hussain, A.M. Baig, T.N Saadawi, and S.A.Ahmed. Infrared pyroelectric sensor for detection of vehicular traffic using digital signal processing techniques. In *proceeding of the IEEEVehicular Technology*, volume 44, 1995.

[HF90] J. Halpern and R. Fagin. Two views of belief: Belief as generalized probability and belief as evidence. *Research report RJ 7221,IBM*, 1990.

[HGS00] H. Hu, D. Gu, and C. Co Sq. Landmark-based navigation of industrial mobile robots. *Industrial Robot: An International Journal*, 27:458–467, 2000.

[HR97] K. D. Harris and M. Recce. Neural model of a grid-based map for robot sonar. In *proceeding of CIRA 97: IEEE International Symposium on Computational Intelligence in Robotics and Automation, chair Sukhm Lee (Piscataway*, pages 34–39, 1997.

[IFSA95] H. Ishigami, T. Fukuda, T. Shibata, and F. Arai. Structue optimisation of fuzzy neural network by generic algorithm. *Fuzzy Sets and Systems*, (71):257–264, 1995.

[imoGr] designer iRobots, manufacturer of Government, and Industrial robots. `http://www.mobilerobots.com`.

[IS] I-Swarm. `http://www.shu.ac.uk/research/meri/mmvl/research/projects/i-swarm/index.html`.

[ISD] Manufacturing Engineering Laboratory Intelligent System Division. `http://www.isd.mel.nist.gov`.

[JME01] A. Jacoff, E. Messina, and J. Evans. A reference test course for autonomous mobile robots. In *proceeding of SPIE-AeroSense Conference, Orlando*, 2001.

[Jon87] A.L. Jones. Image segmentation via fractal dimension. Technical report, Air–force Institute of Technology Wright–Patterson AFB Ohio, 1987.

[Jon04] J. L. Jones. *Robot programming - a practical guide to behaviour–based robotics*. McGraw-Hill, 2004.

[Kar] A. Karperien. `http://rsb.info.nih.gov/ij/plugins/frac-lac.html`.

[KC99] K. Konolige and K. Chou. Markov localisation using correlation. In *proceeding of the IEEE International Joint Conference on Artificial Intelligence*, pages 683–699, 1999.

[KD06] J. Kidney and J. Denzinger. Testing the limits of emergent behavior in mas using learning of cooperative behavior. In *proceeding of the 7th European Conference on Artificial Intelligence (ECAI06)*, pages 260–264, 2006.

[KDFA+04] K. Konolidge, C. Ortiz D. Fox, A. Agno, M. Eriksen, J. Ko B. Limketkai, B. Morisset, D. Schulz, B. Stewart, and R. Vincent. Centibots: Very large scale distributed robotic teams. In *proceeding of International symposium on Experimental Robotics (ISER04)*, 2004.

[KH96] R. Kurazume and Sh. Hirose. Study on cooperative positioning system (basic principle and measurement experiment). In *proceeding of the IEEE, International Conference on Robotics and Automation, Minneapolis*, pages 1080–1087. IJCAI, Inc, 1996.

[KJ97] M. Krishna and J.Bares. Tethering system design for dante ii. In *proceeding of IEEE International Conference on Robotics and Automation*, 1997.

[KL01] S. Koenig and Y. Liu. Terrain coverage with ant robots: a simulation study. In *of the fifth international conference on Autonomous agents, ATM press*, pages 600–607, 2001.

[KL09] A. W. Y. Ko and H. Y. K. Lau. Intelligent robot-assisted humanitarian search and rescue system. *International Journal of Advanced Robotic Systems*, 6(2):121–128, 2009.

[KLC98] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[KR93] L. Kleeman and R.A. Russell. Thermal path following robot vehicle: Sensor design and motion control. In *proceedings of the EEE/RSJ International Conference on Intelligent Robots and systems, Yokohama, Japan*, 1993.

[KR95] O. Kaynak and I. Rudas. Application of coft computing methodologies in mechatronics. In *proceeding of the first Asian Control Conference, Japan*, pages 53–56, 1995.

[KW90] S. King and C. Weiman. Helpmate autonomous mobile robot navigation system. In *proceeding of SPIE conference on mobile robots, Boston*, volume 1388, pages 190–198, 1990.

[KW94] D. Kortenkamp and T. Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. In *proceeding of the national conference on artificial intelligence, AAAI-MenloPark, CA*, pages 979–984, 1994.

[Lab] Intelligent System Lab. `http://welcome.isr.ist.utl.pt/labs/islab/`.

