

# Multi Robot Intruder Search

**Dissertation**

zur

Erlangung des Doktorgrades (Dr. rer. nat)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Mark Moors

aus

Oberhausen

Bonn (September) 2008

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät  
der Rheinischen Friedrich-Wilhelms-Universität Bonn.

1. Referent: Prof. Dr. Armin B. Cremers
2. Referent: Prof. Dr. Joachim Hertzberg

Tag der Promotion: 22.12.2008

Erscheinungsjahr 2009

Diese Dissertation ist auf dem Hochschulschriftenserver  
der ULB Bonn unter [http://hss.ulb.uni-bonn.de/diss\\_online](http://hss.ulb.uni-bonn.de/diss_online)  
elektronisch publiziert.

*For my mother*

## **Acknowledgments**

When a work takes a long time, there are often many people who gave support and helped in one way or the other.

First of all, I want to thank my Advisor Armin B. Cremers for his guidance, his encouragements and especially his patience with me and this work. I wish to thank Dirk Schulz for his support in years of this work and his help with finishing this manuscript. I also would like to thank Joachim Hertzberg for his willingness to be co-referent.

Several other people contributed to this thesis as well. I have to thank Frank E. Schneider, who supported this work from the very beginning and made it possible to put the theoretical concepts in practice. Along with him, I have to thank the Research Establishment for Applied Science (FGAN), who supported this work financially and in terms of logistic. Another thanks must go to Timo Röhling for his profound support in developing the search planner. One has to go to Volker Steinhage, who never gave up pushing me forward.

During the years, there have been many people, who gave me inspiration and help: Thanks to Wolfram Burgard, Cyrill Stachniss, Bernd Brügemann, Michael Klein, Thorsten Belker and Jürgen Schumacher.

Last but not least, I must add a deep thanks to my partner Alexandra Bäcker, who constantly kept my spirit up, even in the final phase of the work.





# Summary

The aim of this work is the development and analysis of methods and algorithms to allow a multi robot system to cooperatively search a closed, 2-dimensional environment for a human intruder. The underlying problem corresponds to the game-theoretic concept of a classical pursuit evasion game, whereas the focus is set to the generation of plans for the group of pursuers. While the main aspect of of this work lies in the field of probabilistic robotics, concepts and ideas are incorporated from differential game theory, algorithmic geometry and graph theory. The probabilistic basis allows the integration of sensor error as well as nondeterministic robot motion.

The main contributions of this work can be divided into three major parts:

- The first part deals with the development and implementation of probabilistic human models. Depending on the specific behavior of an intruder, ranging from uncooperative to unaware, different classes of intruders are identified. Models are proposed for two of these classes. For the case of a clever and uncooperative intruder who actively evades detection, we propose a model based on the concept of contamination. The second class corresponds to a person who is unaware of the pursuit. We show that simple Markov models, which are often proposed in literature, are not suited for modeling realistic human motion and develop advanced Markov models, which conform to random waypoint motion models.
- The second part, which is also the most extensive part of this work, deals with the problem of finding an uncooperative and clever intruder. A solution is presented, which projects the problem on a graph struc-

ture, which is then searched by a highly optimized A\* planner. The solution for the corresponding graph problem is afterwards projected back to the original search space and can be executed by the robotic pursuers. By means of the models proposed in the first part, the performance and correctness of the method is shown. We present experiments in simulation as on real robots to show the practicability and efficiency of the method.

- The third part deals with the problem of finding an intruder who is unaware of the search. Based on the advanced Markov model previously discussed, a greedy algorithm is proposed, which aims at maximizing the probability to find the intruder in the near future. Experimental results for this method are shown and comparisons to simpler methods are given.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Problem Definition . . . . .	2
1.3	Used terms . . . . .	2
1.3.1	Grid Maps . . . . .	3
1.3.2	Sensors . . . . .	4
1.3.3	Kinematic Constraints . . . . .	5
1.4	Area of research and contribution . . . . .	6
1.5	Structure . . . . .	7
<b>2</b>	<b>Fundamentals</b>	<b>11</b>
2.1	Related Work . . . . .	11
2.1.1	Overview . . . . .	11
2.1.2	Graph Theory . . . . .	11
2.1.3	Algorithmic Geometry . . . . .	12
2.1.4	Probabilistic Robotics . . . . .	13
2.1.5	Game Theory . . . . .	14
2.2	Multi Robot Exploration . . . . .	16
2.2.1	Single Robot Exploration . . . . .	16
2.2.2	Collaborative Exploration . . . . .	19
<b>3</b>	<b>Human &amp; Sensor Models</b>	<b>33</b>
3.1	Modeling Intruders . . . . .	33
3.1.1	Intruders without awareness . . . . .	34
3.1.2	Intruders with awareness . . . . .	41

---

3.1.3	Intruders with perfect knowledge . . . . .	42
3.2	Sensors . . . . .	45
3.2.1	Sensor Models . . . . .	45
3.2.2	Sensor Integration . . . . .	47
3.2.3	Real World Sensors . . . . .	50
<b>4</b>	<b>Metrics and Simple Search Methods</b>	<b>57</b>
4.1	Metrics . . . . .	57
4.1.1	Probability of catching the intruder . . . . .	57
4.1.2	Contamination . . . . .	59
4.1.3	Entropy . . . . .	59
4.2	Experimental Setup . . . . .	59
4.2.1	Maps . . . . .	59
4.3	Simple Search Methods . . . . .	60
4.3.1	Static Observers . . . . .	60
4.3.2	Random Walking Observers . . . . .	62
4.3.3	Fixed Walkarounds . . . . .	64
<b>5</b>	<b>Contamination Based Searching</b>	<b>71</b>
5.1	Space Decomposition . . . . .	71
5.1.1	General outline . . . . .	72
5.1.2	Computing the vertices . . . . .	72
5.1.3	Computing the edges . . . . .	74
5.1.4	Final Graph . . . . .	74
5.2	Graph Decomposition . . . . .	75
5.3	A* Planning . . . . .	78
5.3.1	Recontamination . . . . .	79
5.3.2	Heuristics . . . . .	79
5.3.3	State Hashing . . . . .	80
5.4	Merging of plans . . . . .	81
5.5	The complete contamination based planner . . . . .	81
5.6	Complexity and Scalability . . . . .	82
5.6.1	Planning Complexity . . . . .	82
5.7	Alternative Planning Space . . . . .	83
5.8	Implementation Details . . . . .	85

---

5.8.1	Nondeterministic Movement . . . . .	85
5.8.2	Time Delays . . . . .	87
5.8.3	Choosing the starting nodes . . . . .	88
5.9	Experiments . . . . .	89
5.9.1	Time based planning . . . . .	89
5.9.2	Time Based Planning II . . . . .	92
5.9.3	Wavefront Expansion . . . . .	96
5.10	Advanced Problems . . . . .	96
5.11	Additional Robots . . . . .	107
5.12	Critic & Directions of research . . . . .	108
<b>6</b>	<b>Greedy Methods</b>	<b>111</b>
6.1	Greedy methods . . . . .	111
6.2	Problem Formulation . . . . .	111
6.3	General outline . . . . .	114
6.3.1	Costs . . . . .	114
6.3.2	Utility . . . . .	114
6.3.3	Coordination . . . . .	115
6.4	The Algorithm . . . . .	115
6.5	Example . . . . .	116
6.5.1	Target point selection . . . . .	116
6.5.2	Multi Robot Searching Algorithm . . . . .	117
6.6	Evaluation and Comparison . . . . .	121
6.7	Critic & Directions of research . . . . .	123
<b>7</b>	<b>Perspectives</b>	<b>125</b>
7.1	Conclusions . . . . .	125
7.2	Future Work . . . . .	126
<b>8</b>	<b>Appendix</b>	<b>129</b>



# Chapter 1

## Introduction

### 1.1 Motivation

Security is one of the most important demands in modern societies. Consequently, public and private spending on security issues is often only excelled by spending on social security and education. In Germany alone public spending on security in form of the police force is more than 11 billion euros per year [Bun00], resulting in more than 260.000 policemen. Almost the same amount, approx. 10 billion euros, is additionally spent on private security [Gmb06], adding an additional quarter of a million private security workers.

One task in this context is the observation and monitoring of a certain closed area or building, which is often carried out by a combination of static sensors and patrolling watchmen. Due to the high demand of human labor in such a system and the resulting costs, such systems can only be found in areas and applications, where the demand for security is especially high, for example in museums or military compounds. One way to reduce the costs of such systems and increase their reliability as well as their capabilities is the use of mobile robots. Mobile robots offer a long list of possible advantages, from which we just mention a few: Robots are becoming cheaper every year and will someday be cheaper in comparison to human watchmen, they do not get bored or loose concentration, they can carry lots of different sensors and sense possible dangers much faster than humans, they cannot get hurt or injured, they can work in the dark and they can be perfectly coordinated

if working in a group.

But even if the future possibilities are promising, commercial marketing of robots, especially autonomous robots, is still in its infancy. Most of today's commercial and applied robotic security systems possess only a very limited degree of autonomy, due to safety demands as well as the complexity of their tasks. Very often, a human operator is required to analyze some sensor inputs, as well as monitor the system for safety. Accordingly such semi-autonomous systems are at the moment quite expensive and their scalability is limited, which is one cause of their limited distribution. Two examples

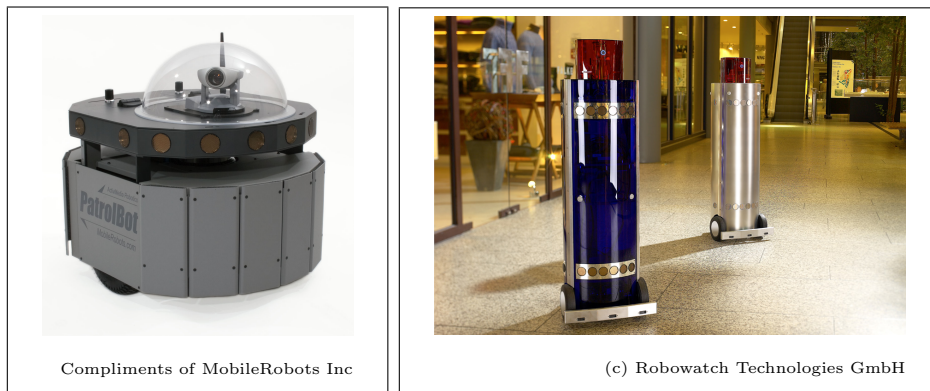


Figure 1.1: PatrolBot / MosRo1

of such systems are the Patrol Bot (Figure 1.1, left), sold by ActivMedia, as well as the Mosro1 (Figure 1.1, right) sold by Robowatch Technologies.

## 1.2 Problem Definition

The task of monitoring or searching a given area by a group of robots is naturally a very complex one and can (and should) be divided into a number of subtasks, which all have to be solved to generate a system which can autonomously perform the desired function. While discussing and giving solutions to most of the robotic tasks, the main aspect of this work is on searching, planning and coordination of a group of autonomous robots in search for an intruder.



## 1.3 Used terms

We will use a number of terms through the rest of this manuscript, which can easily be misunderstood, so we will define them at this point for clarification.