[LCK95] M. Littman, A. Cassandra, and L. Kaelbling. Learning policies for partially observable environments: Scaling up. In *proceeding International Conference on Machine Learning*, pages 362–370, 1995.

[LH02] J.H. Lee and H. Hashimoto. Intelligent space-its concept and contents. *Advanced Robotic Journal*, 16(4), 2002.

[Lo89] Andrew W. Lo. Long-term memory in stock market prices. Technical Report chapter 6, NBER Working Papers, 1989.

[LS70] P.B.S. Lissaman and C.A. Shollenberger. Formation flight in birds, 1970.

[LWS07] L. Lai, Ch. Wu, and Y. Shiue. A potential field method for robot motion planning in unknown environments. *Journal of the Chinese Institute of Engineers*, 30:369–377, 2007.

[MAB+99] R. Murphy, M. Assumes, M. Bugajsk, T. Johnson, N. Kelley, J. Kiefer, and L. Pollock. Marsupial-like mobile robot societies. In *proceeding of fourth International Conference on Autonomous Agents, ACM press*, pages 364–365, 1999.

[Man93] B.B. Mandelbrot. *The Fractal Geometry of Nature*. Freedman, San Francisco, 1993.

[Mat90] Maja J. Mataric. A distributed model for mobile robot environment-learning and navigation. Technical report, Cambridge, MA, USA, 1990.

[MC09] M. Micire and J. Casper. personal communication. *ASTM Standardication News*, 2009.

[MCL+06] C. F. Marques, J. Cristovao, P. U. Lima, J. Frazao, M. I. Ribeiro, and R. M. M. Ventura. Raposa: Semi-autonomous robot for rescue operations. In *proceeding of IEEE/RSJ International conference on Intelligent Robots and systems(IROS)*, pages 3988–3993, 2006.

[MCM01] R. R. Murphy, J. Casper, and M. Micire. Potential tasks and research issues for mobile robots in robocup rescue. In *RoboCup-2000: Robot Soccer World Cup IV, Lecture keywordss in Artificial Intelligence*, pages 339–344. Springer Verlag, 2001.

[MD06] J.M.M. Montiel and A.J. Davison. A visual compass based on slam. In *proceeding of IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[Mgs] SpO_2 sensor Microsenso and gas senso. http://www.microsens.ch/products/gas.html.

[MJ07] E. Messina and A. Jacoff. Performance standards for urban search and rescue robots. *ASTM Standardication News*, 34(8), 2007.

[ML05] P.E. Merloti and J. Lewis. Simulation of artificial ants behaviour in a digital environment. In *proceeding of International Conference on Artificial Intelligence (ICAI05), Las Vegas*, 2005.

[MM01] M. Mitchell and M.Newman. Complex systems theory and evolution, 2001.

[Moo01] S. J. Moorehead. Autonomous surface exploration for mobile robots. Technical report, Carnegie Mellon University, Pennsylvenia, 2001.

[Mor76] H.P. Moravec. *A Mathematical Theory of Evidence*. 1976.

[Mor88] H.P. Moravec. *Sensor Fusion in Evidence Grids for Mobile Robots*, volume 9. 1988.

[MSf] The Mobility Open Architecture Simulation MOAST Sourceforge and Tools (MOAST) framework. `http://sourceforge.net/projects/moast/`.

[MT04] Y. Matsuo and Y. Tamura. Tree formation multi-robot system for victim search a devastated indoor space. In *proceeding of IEEE/RSJ International conference on Intelligent Robots and systems(IROS), Sendal, Japan*, volume 2, pages 1071–1076, 2004.

[Mur04] R.R. Murphy. Activities of the rescue robots at the world trade centre from 11-12 september 2001. *proceeding of IEEE Robotics and Automation magazine*, 3(11):851–864, 2004.

[MY09] S.K. Hayat M. Yim, T. Cragg. Towards small robot aided victim manipulation. In *proceeding of IEEE international workshop on Safety, Security, and Rescue Robotics(SSRR09)*, 2009.

[NBSL99] J. Nettleton, D. Barr, B. Schilling, and J. Lei. Micro-laser range finder development: Using the monolithic approach, 1999.

[NGD+05] Sh. Nouyan, R. Gro, M. Dorigo, M. Bonani, and F. Mondada. Group transport along a robot chain in a self-organised robot colony. In *proceeding of the 9 th International Conference on Intelligent Autonomous Systems, IOS*, pages 433–442. IOS Press, 2005.

[NPB95] I. Nourbakhsh, R. Powers, and S. Birchfield. An office-navigating robot. *AI Magazine*, 2(16), 1995.