- We define the **area** to be searched as a closed, 2-dimensional structure, for example one floor of an office building or one level of a parking garage.
- The **team of robots**, also called **searchers** or **pursuers** is a team of multiple homogenous or heterogenous robots, which have at least one omnidirectional sensor, which can detect persons at a short distance and can also move through the area. One can think of a RWI-B21 robot or ActivMedia Pioneer 3 robot with two SICK-Laser-Range-Scanners as an example.
- We define an **intruder** as a human, who is not allowed to be in the area and who can or cannot actively avoid detection by the robot team. His speed is limited by human standards (so it cannot exceed 10 m/s) and he can be detected by the robots' sensors.
- We will also sometimes use the term **contamination**. Contamination is a binary attribute which can be added to a point in space, a small area (cell) or a node in a graph. The state can either be *contaminated* or *free*. Contaminated in this context means that an intruder could possibly be at the place, while free means that no intruder could be there. Since our work is mostly probabilistic, at some point we will slightly loosen this definition, so that *free* does no longer mean an absolute value of 0, but a very, very low probability.
- To avoid confusion, we will use the term **contamination level**, when the term is not used as a binary concept. A contamination level of a place means the probability that an intruder could still be at this point in space.

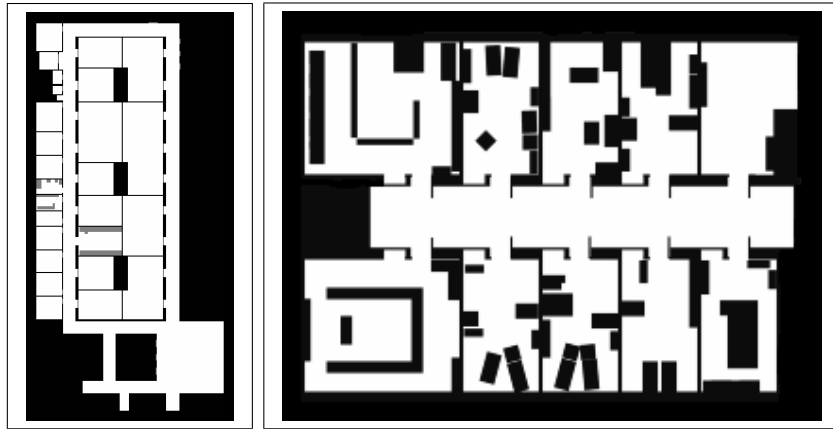


Figure 1.2: Two examples of grid maps. Left side shows a level of the the Wean Hall at CMU Pittsburgh. Right side is a floor at the Institute for applied Informatics at the University of Bonn.

### 1.3.1 Grid Maps

To generate plans for robots enabling them to do better than just reacting to their latest sensor inputs, a planning space in form of a world model is needed. While there are lots of different possibilities to choose from, grid maps [Mor88] have proven to be a good choice in many robotic applications [BBC<sup>+</sup>95][BCF<sup>+</sup>98][TBB<sup>+</sup>99]. Grid maps discretize a 2-dimensional world into equal sized, quadratic cells (grid-cells). An example would be to split up an area of 100x100 meters into 1 million grid cells, where each has the size of 10x10 cm. Every such cell gets a value assigned, which represents the probability of an obstacle in the cell. One can think of such a representation as a floor plan, which also offers the advantage of being easy to understand by a human operator. Examples of such grid maps can be seen in figure 1.2, black cells represent an occupancy value of 1, while white cells represent 0 and values between 0 and 1 are shown in grey.

Because of the fine discretization in the maps, the occupancy values are mostly binary, only in regions where the grid cells are only partly covered by obstacles, values between 0 and 1 can be found. In later chapters we will also derive smaller planning spaces from grid cells, but the world model itself

is always given as a grid map.

### 1.3.2 Sensors

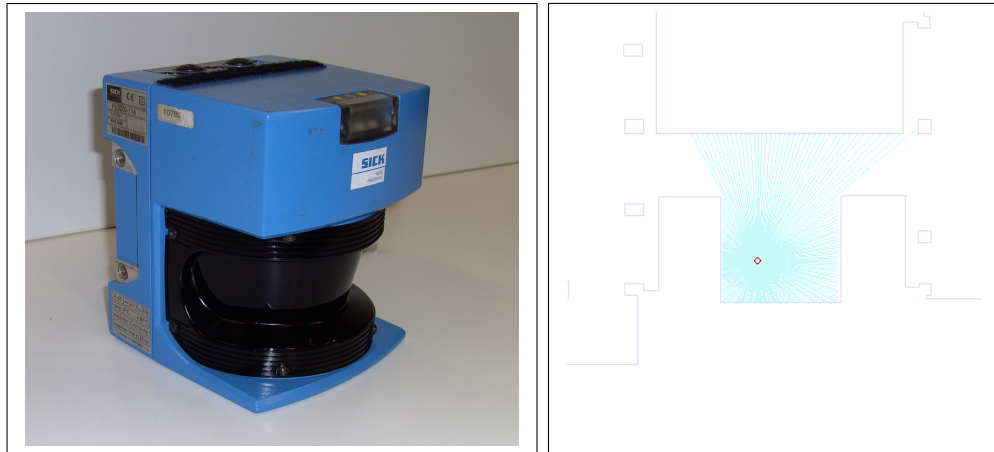


Figure 1.3: Left side shows a SICK-LMS 200 Laser-Scanner. Right Side shows a typical scan of two scanners mounted back to back on a B21 platform.

As stated before, the robot pursuers need at least one sensor to detect an intruder in its surrounding. Unless dealing with deterministic robots, such a pursuer also needs at least sensors for localization. A good choice for both requirements is the use of SICK-Laser Scanners [AG], which can be mounted back to back on many robotic platforms (including B21 and Pioneer 2/3/AT robots). Since one SICK scanner has a field of view of 180 degrees, two scanners offer almost (under the negligence of a small blanking interval) complete visibility of the surrounding area, if not blocked by an obstacle. This design has proven to offer robust localization in single [TFBF01] [BFGK98] as well as in multi robot [FBKT00] scenarios. It can also be used to robustly detect and track people over time [SBFC01] [SBFC03] and can also be used for safe navigation. We will discuss the sensors application to intruder detection in chapter 3 in more details.

### 1.3.3 Kinematic Constraints

As defined before, the basis of our work are on one hand the group of pursuers and the intruder on the other. With the idea of catching even the most smart and agile evaders in mind, it is easy to see that constraints imposed on an intruder model should be as few as possible. Consequently, in several other approaches (see next chapter) intruder motion is completely unrestricted, allowing even physically impossible speed. Since we would like to work with realistic sensor models (therefore allowing sensor error and limited sensor speed), unlimited speed would allow an intruder to "tunnel" from one place to another, without the possibility to be seen on the way. This would allow an intruder to stay hidden for an arbitrary time span, as long as the pursuers are unable to oversee the complete environment at one time. However, we are not interested in searching for entities with superhuman speeds or abilities, so it makes sense to restrict the intruder model to human standards. We will therefore assume the intruder's kinematic as holonomic with an upper speed limit of 10 m/s, which we think to be an adequate upper bound for human motion. The constraints of the pursuers on the other hand should be chosen with respect to the used robots. Since we are mostly working with B21, Pioneer 2/3, and Magellan robots, we usually assume nonholonomic robots, which can turn on a spot and do not exceed a speed of 50 cm/s.

## 1.4 Area of research and contribution

While we will discuss related work in details in chapter 2, we would like at this point to give the reader a general overview of the context of this work. A rough scheme is shown in figure 1.4, the central, white area is the field of this work.

The aim of this work is the design and evaluation of practical algorithms and techniques for coordinating a group of pursuers to locate an evader or intruder in a closed, 2-dimensional environment. As one can see in figure 1.4, this problem can be seen from many different aspects and viewpoints. Solution for pure graph-based problems exist, as well as work, which is purely based on Game Theory and Robotics. Our solutions are based upon the

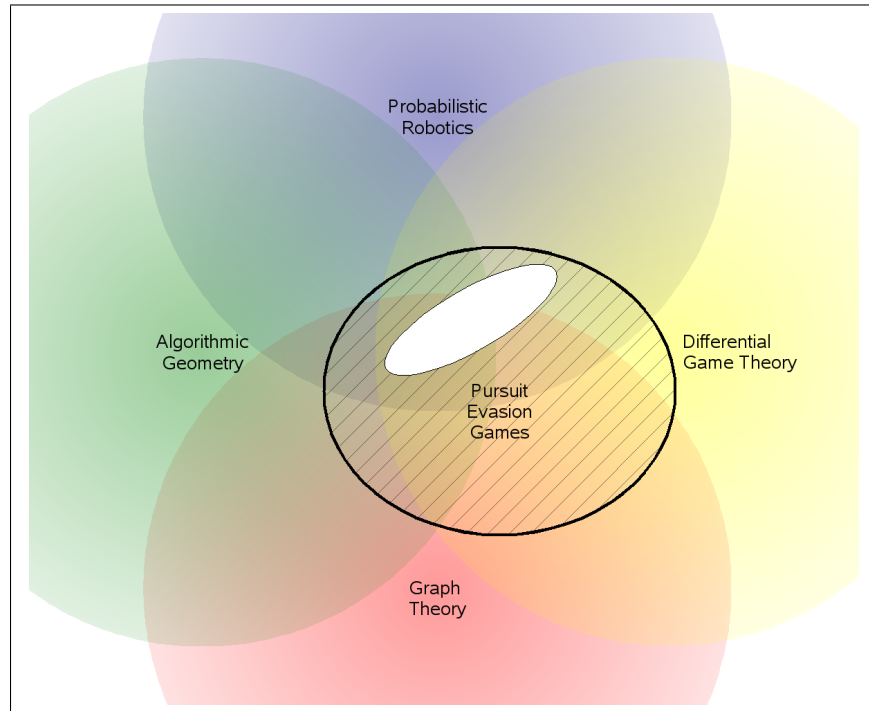


Figure 1.4: Research Context of this work. Area of main contribution is drawn in white.

integration of all four fields of research to a complete system, which not only generates plans for the pursuers, but also provides an evaluation system to measure performance.

### **Publications**

Parts of this work have already been published on international conferences.

- Multi robot exploration, which we consider a fundament for the task of multi robot intruder searching, has lead to a number of publications on conferences [BMF<sup>+</sup>00][SAB<sup>+</sup>00][BMS02] and a journal article [BMSS05].
- The first algorithm for contamination based searching, along with the

necessary human models has been published on the IROS 2005 conference [MRS05].

- The greedy search method, along with the necessary human models has been published on the IROS 2006 conference [MS06].

## 1.5 Structure

The remainder of this work is structured in 6 chapters.

### **Fundamentals**

In chapter 2 we will start with the discussion of related work. We will set this work in the context to other approaches and discuss the basic fields of research, which are needed to understand later chapters. While single-robot exploration and multi-robot exploration are not the primary focus of this work, both are vital (and fundamental) for a multi-robot-security system, we will therefore add them also to the chapter on fundamentals.