[NPR+03] H.G. Nguyen, N. Pezeshkian, M. Raymond, A. Gupta, and J.M. Spector. Autonomous communication relays for tactical robots. In *proceeding of the 11th International Conference on Advanced Robotics*, 2003.

[NS04] N.Vlassis and M.T.J. Spaan. A fast point -based algorithm for pomdps. In *proceeding of the Annual Machine Learning Conference of Belgium and the Netherlands,Brussels,Belgium*, pages 170–177, 2004.

[NSCPK99] L. E. Navarro-Serment, C.J.J., Paredis, and P.K. Khosla. A beacon system for the localization of distributed robotic teams. In *proceeding of the International Conference on Field and Service Robotics*, pages 232–237, 1999.

[NT03] J. Neira and J.D. Tardos. Linear time vehicle relocation in slam. In *proceeding of IEEE International Conference on Robotics and Automation (ICRA)*, pages 427–433, Taipei, Taiwan, 2003.

[OB00] L. Ojeda and J. Borenstein. Experimental results with the kvh c-100 fluxgate compass in mobile robots. In *proceeding of the IASTED International Conference Robotics and Applications, Honolulu, Hawaii*, 2000.

[OBY06] E. Osherovich, A. M. Bruckstein, and V. Yanovski. Covering a continuous domain by distributed, limited robots. In Marco Dorigo, Luca Maria Gambardella, Mauro Birattari, Alcherio Martinoli, Riccardo Poli, and Thomas Stützle, editors, *proceeding of ANTS Workshop*, volume 4150 of *Lecture Notes in Computer Science*, pages 144–155. Springer, 2006.

[oHS08] Departement of Homeland Security. National institute of standards & technology - urban search and rescue robot performance standards, 2008.

[OMB99] M. Oussalah, H. Maaref, and C. Barret. Positioning of a mobile robot with landmark-based method. In *proceeding of the IEEE/RSJ International Conference on the Intelligent Robots and Systems, IROS97*, pages 232–237, 1999.

[Oor97] S. Oore. A mobile robot that learns its place, 1997.

[Owe09] J. Owen. How to use player/stage, `http://playerstage.sourceforge.net/doc/playerstage_instructions_2.0.pdf`, 2009.

[PGT03] J. Pineau, G. Gordon, and S. Thrun. Policy-contingent abstraction for robust robot control. In *proceeding International Conference on Uncertainty in Artificial Intelligence*, pages 447–484, 2003.

[PJS04] H. C. Peitgen, H. Jurgens, and D. Saupe. *Chaos and Fractals: new frontiers of science*. Springer-verlag, New York, 2004.

[PL07] Q. W. Pan and D. Lowe. Search and rescue robot team rf communication via power cable transmission line- a proposal. In *International Symposium, SSSE07,Montreal, Canada*, 2007.

[PNB06] N. Pezeshkian, H. G. Nguyen, and A. Burmeister. Unmanned ground vehicles non-line-of-sight operations using relaying radios. In *proceeding of the 12th IASTED International Conference Robotics and Applications, Honolulu, Hawaii*, 2006.

[Pro] European FP6 Projects. `http://www.shu.ac.uk/mmvl/research/viewfinder/` & `http://www.shu.ac.uk/mmvl/research/guardians/`.

[PSs] A networked interface to robots Player/Stage Sourceforge and sensors. `http://sourceforge.net/projects/playerstage/`.

[RDM97] Ioannis M. Rekleitis, Gregory Dudek, and Evangelos E. Milios. Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *proceeding of the International Joint Conference on Artificial Intelligence (IJCAI*, pages 1340–1345, 1997.

[Rei91] D.B. Reister. A new wheel control systems for the omnidirectional hermies iii robot. In *proceeding of the IEEE International Conference on Robotics and Automation, Sacramento*, pages 2322–2327, 1991.

[RK04] C.E. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. *Advances in Neural Information Processing Systems, MIT press*, 161(10):751–759, 2004.

[R.M04] R.Murphy. Rescue robotics for homeland security. In *Special Issue on Homeland Security*, volume 2, 2004.

[RN03] S. Russell and P. Norvig. *Artificial intelligence - a modern approach- a modern approach, 2nd ed.* Prentice Hall Series in Artificial Intelligence, New Jersey, 2003.

[Roba] White Box Robotics. `http://www.whiteboxrobotics.com/`.

[Robb] SuperDroids Robots. `http://superdroidrobots.com/ATR.html`.

[Roe08] T. M. Roehr. Control of a hierarchical team of robots for urban search and rescue. Technical report, 2008.

[Rou00] S. Roumeliotis. *Robust mobile robot localization: from single-robot uncertainties to multi-robot interdependencies*. PhD thesis, University of Southern California, Los Angeles, USA, 2000.