### **Human & Sensor Models**

In this chapter, our main focus is on human models as well as sensor models. Since a multi-robot-system has to plan trajectories for the robots, the planner needs not only a model of the world, but also a model, how a human intruder behaves and where he can be found. We will define three different models of the intruder, depending on his level of knowledge and discuss different implementation techniques for them. We will also need a model for the used sensors and describe the sensors effect on the intruder models.

### **Metrics and Simple Search Methods**

Before presenting the algorithms to solve the problem, we will first state some metrics to allow the evaluation of the different methods. Since the algorithms presented will be very different in nature, there cannot be one metric to compare them all. Following that, we will present some experiments based on the simple search methods, which we use in later chapters for comparison.

### **Contamination Based Searching**

As stated above, the main goal of this work is finding methods and algorithms for coordinated searching. In this chapter, we will present one method based on a worst-case intruder model, which allows a group of pursuers to catch an intruder, regardless of his behavior. The method is based on a map decomposition technique, which allows the reduction of the general problem to a graph based problem, as well as an A\* implementation for solving the resulting problem on graphs. Besides analyzing the complexity, strengths and weaknesses of the method, we will also discuss a variation of the planning component. Following that, we will show experiments to prove the results of the method and show its superiority compared to simple methods.

### **Greedy Methods**

In this chapter, we will discuss the case, in which the group of pursuers is too small to guarantee catching a very smart intruder. In this scenario worst-case estimations become of limited use. We will therefore present a technique for catching a more "average" intruder, especially one, who doesn't know of the pursuit. The method is based on the priorily discussed human models and uses a greedy approach to maximize the probability of catching an intruder in the near future. Again, we will give experimental proof of the outcome and compare the results with less sophisticated methods.

### **Perspectives**

In the final chapter, we will conclude our results. We will discuss strengths and weaknesses of the techniques and give some ideas for future research.





# Chapter 2

## Fundamentals

### 2.1 Related Work

#### 2.1.1 Overview

Searching for an unknown target in a closed environment, also often referred as a pursuit-evasion-game, has already been studied in a large number of contexts. While it is mostly impossible to sort a specific work into exactly one field of research, the problem is often discussed and examined based on one or two of the four topics: Graph-Theory, Algorithmic Geometry, Probabilistic Robotics and Game Theory.

#### 2.1.2 Graph Theory

The first discussion of the problem in the field of graph-searching was done by Breitsch [Bre67] in 1967 in the context of speleology. He proposed a man lost in a dark cave, who is wandering unpredictably and a group of searchers who are send to find him. The main question was how many men are needed, given a certain cave, to find the lost man, regardless of his behavior. Parsons [Par76] [Par78] did some early work on this question and formulated the problem on graphs. He introduced the terms search number  $s(G)$  of a graph, which is the minimum number of searchers needed to find the lost man (see figure: 2.1), and contamination, which is a binary state of a given edge or

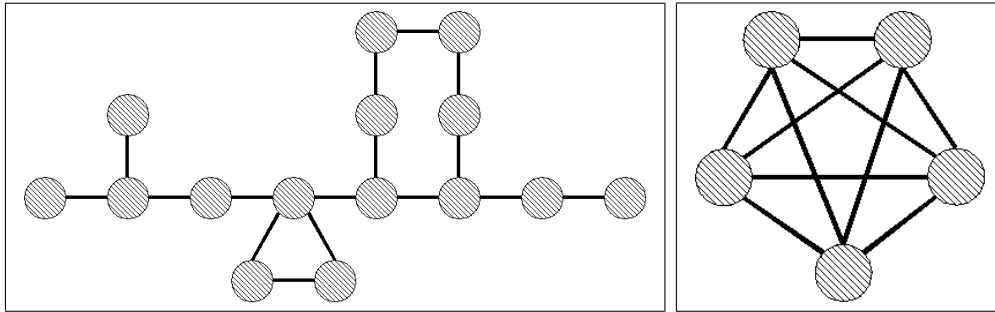


Figure 2.1: Example graph with  $s(G) = 2$  / Example graph with  $s(G) = 4$

vertex. He also showed that the continuous problem, where a searcher moves along an edge in continuous time is equivalent to the same formulation with discrete time. While showing that some special problems (i.e. fully connected graphs or trees) can be solved in polynomial time, it remained for a long time unclear how difficult the general problem is. Megiddo [MHG<sup>+</sup>88] showed 1988 that determining the search number of a general graph is at least NP-hard. Parallel to that, LaPaugh [LaP93] showed that recontamination of a graph does not help searching it (implicating that if a graph can be successfully searched by a specific number of searchers, it is always possible to find a plan for the same number of searchers to successfully search the graph without the need to recontaminate an already cleared vertex or edge), and based on the NP-hardness shown before proved the general problem to be NP-complete. Since finding a concrete solution to search a graph given a specific number of robots is at least as difficult as computing the search number (under the neglect of a linear factor), the problem of finding such a solution is also at least NP-hard.

### 2.1.3 Algorithmic Geometry

A similar problem to the above was introduced by Suzuki and Yamashita [SY92] in the context of Algorithmic Geometry. Given a closed polynomial environment they define a pursuer as a freely moving point in free space, which can cast a beam (also called a "flashlight") in any direction. An evader is considered caught if he is directly hit by the beam. The authors

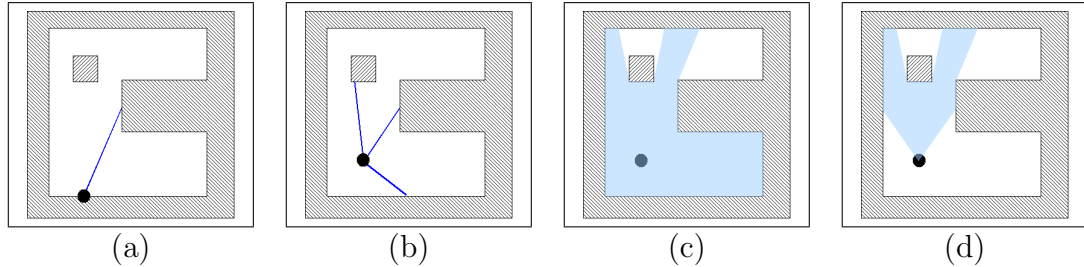


Figure 2.2: (a) 1-searcher (b) 3-searcher (c)  $\infty$ -searcher (d)  $\phi$ -searcher

introduced a number of  $k$ -searchers, where  $k$  is the number of beams, which a pursuer can emit (see figure 2.2 (a) and (b) for examples). The beams are not range limited (but are blocked by obstacles) and do not take any errors or range limitations into account. A special case is the  $\infty$ -searcher whose vision is only blocked by obstacles and can see in every direction at once (see figure 2.2 (c)). Guibas et al. [GLL<sup>+</sup>99] proved in 1999 that establishing the number of  $\infty$ -searchers for a given environment is NP-hard. A recent definition, the  $\phi$ -searcher, was added by Gerkey, Thrun and Gordon [GTG04]. It is strongly inspired by the sensor geometry of many modern robots and limits the field of view as an arc (see figure 2.2 (d)). The robot can detect any object inside the arc, regardless of range but cannot detect anything outside of his field of view. Computing the number of  $\phi$ -searchers to clear a given environment is also proven to be NP-hard [GTG04].

### 2.1.4 Probabilistic Robotics

Hespanha et al [HKS99][HPS00] suggested a probabilistic framework to address the problem of searching a closed space with a group of robots. In this framework, the complete system is described as a combination of a number or random variables, representing the states of the environment as the state of the intruder as well as the state of the pursuers. The framework allows sensor error and is also able to map the environment while searching for the intruder. For the sake of simplicity the problem of localization error is excluded. The strategy for the pursuing groups is called a policy and the authors give a greedy policy which is also shown in experiments to successfully

catch an intruder in short time. The authors also prove, that under some assumptions about the problem, the probability of finding an intruder can be raised to any value  $1 - \epsilon$  in a limited time frame ( $\epsilon > 0$ ).

While we agree with most of the assumptions, we disagree with the demand, that the probability of an evader being in a cell should not decay more than a certain amount unless a pursuer reaches this cell. This assumption is incompatible with all but the simplest Markovian motion models and would especially exclude intelligent evasion. Accordingly, the authors use a simple Markovian motion model for the evader with equal transition probabilities to all adjacent cells. We will discuss in chapter 3 in more details, why we think that these models are unsuitable for realistic human motion.

An expansion to this work was done by Vidal et al [VSK<sup>+</sup>02], who also describe an actual system, based on unmanned aerial and ground vehicles. The idea is to use the aerial vehicles (in this case small helicopters) for a fast coverage of the environment and to lead the ground vehicles to the searched target. Their framework is almost the same as the one described above and two greedy policies are discussed for catching the intruder. The idea of intelligent evasion is shortly raised but not discussed in detail, experiments are based on a simple Markovian model, as described above.

Bourgould et al [BFDW03][BFDW04] devised a similar model, which mainly focuses on the probability density function of a lost target. The target in mind is considered as drifting on the open sea and is therefore a non evading target with no own intentions and no reactions to the search itself. The emphasis of the work is how the probability density function evolves over time and how this density function can be efficiently manipulated and stored. With the idea of a floating target in mind, the density function is basically a convolution operation, but the authors identify a number of external constraints (such as obstacles and currents) which leads to realistic density distributions. Similar to the last discussed system, they design a greedy algorithm, which distributes a group of searchers (in their case small airplanes) over the search space, which maximizes the chance of finding the target in the next time step of the planning system.

### 2.1.5 Game Theory

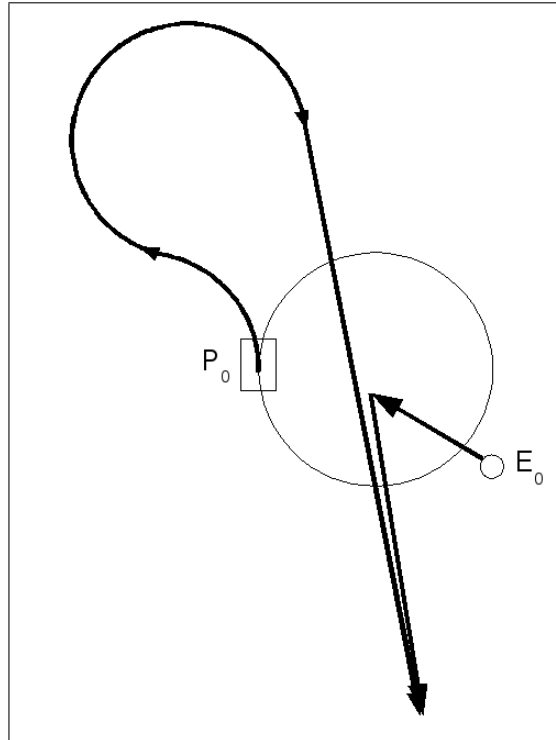


Figure 2.3: Example of a Homicidal Chauffeur Game. Evader starts at  $E_0$ , Pursuer at  $P_0$ . A naive pursuer would turn right and therefore allow the evader to enter the minimal curvature radius and evade capture. The ideal strategy for both parties is shown with thick lines, the pursuer has to reach some distance before turning toward the evader.