[Roy03] N. Roy. Technical report, 2003.

[Rsa] Technical Notes Rae system. `http://www.reasysystems.com`.

[RSb] The IEEE Robotics and Automation Society. `http://www.ieee-ras.org/`.

[RT99] N. Roy and S. Thrun. Coastal navigation – mobile robot navigation with uncertainty in dynamic environments. In *proceeding IEEE International Conference on Robotics and Automation*, pages 35–40, 1999.

[Sam84] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

[SB03] C. Stachniss and W. Burgard. Mapping and exploration with mobile robots using coverage maps. In *proceeding, international conference on Intelligent Robots and Systems(IROS), Las Vegas)*, 2003.

[SBD05] S.Thrun, W. Burgard, and D.Fox. *Probabilistic robotics*. MIT Press, Cambridge, Massachusetts, 2005.

[SBF+98] S.Thrun, W. Burgard, D. Fox, H. Hexmoor, and M. Mataric. A probabilistic approach to concurrent mapping and localization for mobile robots. In *Machine Learning*, pages 29–53, 1998.

[SBFC01] D. Schulz, W. Burgard, D. Fox, and A. B. Cremers. Tracking multiple moving targets with a mobile robot using particle filters and statistical data association, 2001.

[SC86] R.C. Smith and P. Cheesman. On the representation and estimation of spatial uncertainty. *International Journal Robotic Research*, 5(4):56–68, 1986.

[Sch97] Sven Schuierer. Efficient robot self-localization in simple polygons. In *Intelligent Robots—Sensing, Modelling and Planning*, page pages. World Scientific Publ, 1997.

[Sch01] A. Schultz. The 2000 aaai mobile robot competition and exhibition. *AI Magazine*, 22(1), 2001.

[SCs] Datasheet SensorChip Co2 sensor. `http://www.metax.co.uk`.

[SD98] R. Sim and G. Dudek. Mobile robot localization from learned landmarks. In *proceeding of IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, pages 1060–1065, 1998.

[Sed01] R. Sedgewick. *Algorithms in C++ Part 5: Graph Algorithms, 3rd eddition*. Addison-Wesley Professional, 2001.

[Sen] Microminiature Sensors. `http://www.microstrain.com/3dm-gx1.aspx`.

[SFG] C. Stachniss, U. Frese, and G. Grisetti. Openslam, http://www.openslam.org/.

[SH04] P.T. Szemes and H. Hashimoto. Fuzzy neural network based mobile agent control for intelligent space. pages 1372–1377, 2004.

[SK95] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *proceeding of IJCAI-95*, pages 1080–1087. IJCAI, Inc, 1995.

[SK97] H. Shatkay and L. P. Kaelbling. Learning topological maps with weak local odometric information. In *proceeding of IJCAI-97. IJCAI, Inc*, pages 920–929, 1997.

[SK04] J. Svennebring and S. Koenig. Building terrain-covering ant robots: A feasibility study. *Auton. Robots*, 16(3):313–332, 2004.

[SL01] S.Pletl and B. Lantos. Advanced robot control algorithms based on fuzzy, neural and genetic methods. *Journal of Advanced Computational Intelligence and Intelligent Informatics (JACIII)*, 5(2):81–89, 2001.

[SM89] C. Silva and A. Macfariane. *Knowledge- -Based Control with Application to Robots*. Springer-Verlag, 1989.

[SN04] R. Siegwart and I. R. Nourbakhsh. *Introduction to autonomous mobile robots*. MIT Press, 2004.

[SS09a] P. Saeedi and S.A. Sorensen. An algorithmic approach to generate after-disaster test fields for search and rescue agents. In *proceeding of International Conference of World Congress Engineering(IAENG-WCE), London*, 2009.

[SS09b] P. Saeedi and S.A. Sorensen. Mathematical-based benchmarking: to predict ant exploration time–series dataset. In *proceeding of IEEE International Conference on Computational Intelligence and Software Engineering (CISE09), Wuhan*, 2009.

[SS10] P. Saeedi and S.A. Sorensen. *Electronic Engineering and Computing Technology*. Springer, Verlag Berlin Heidelberg, 2010.

[SSH94] S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *proceeding of the twelfth national conference on Artificial intelligence (AAAI 94)*, pages 426–431, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[SSH09] P. Saeedi, S.A. Sorensen, and S. Hailes. Performance-directed exploration algorithm for search and rescue robots. In *proceeding of IEEE international workshop on Safety, Security, and Rescue Robotics(SSRR09)*, 2009.