The term pursuit-evasion game is also often found in the context of classical game theory [Isa65][Haj75][BO99]. In this context pursuit-evasion is considered as a differential game between two players, where the interests of both players are diametrically opposed. A typical example of such a game is the Homicidal Chauffeur Problem (see figure 2.3), which was first introduced by Isaacs [Isa65]. In this game one player (the chauffeur), who is able to move fast but is limited in maneuverability, tries to overrun the other player (the runner) who is slower but has a higher maneuverability. The game can

be described by a small number of nonholonomic constraints, which limit turning radius and speed. Naturally, the question arises what strategies for the players lead to an inevitable collision. Since an important direction of Isaacs' Book was on the application of differential games in military strategy, the canonical application was found in algorithms for the interception of missiles. Other applications of pursuit-evasion were studied in the context of air traffic control [BO99] and tracking [Haj75].

There are two main differences between this field of research and our work. First, the field usually deals mostly with problems formulated in free space, geometric constraints are usually not considered. Second, most of these games are formulated as games with complete (sometimes even perfect) information. Roughly speaking, one can say, that game theory usually tries to solve the problem, how to move to catch an evader, of whom one exactly knows the position, while this work is about the question, how to move to catch an evader whose position is unknown.

## 2.2 Multi Robot Exploration

Exploring an unknown environment is a fundamental problem of modern robotics in general and also in the discussed problem of intruder searching. In order to generate plans for robots and to estimate a yet undetected intruder's position, one usually needs a model of the environment. When no ad-hoc model is given, which is often the case, the robots have to acquire the model on their own, prior to the task of intruder searching. Some authors [HKS99] suggest to do both tasks simultaneously, but we think this to be vulnerable to local optima as well as unnecessary, since the time spent to do the exploration is often insignificant to the time spent searching afterwards. We will first briefly discuss the issue of single robot exploration and then introduce the extension to multi-robot exploration.

Parts of this work have already been published on international conferences and journals, for reference see [BMF<sup>+</sup>00][SAB<sup>+</sup>00][BMS02][BMSS05].

### 2.2.1 Single Robot Exploration

Due to the fundamental nature of the problem, single robot exploration has been the focus of intensive research in the past [DJMW91][Thr93][EvP94][YSA99]. The key question in exploration is, which path a robot should move to minimize the time to completely explore the environment. Unfortunately, this problem is already NP-hard for the case of known, graph-like environments, since it can be directly mapped to the well known Traveling Salesperson Problem (TSP). But even if an efficient solution for the TSP Problem would exist, there would still be the problem of generating paths for an, at the time of planning, still unknown environment.

#### Frontier Based Exploration

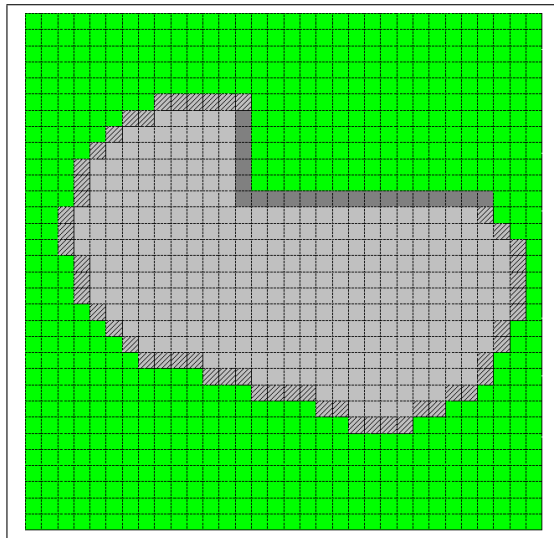


Figure 2.4: Example of a frontier. Frontier cells are drawn hatched.

A simple, but nonetheless efficient solution for this problem is a technique known as Frontier Based Exploration. In this case, the robot marks, depending on the used environment model, the areas between known and unknown space. In case of grid maps, all empty cells with a neighboring unexplored

cells are marked as frontier cells <sup>1</sup> (figure 2.4). The idea is that the robot can always generate a plan to reach a specific frontier cell and would, once he reaches the frontier, allow its sensors to expand the known space. It is easy to see, that if the environment is closed, this strategy leads to a complete exploration. While completeness is surely the main goal, the strategy also has to be efficient. Naturally, the sequence of frontier cells to explore greatly affect the total time needed for the exploration. After every expansion of the known space, the robot has to decide which frontier to explore next. A good solution for the decision problem was found by defining costs and utilities for all cells and choose the cell with the best tradeoff to be the next target. Utility is defined as the expected information gain at a cell, in this case frontier cells get an utility assignment of 1 and all non-frontier cells get 0 as assigned value. Costs can be calculated by a simple MDP planner. Value iteration, a well known dynamic programming algorithm, is usually used to solve this problem.

### Utility and cost iteration

For the special case of single robot exploration, there is a simple technique to save calculation time and make the whole implementation simpler. Instead of calculating the cost of every cell and then calculate the tradeoff, one can also extend the definition of utility to already include the costs. Therefore the definition of utility has to be slightly extended. While the frontier cells still represent the information gain and are set to 1, the non-frontier cells represent the tradeoff between the utility of the next frontier and the costs for reaching it. The first technique (cost calculation) resembles the single-source-shortest-path problem. The alternative is calculating a "multiple-target-shortest-path" problem. In case of single robot exploration, the calculation time is almost the same, while the solution based on the iteration of utility can better cope with nondeterministic robot movement. An example of both

---

<sup>1</sup>It shall be noted that the definition of a frontier is often seen in two different forms. One possibility is to define a frontier cell as a known, reachable cell with a neighboring unknown cell. The second possibility is to define frontier cell as any known cell with a neighboring unknown cell, skipping the test for reachability. In practice, both definitions lead to the same results, since an unreachable frontier cell has always higher costs than a reachable one.



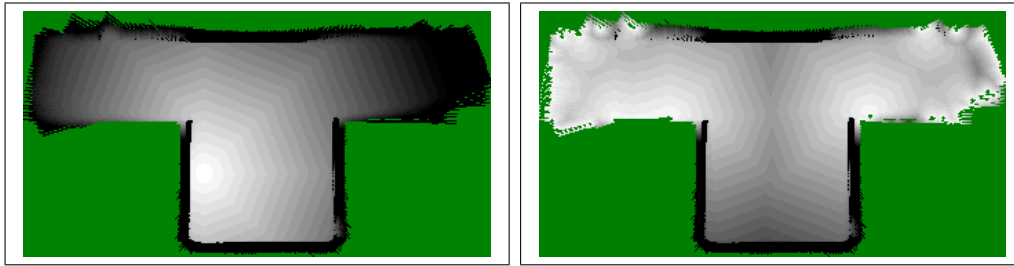


Figure 2.5: Costs and utility in a single robot exploration situation. Left: Cost map, white color denotes low costs, darker color stands for higher costs. Right: Utility map, white denotes high utility, darker color stands for lower utility.

techniques with the same starting situation can be seen in figure 2.5. In the left part, one can see the robot in the map and the calculated costs. The solution is naturally the frontier cell with the lowest costs and a path can be calculated by steepest ascent from this cell to the robots position. This path is suitable as long, as the robot does not leave this path. In the right part, one can see the same situation based on iteration of utility. In this case, the best solution is still the same, but the calculation of an explicit path is not needed, since every cell has a policy (move to the neighboring cell with the highest utility). This offers the advantage, that even if the robot deviates from a planned movement, the best solution where to go from the new position is already calculated, a recalculation of utility is only needed once a frontier has been reached. For the case of single robot exploration, the utility iteration is, due to the nondeterministic nature of robot motion, the better choice.

### 2.2.2 Collaborative Exploration

The speed of Single Robot Exploration is naturally bounded by the abilities of the used robot, for example the sensor range and the physical limits of its drive. Even if the exploration path is almost optimal, exploring a large environment can be a very slow process. One way to increase the speed of the exploration process is the use of multiple robots. Since the main goal of

this work is a multi-robot-security system, the use of multiple robots is self-evident in this context. When multiple robots have to accomplish a common task, one question of central importance is how the group should be coordinated. This raises the question to what extent the performance of a system is improved by applying coordination and if the application of a coordination scheme is justified, given that it also produces some overhead in terms of calculation time and hardware. While many coordination schemes are imaginable, we would like to base the system upon the already proven Single Robot Exploration algorithm, which also gives fair comparison between an uncoordinated and a coordinated system.

### **Implicit Coordination**

When developing an algorithm for coordinated multi-robot exploration, it is reasonable to evaluate a multi-robot exploration system without applying coordination or at least without applying an explicit coordination scheme. Before doing so, we have to clarify what we mean, when we speak of an uncoordinated system. A totally uncoordinated multi-robot system would be one, in which no information is shared at all. Since the robots have no means to know what part of the map has already been explored by others, the use of multiple robots for the task is completely superfluous, since the chances of outperforming a Single Robot System are minimal, especially since the presence of the other robots can usually only deteriorate the outcome. Therefore we allow the robots to exchange their position and map information per broadcast, so that this information is always common knowledge for the whole group. Apart from this, no communication takes place between the robots. Robots therefore maximize their own utility by going to the nearest frontier, but will not go after frontiers already explored earlier by any robot. We call this form of coordination either implicit coordination, since some coordination takes place through the exchange of map information, or simply uncoordinated. Such a system for multi-robot exploration was first proposed by Yamauchi [Yam98]. We conducted some experiments with such a system, with different group sizes, ranging from 2 to 20. An example of such an experiment, in this case with a group of 2 robots, can be seen in figure 2.6. Judging from these experiments, two observations of the performance

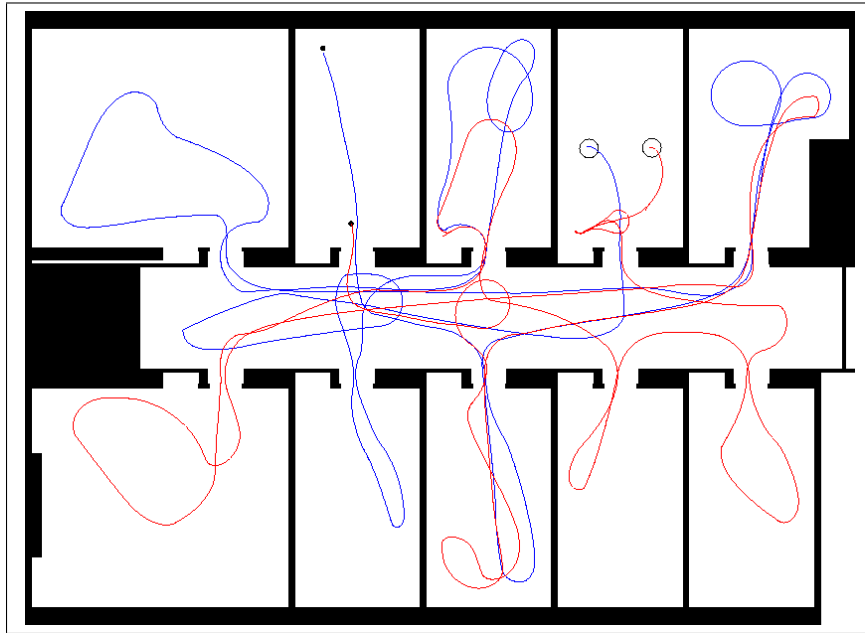


Figure 2.6: Example Experiment of implicit coordination.

can be made: Firstly, the time needed for exploration is indeed decreasing with the number of used robots. For the case of 2 robots, the group can do the task in almost half of the time that a single robot needed. Secondly, the speedup itself is decreasing with the number of robots. We also observed a certain behavior, which is (as we will see later) specific to implicit coordination. The robots sometimes tend to walk in a single file. This behavior is strongly increasing towards the end of the exploration and in the case when larger groups are used.