[Sta04] W. Stalling. *Data and Computer Communication*. Pearson,Prentice Hall, USA, 2004.

[Sta06] C. Stachniss. Exploration and mapping with mobile robots. Technical report, 2006.

[STC04] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[Ste95] L. Steels. *The Biology and Technology of Intelligent Autonomous Agents.*, 144, 1995.

[SUA] SUAAVE. `http://www.suaav.org/`.

[SV05] M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[SWGC04] W. M. Shen, P. Will, A. Galstyan, and Ch. M. Chuong. Hormone-inspired self-organization and distributed control of robotic swarms. *Auton. Robots*, 17(1):93–105, 2004.

[TB96] S. Thrun and A. Bucken. Integrating grid-based and topological maps for mobile robot navigation. In *proceeding of the AAAI Thirteenth National Conference on Artificial Intelligence*, 1996.

[TBF00] S. Thrun, W. Burgard, and D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. In *proceeding of IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[Tea] Willow Garage Leadership Team. `http://www.willowgarage.com/pages/about-us/leadership-team`.

[Tec] Crossbow Technology. `http://www.xbow.com/Products/iproductsoverview.aspx`.

[TERT] ELROB The European Robot Trial. `http://www.elrob.org/`.

[TFU94] T. T. Fukuda and T. Ueyama. *Cellular Robotics and Micro Robotics Systems*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1994.

[Thr02] S. Thrun. Learning occupancy grids with forward sensor models. *Autonomous Robots*, 15:111–127, 2002.

[TM02] G. Theocharous and S. Mahadevan. Approximate planning with hierarchical partially observable markov decision process models for robot navigation. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1347–1352, 2002.

[TS94] P. Tsai and M. Shah. Shape from shading using linear approximation. *Image and Vision Computing*, (8):487–498, 1994.

[Ut] A Game-based Simulation of the NIST Reference Arenas USARsim tool. `http://usarsim.sourceforge.net`.

[Vos88] R. F. Voss. Fractals in nature: from characterization to simulation. *The Science of Fractal Images*, pages 21–70, 1988.

[VPnLO92] V.Genovese, P.Dario, and R Magni nd L. Odetti. Self organising behaviour and swarm intelligence in a pack of mobile miniature robots in search of pollutants. In *proceeding IEEE International Conference on Intelligent Robots and Systems, Raleigh, North Carolina*, 1992.

[VW05a] A.W.M. Voshell and D. D. Woods. Overcoming the keyhole in human–robot coordination: Simulation and evaluation. In *proceeding of the Human Factors and Ergonomics Society 49th Annual Meeting*, 2005.

[VW05b] M. Voshell and D. Woods. Breaking the keyhole in humanrobot coordination: Method and evaluation. In *proceeding of the Human Factors and Ergonomics Society 49th Annual Meeting*, pages 442–446, 2005.

[WFW95] E. Wolfart, R. B. Fisher, and A. Walker. Position refinement for a navigating robot using motion information based on honey bee strategies. In *proceeding of International Symposium on Robotic Systems(SIR), Pisa, Italy*, pages 257–264, 1995.

[WH05] T. D. Wolf and T. Holvoet. Emergence versus self-organisation: Different concepts but promising when combined. *Engineering self-organising systems: methodologies and applications*, pages 1–15, 2005.

[Wika] Exponential smoothing Wikipedia. `http://en.wikipedia.org/wiki/Exponential_smoothing`.

[Wikb] S-bot mobile robot Wikipedia. `http://en.wikipedia.org/wiki/S-bot_mobile_robot`.

[XGCS07] G. Xiong, J. Gong, H. Chen, and Z. Su. Multi-robot exploration based on market approach and immune optimizing strategy. In *proceeding of the Third International Conference on Autonomic and Autonomous Systems:ICAS '07*, page 29, Washington, DC, USA, 2007. IEEE Computer Society.

[Yam97] B. Yamauchi. A frontier-based approach for autonomous exploration. In *proceeding of the IEEE International Symposium on Computational Intelligence, Robotics and Automation*, pages 146–151, 1997.

[YVLP09] X. Yang, R. M. Voyles, K. Li, and S. Povilus. Experimental comparison of robotics locomotion with passive tether and active tether. In *proceeding of IEEE international workshop on Safety, Security, and Rescue Robotics(SSRR09)*, 2009.

[ZN98] B. F. Zhan and Ch.E. Noon. Shortest path algorithms: An evaluation using real road networks. *Transportation Science*, 32(1):65–73, 1998.

[ZZ01] N. L. Zhang and W. Zhang. Speeding up the convergence of value iteration in partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 14:2001, 2001.