It is easy to see, that this behavior leads to suboptimal behavior, since the robot who goes first does the actual exploration of the frontier, while the other robots in the line just follow him and contribute almost nothing to the task. Because this behavior is seen more often in larger groups, the speedup naturally decreases.

### Explicit Coordination

One way to overcome the above mentioned limitations is the implementation of an explicit coordination scheme. Instead of letting each robot choose its own target, a central coordinator is introduced, which collects the same information as the robots and assigns all robots specific, individual targets. Effects like single file walking can be avoided. Instead of maximizing the utility/cost ratio for a single robot, the coordinator tries to maximize the utility/cost ratio for the whole group. To accomplish this goal, we have to redefine the utility of exploring a frontier. As we have seen in the last subsection, reaching a frontier, that another robot has reached before does not provide any utility at all. In contrast to the single robot case, the utility of a frontier cell is not constant until it is reached, it depends on the assignment of the other robots to the frontiers. Roughly speaking, this leads to a bipartite matching problem, where every assignment reduces the value of other assignments.

Let  $a^i$  be the coordinates of the assignment for the  $i$ .th robot and let  $U(a^i)$  be the a priori utility of the assigned coordinates. Also let  $C^i(a^i)$  be the calculated costs for the  $i$ .th robot to reach these coordinates. To estimate these reductions of utility, we introduce a suitable discount function  $f$ , which is an estimation of the range in which new information is gained at a goal point. This function is an estimator, how likely it is that a certain location is visible from a goal point. Given this function, we can define the overall utility of an assignment as: <sup>2</sup>

$$U_{overall}(a^1, \dots, a^{|R|}) = \sum_{r \in \{1 \dots R\}} \left( \left( U(a^r) \cdot \prod_{s \in \{1 \dots R\} \setminus \{r\}} (1 - f(\|a^r, a^s\|_2)) \right) - \alpha \cdot C^r(a^r) \right) \quad (2.1)$$

Maximizing this function over the goal-assignments  $a^1, \dots, a^{|R|}$  would lead to an optimal assignment for the next time step. Unfortunately, this assignment is very difficult to compute. In contrast to the well understood bipartite maximum weight matching problem (which can be solved in polynomial time), a single assignment can change the gained utility of every other priorily made

---

<sup>2</sup>For an explanation if the parameter  $\alpha$ , see subsection 2.2.2

assignment, which makes the problem far more difficult. At this point it is unknown if an optimal solution can be found in polynomial time.

To solve this problem, we developed a greedy algorithm, which proves to be working fine in practice.

The idea is to greedily find assignments for robots one at a time and then de-counting the utility of a region after such an assignment is made, so that future decisions depend on already made assignments. This idea leads to the question which robot should be assigned first. The first approach of assigning the robots in a fixed order (robot #1 before robot #2 before robot #3....) proved to be unsuitable, since it produced some bad side effects, such as robots from farther away but with a lower number "stealing" away good targets of other robots. The second approach was to assign the robots in the order of the best match in terms of utility and costs. The score of such a match is the difference between the utility and costs of reaching a certain frontier cell.

This leads to the complete coordination algorithm:

1. Determine the set of frontier cells  $\mathbf{G}$ .
2. Compute for each robot  $i$  the costs ( $\mathbf{C}^i(g)$ ) for reaching every cell  $g \in \mathbf{G}$ .
3. Set the utility  $\mathbf{U}(g)$  of every frontier cell  $g$  to 1.
4. While there is one robot without a target point:
  - (a) Determine a robot  $i$  and a frontier cell  $g$ , which satisfy:  
 $\langle i, g \rangle = \operatorname{argmax}_{\langle i', g' \rangle} U(g') - \alpha \cdot C^{i'}(g')$
  - (b) Add  $\langle i, g \rangle$  to the assignment.
  - (c) Reduce the utility of every frontier cell  $g'$  according to:  
 $U(g') = U(g') * (1 - f(\|g, g'\|_2))$

Target point Selection Algorithm

### The parameter $\alpha$

Both, the algorithm, as formula (2.1) use the parameter  $\alpha$ , which is used to adapt the unit space of the costs (which is usually given in terms of time or distance) to the space of utility (which is usually without any unit). Giving a concrete value for  $\alpha$  is difficult and naturally depends on the unit of the costs. In our implementation, using a robot, which can travel the distance of its own sensor range in roughly ten seconds, an alpha value of 0.1 proved to be suitable. Larger values give more weight to finding frontiers close to the robots, while smaller values put more weight on the distribution.

### Example Situation

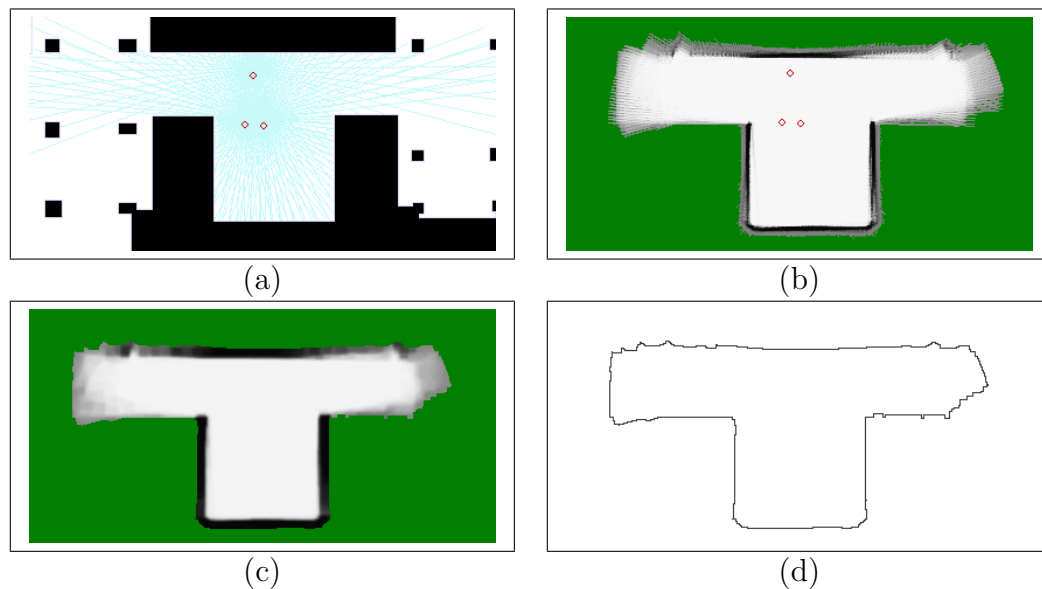


Figure 2.7: Example Situation at the beginning of an exploration experiment: (a) The situation seen from the simulator. Laser Beams are shown in cyan. (b) The map part acquired at the beginning. (c) The same map clipped and convolved. (d) The extracted frontier.

Let us give an example situation, which illustrates the different behavior. The situation is shown in figure 2.7(a) and shows a typical situation at the

beginning of an exploration. Three robots have taken initial scans and fused their data to build the map as shown in figure 2.7(b). The map is clipped and convolved to cope with nondeterministic robot movement, resulting in the map in figure 2.7(c). The initial frontiers are extracted as shown in figure 2.7(d). The next step is to calculate the costs for all three robots to reach

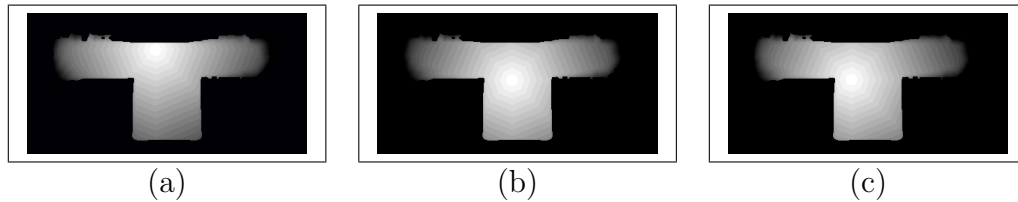


Figure 2.8: Cost functions for the three robots

any target. This is shown in figure 2.8. Until now, there is no difference between implicit and explicit coordination. Implicit coordination would now choose independently for every robot the cell with the best utility/cost ratio and make for every robot its best assignment. In this case this leads for every robot to the same assignment (figure 2.10(a)). In case of explicit coordination, the algorithm chooses the robot with the best overall utility/cost ratio. In this case, the robot on the lower left is chosen and its target point is assigned. Contrarily to implicit coordination, the utility for the frontier cells is changed. This can be seen in figure 2.9. In part (a), the basic utility for all frontier cells is the same. In part (b), the utility of the area around the target of the first robot gets reduced. Based on this new utility values, the algorithm chooses the robot with the highest utility/cost tradeoff.

This leads to the assignment of the lower right robot to a target on the other side of the area (see figure 2.10(b)). The utility around the target is reduced again and the last robot can be assigned to its target. For the sake of completeness, the final utility map is shown in figure 2.9(d), but is no longer needed for any assignment. The result of the algorithm can be seen in 2.10(b).

This example shows that the algorithm is able to divide the robots over the frontiers. The result of a complete experiment is shown in figure 2.11. By comparing the experiment shown in figure 2.6 with the one in figure 2.11,

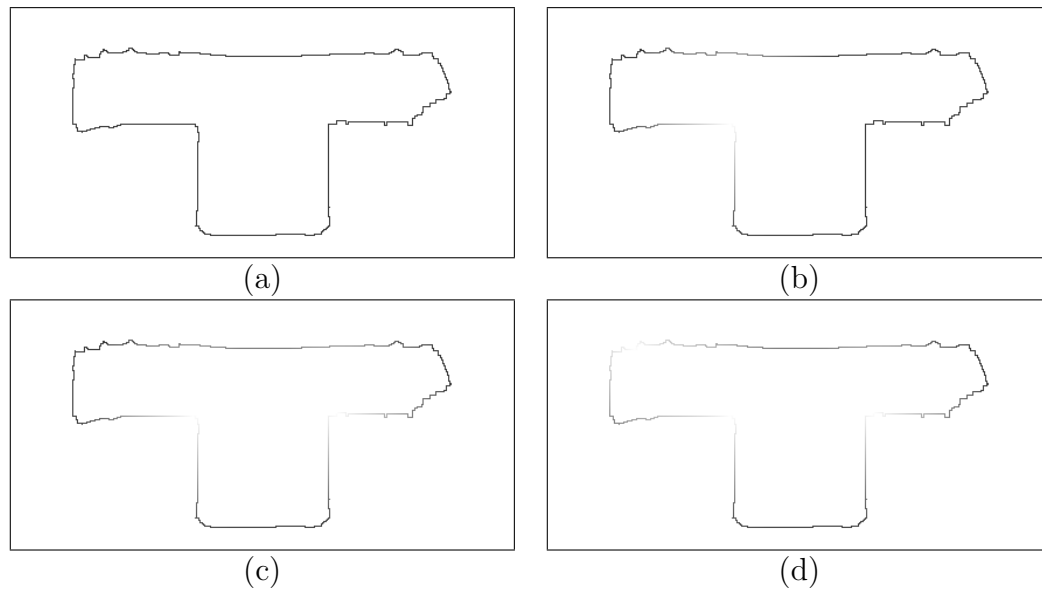


Figure 2.9: Changes of utility during the coordination algorithm: (a) Initial utility, where every frontier cell gets a high utility value (painted in black) assigned. (b) Utility after the assignment of the first robot. (c) Utility after the assignment of the second robot. (d) Utility after the assignment of the third robot.

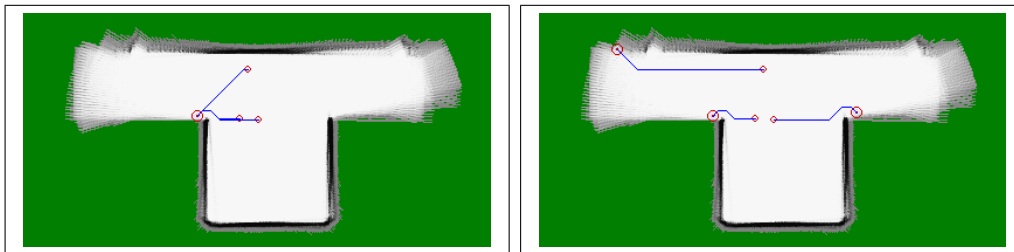


Figure 2.10: Robots and targets at the beginning of an exploration experiment: (a) Uncoordinated Behavior. (b) Coordinated Behavior.

one can easily see the superiority of explicit coordination.



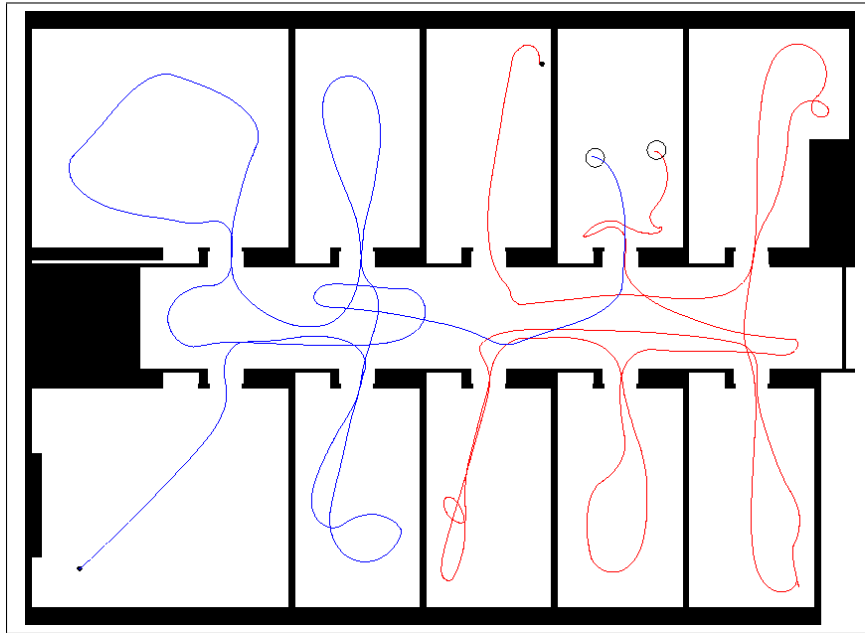


Figure 2.11: Example Experiment of explicit coordination.

### Adaptive Behavior

In the next to last subsection we introduced a suitable function  $f$  to reduce the utility around a robot's target. This function is an approximation of the probability that the sensor measures at least the given distance, which is the cumulative distribution function of the sensor measurements during an experiment.

This function is strongly dependent on the environment to explore, for example in an environment consisting of many small passages and small rooms, the probability for measuring at least 5m at a random location is very small, while in an environment consisting of large hallways the same probability at a random location would be much higher. See figure 2.12 for some functions of different environments.

Since the environment is unknown before the exploration, it has to be acquired during the experiment, so that the robots adapt their exploration to the environment. Fortunately, the statistically learned function usually con-

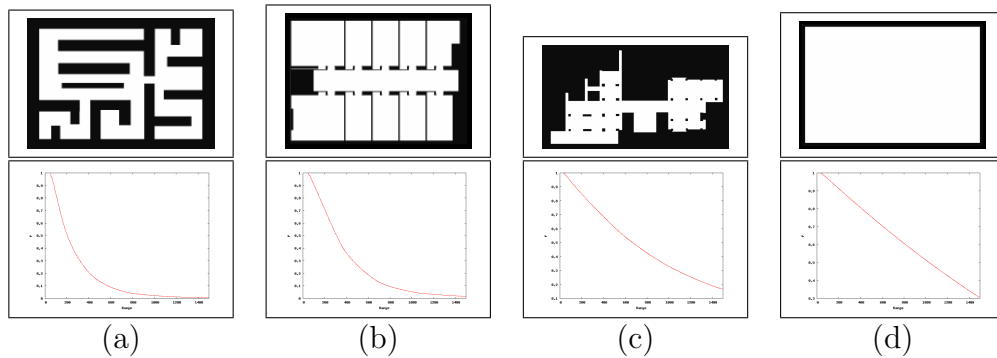


Figure 2.12: Expected Distance functions for different environments.

verges at an early stage of the exploration to the correct function. The left part of figure 2.13 shows the laser statistic during a typical exploration run, while the right part shows a probability function based on this data as well as the function calculated on the full environment. As one can see, both functions are already very similar.

### Goalpoint Exchange

As stated before, the coordination scheme is based upon a greedy strategy, since the optimal solution for equation 2.1 is difficult to compute. In some rare cases, this can lead to a situation, that the decision of one robot forces a very bad decision for another. In these seldom cases, the utility for the system can be increased if both robots exchange their goal points after the target selection algorithm is complete.

### Complexity

The complexity of the algorithm consists of three parts. Let us define  $n$  as the number of cells and the number of robots as  $r$ .

- The determination of the frontier cells can be done by applying a  $3 \times 3$  kernel over the map, which identifies a frontier cell if a cell has the state "UNEXPLORED" and has an already explored neighbor. The complexity is therefore  $9 \cdot n$ , which can be stated as  $O(n)$ .

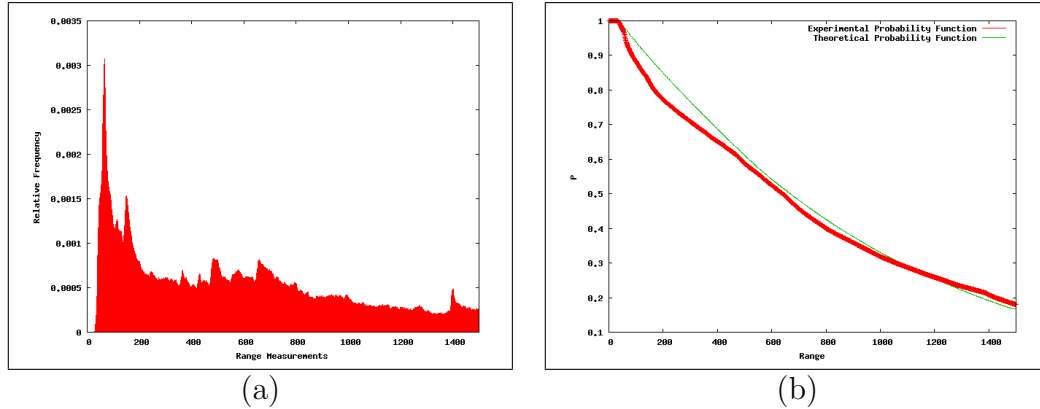


Figure 2.13: (a) Statistics over Laser-Range-Data acquired during an exploration experiment. (b) Probability function of measuring at least the given range. The thin, green line represents the function for the complete map, the fat, red line shows the function derived from the statistics shown in (a).

- Calculating the costs to reach a cell has to be done for all robots  $r$ . Since we use a deterministic version of an MDP, the planning can be done by an implementation of Dijkstra's algorithm. Because the edge structure of the map is sparse, the complexity is within  $O(n \cdot \log(n))$  for every execution of Dijkstra's algorithm. Therefore the calculation of all costs is within  $O(r \cdot n \cdot \log(n))$ .
- Calculating the robot and cell with the best utility/cost ratio can be done by simply checking every combination, which takes  $r \cdot n$  comparisons. After the assignment, the utility map has to be changed accordingly. In the worst case, depending on the sensor range, this takes  $n$  calculations. Since  $r$  robots have to be assigned this way, the complexity for the complete assignment is therefore within  $O(r \cdot ((r \cdot n) + n)) = O(r^2 \cdot n)$ .

The whole complexity of the planning algorithm is therefore within

$$O(r \cdot n \cdot \log(n) + r^2 n).$$

## Speedup

After introducing the idea of multi robot exploration, we were interested in the question: How much performance can be gained by applying an explicit coordination scheme ? This also includes an answer to the question: How much speedup can generally be gained by the use of multiple robots ? This question can not be answered for the general case, since the environment to be explored can be suited or unsuited for parallelism. For example an x-shaped environment with the robots starting in the middle is ideally suited for parallel exploration by 4 robots, while a long and narrow corridor with the starting point at one side of the corridor is completely unsuited for any kind of parallel exploration. While exact derivation is impossible, we did a number of experiments with different maps, which we consider typical indoor environments. Each experiment was a complete exploration of the environment and the total time needed was measured for comparison. Starting points were chosen at random, but for the case of multiple robot exploration, the group started at a joint location.

Results of these experiments are shown in figures 2.14,2.15 and 2.16 for three example environments. On the left side of the figures, the used map is shown, while on the right side, the results are plotted. The results themselves show the average time needed until the exploration is completed, given a group size and a coordination scheme.

From these experiments, one can make a number of observations:

- In all three tested environments, the time needed for exploration can be strongly reduced by the use of multiple robots.
- The use of an explicit coordination scheme always leads to a system which significantly outperforms an implicit coordinated system of the same size.
- The increase in performance itself is reduced with every additional robot. Depending on coordination scheme and map, a saturation point is reached with a certain team size, where additional robots only offer minimal increases in the system's performance.

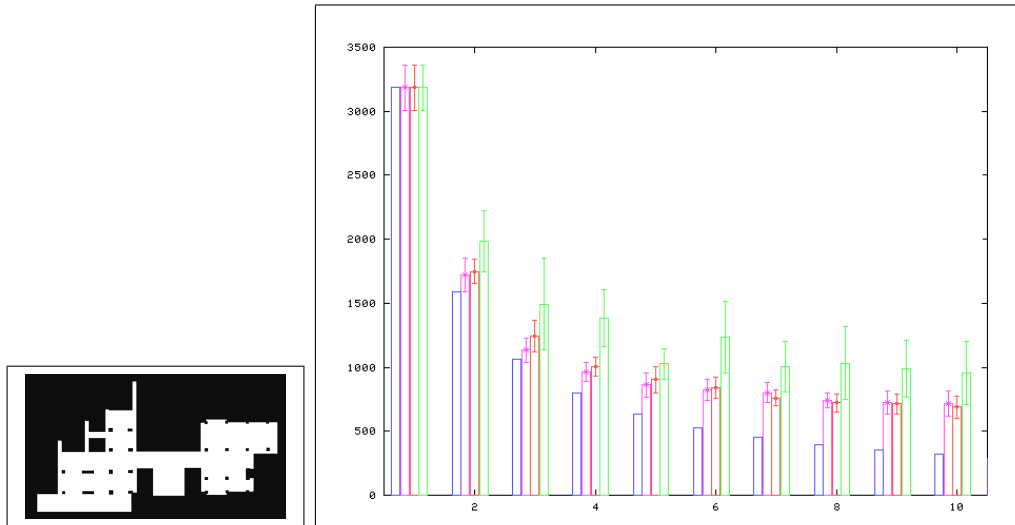


Figure 2.14: Exploration times using different group sizes and techniques. Blue: Theoretical system using perfect parallelism (comparison purpose). Pink: Explicit coordination scheme with goal point exchange. Red: Explicit coordination scheme without goal point exchange. Green: Implicit coordination.

While the first two observations justify the use of an explicit coordination scheme for multi robot exploration, the third observation is especially interesting in regard to later chapters in this work. We found that for almost all environments we used during this work, the number of robots at the saturation point is strongly correlated with the number of robots needed, if a complete sweep of the environment is required. We will come back to this observation at the end of chapter 5.

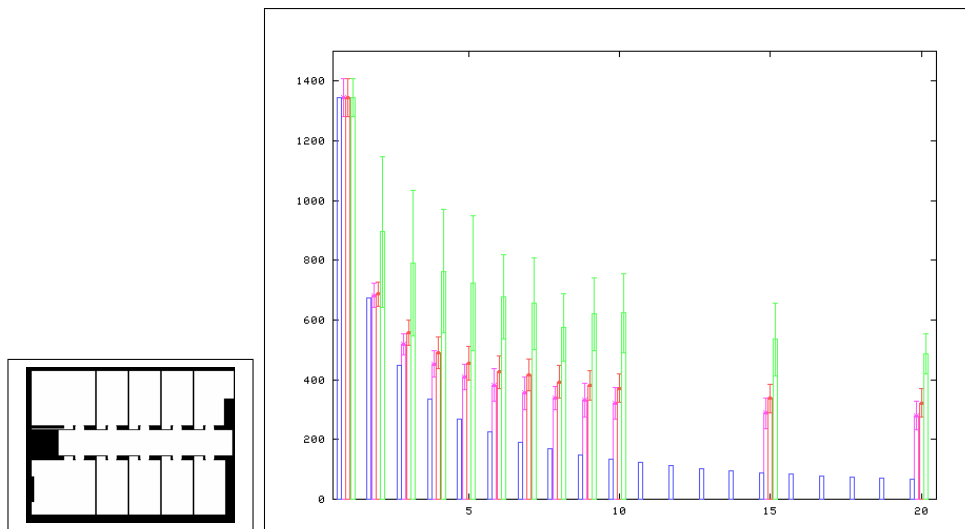


Figure 2.15: Exploration times using different group sizes and techniques. Blue: Theoretical system using perfect parallelism (comparison purpose). Pink: Explicit coordination scheme with goal point exchange. Red: Explicit coordination scheme without goal point exchange. Green: Implicit coordination. different group sizes and techniques.

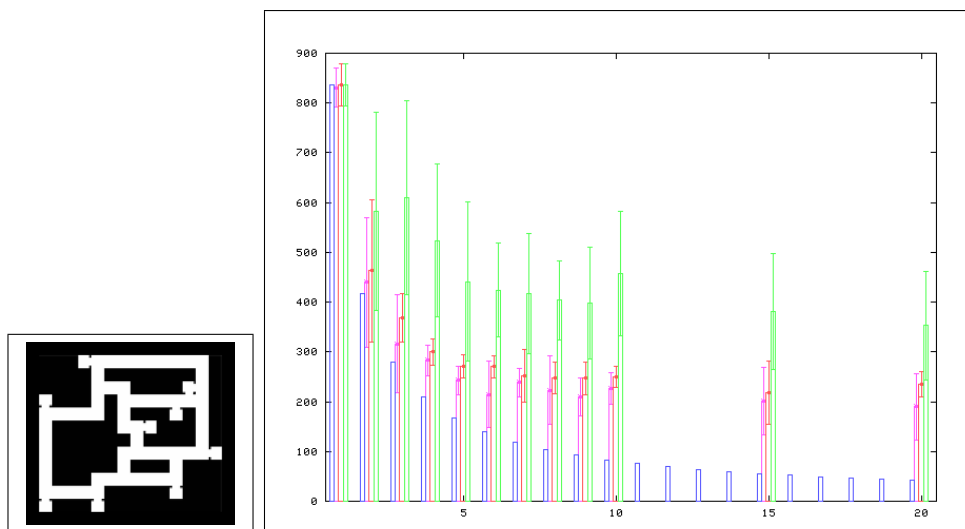


Figure 2.16: Exploration times using different group sizes and techniques. Blue: Theoretical system using perfect parallelism (comparison purpose). Pink: Explicit coordination scheme with goal point exchange. Red: Explicit coordination scheme without goal point exchange. Green: Implicit coordination. different group sizes and techniques.





# Chapter 3

## Human & Sensor Models

### 3.1 Modeling Intruders

When building a probabilistic system to look for intruders, one often has to build a model of the intruder itself. This model has to satisfy a number of constraints:

- It should model a real human intruder adequately.
- It should be simple enough to allow an on-line simulation.
- It should be complex enough, that it is able to cope for uncommon behavior of intruders.

One way to deal with these three points is to model the intruder using Hidden Markov Models (HMM), which allows us to simulate intruders in time, even without new sensor input. The most common methods for computing such models are grids on the one hand and particle based filters on the other. The interaction between the intruder and the pursuers can be broken down to two main aspects. The first is an intruder entering the field of view of a searching robot. Naturally there is a chance for the intruder to be detected. When no detection occurs, the probability for the intruder being at that point is reduced. This aspect will be discussed in details in section 3.2. The second aspect is an intruder noticing a searching robot and actively starting

to evade him, which is described in Section 3.1.2.

While there are many possibilities to build models of human intruders, we can classify the models in three classes, corresponding to their knowledge of the pursuers.

### 3.1.1 Intruders without awareness

The first class, which is often found in the literature [HKS99], leads to models of intruders not even aware of the fact, that they are pursued. These models have the advantage, that they lack a reactive component and are independent of the pursuers' motions.

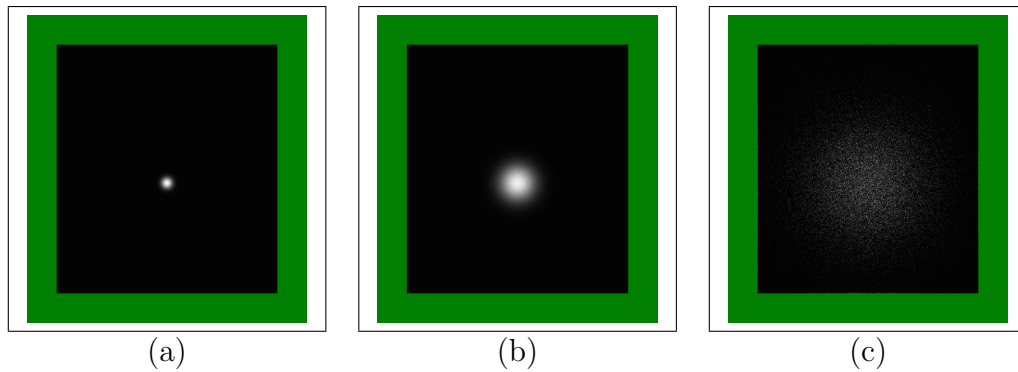


Figure 3.1: Distribution of an intruder starting in a known point using Brownian Motion: (a) After 100 Time Steps. (b) After 1000 Time Steps. (c) After 10.000 Time Steps. Lighter Colors show higher probability values.

#### Brownian Motion

Perhaps the most simple model is Brownian Motion, which is also often referred as a Wiener Process. Brownian Motion can be computed with grid cells as well as with particle filters achieving similar results. The naive approach would use grid cells, divide the whole configuration space of the intruder into even-sized grid-cells and assign to every grid-cell a variable containing the probability of an intruder being in this cell. If we know the intruder's initial

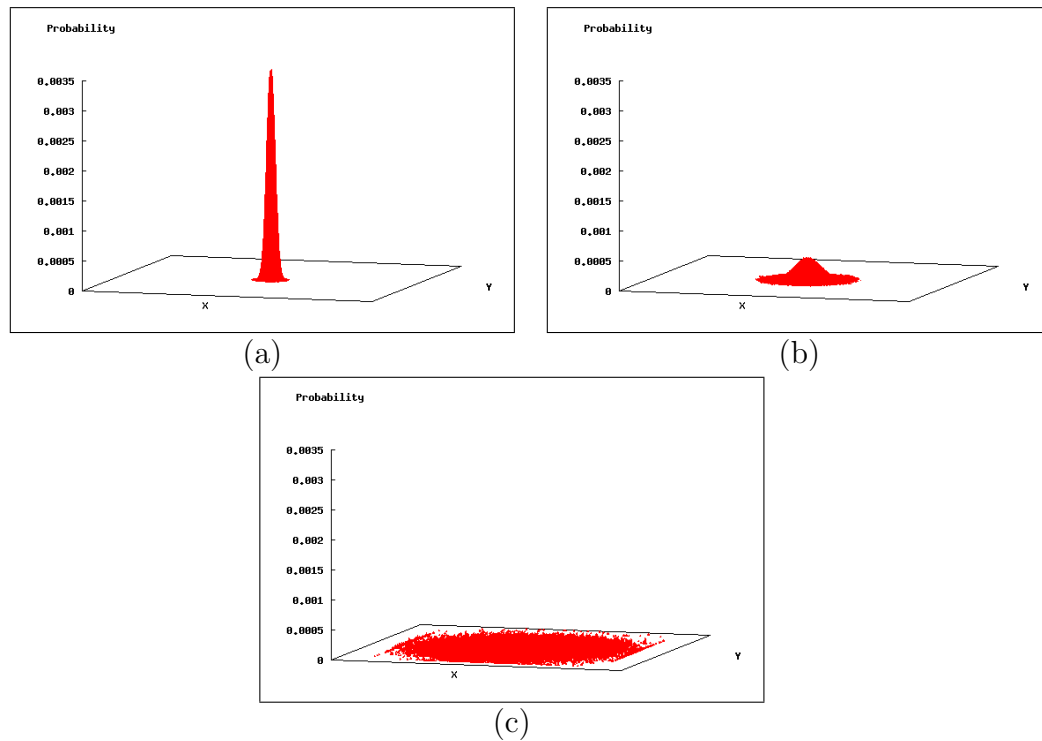


Figure 3.2: Distribution of an intruder starting in a known point using Brownian Motion as a 3D Plot: (a) After 100 Time Steps. (b) After 1000 Time Steps. (c) After 10.000 Time Steps.

position, the grid cell containing this position would start with a probability of 1. After one move of the intruder, the distribution would be an Gaussian distribution around the first cell. But since the cells don't contain any information where the intruder came from, the second move would also compute a high probability for the intruder going back to the initial cell. While being easy to compute, the approach is too simple to provide a realistic human motion model. In fact the probability of an intruder moving down a floor of 100 cells in 100 time steps would be absolutely insignificant compared to the probability of an intruder still being at the initial position. The so modeled intruder would be a person who changes his mind where to walk several times a second. A simulation of such a behavior can be seen in figures 3.1 and 3.2,

where the distribution is plotted for 100, 1000 and 10.000 time steps.

As one can see, the distribution is far from realistic, in fact the distribution is a slowly flattening Gaussian distribution with its peak at the starting point.

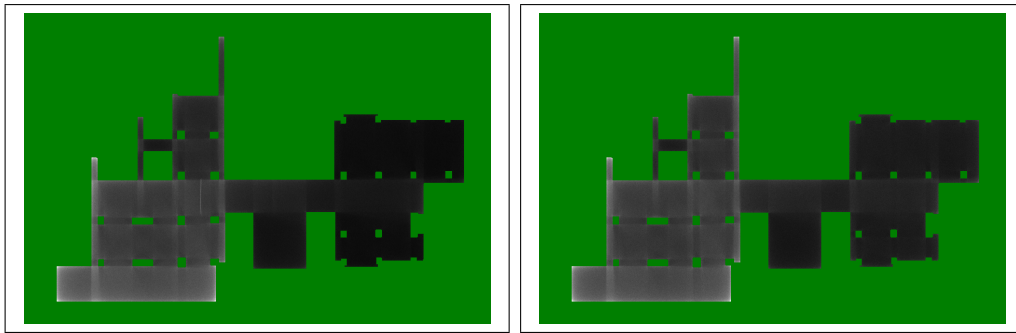


Figure 3.3: Distribution of an intruder starting in a known point using simple random walk in an open hallway. Left: Intruder distribution after 1000 time steps. Right: Intruder distribution after 2500 time steps.

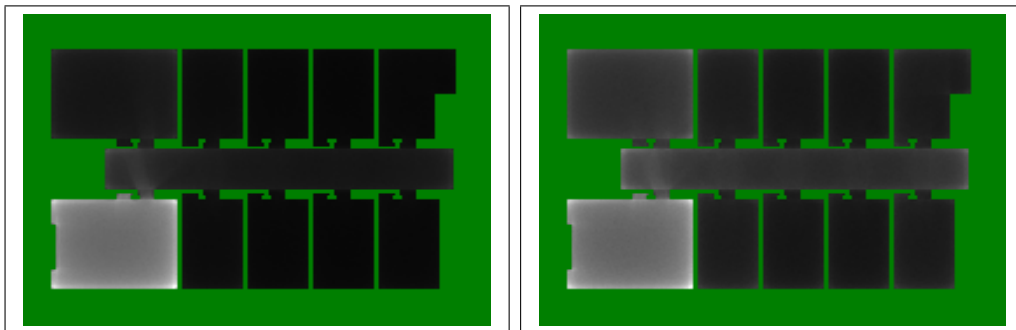


Figure 3.4: Distribution of an intruder starting in a known point using simple random walk in a map with small doors. Left: Intruder distribution after 1000 time steps. Right: Intruder distribution after 2500 time steps.

### **Brownian motion with a drift**

An extension of the simple Brownian Motion is the incorporation of a drift. This is often referred as simple random walk. A person or intruder walks

at a fixed speed, until he reaches a wall, where he changes his direction to a random new direction. This technique is best implemented with particle filters. While this leads to quite realistic distributions in maps with large halls and wide passage ways (see figure 3.3 for an example), it becomes unreasonable when a map contains small bottlenecks, such as small doors. A simulation example can be seen in figure 3.4. As one can see, even after 2500 time steps the main part of the distribution is still in the room, where the intruder started, which would be highly unlikely for a person (when directly comparing the figures, please take into account that the hallway is about 4 times the size of the office floor).

### Improved random walks with intentions

A logical extension to cope with this problem is to give a person not only a travel direction, but also an intention of reaching a certain point in space. This could be done by enlarging the state-space by multiplying it with intentions, so that every cell would not only contain the probability of the intruder being in it, but also a distribution over the intruder's intentions. Unfortunately, calculating the complete state space from time step to time step is intractable for real time simulation.

Particle filters [IB96][GSS93] on the other hand can adopt to human intentions in a straightforward and efficient way, also they adjust to the available computing power. A particle in this context is a hypothesis about the position of an intruder as well as the intention of reaching a certain target. In our work, we use particle filters with up to 10.000 samples. The particles are simulated over time, traveling through the configuration space of the intruder and are also able to react to the searching robots (what we will discuss in the next section). When a particle reaches its target (a certain point in space), it chooses a new one and starts over. Together all particles represent a distribution of the modeled intruder over the state space. So each particle gets a probability variable assigned, which is basically 1 divided through the total number of particles ( $\frac{1}{n}$ ). An example of such a filter can be seen in figure 3.5. All samples start at the same point (a) and start to distribute over the space. After a short time interval, the system reaches a stable state (e). While the particles shown in this point can give a fairly good idea of

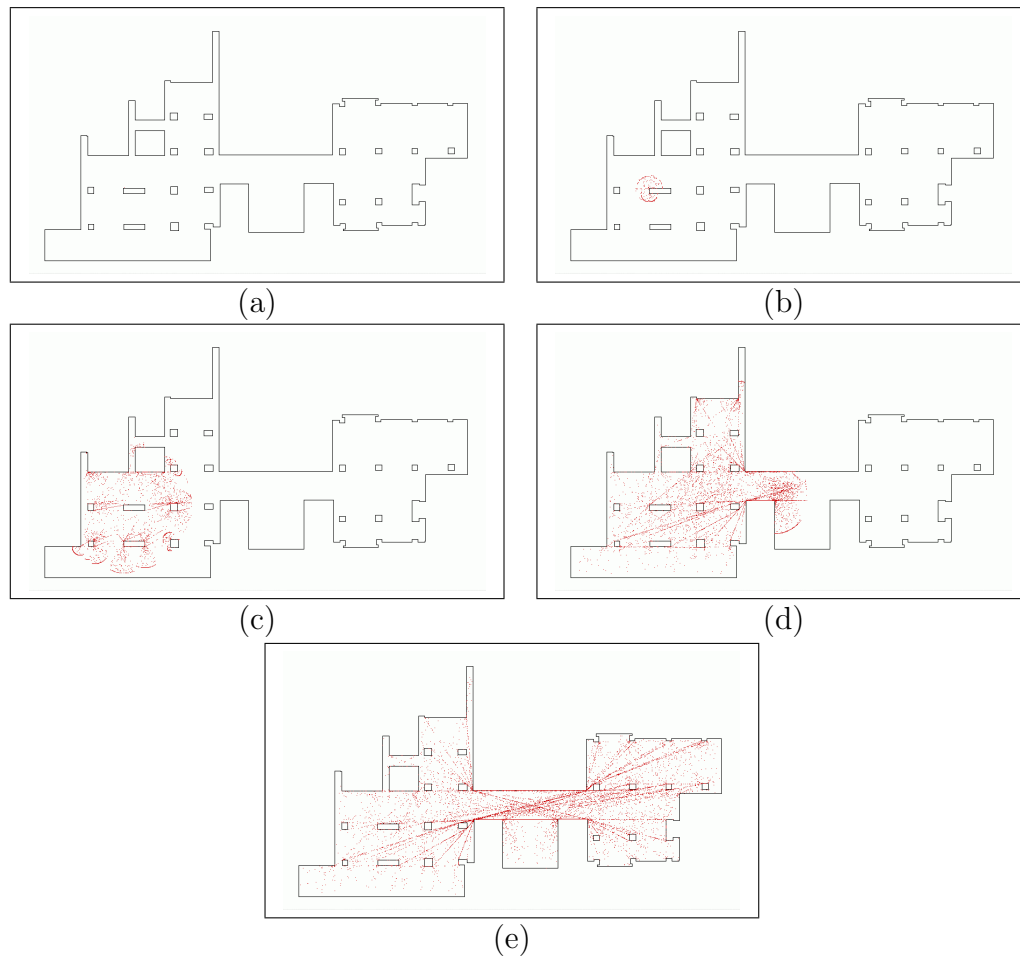


Figure 3.5: Distribution of Intruder with a particle filter

the underlying probability distribution in terms of quality, the order of magnitude in different regions is often difficult to see. We therefore projected the particles with a smoothing filter on a grid to get a better image of the probability density function, which can be seen in figure 3.6. As one can see, the resulting distribution is far more realistic than the ones acquired with Brownian motion/random walk.

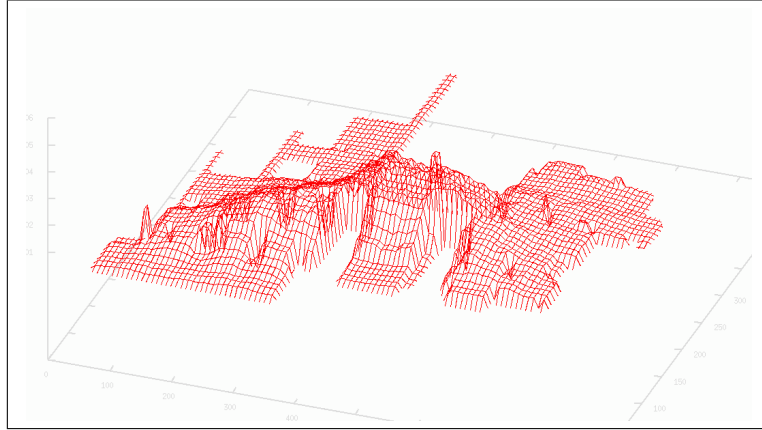


Figure 3.6: Probability Density Function of an intruder with intentions.

### Improved Markov Models

While the model from the last section leads to quite realistic distributions in the absence of pursuers, things become more difficult when pursuers are incorporated which change the distribution through the use of sensors. Especially when the number of pursuers is relatively high in proportion to the size of the configuration space of the intruder, the pursuers' sensor measurements lead to large shifts in the distribution. As a result, the distribution becomes more and more unrealistic which results from the strong reduction of the effective sample population. This will be discussed in more details in section 3.2. As a result the number of samples would have to be raised to a much larger amount, rendering the problem intractable to be calculated online. Therefore we introduce another Markov Model, which is based on a finite number of states and requires a constant calculation time per time step. We divide the space into a grid of equal-sized cells and divide each cell further into 9 possible movement directions (consisting of 8 basic directions and one which represents no movement). At first hand this looks like a contradiction to the next-to-last section, where we showed, that a model based on Brownian Motion with a drift does not lead to realistic distributions. This came from the fact, that in that context, the model was too simple in its transition function, which was only dependent on local geometry. In the new model

introduced here, we will use a transition function, which is derived from the full space configuration. For example, consider the map on the left part of

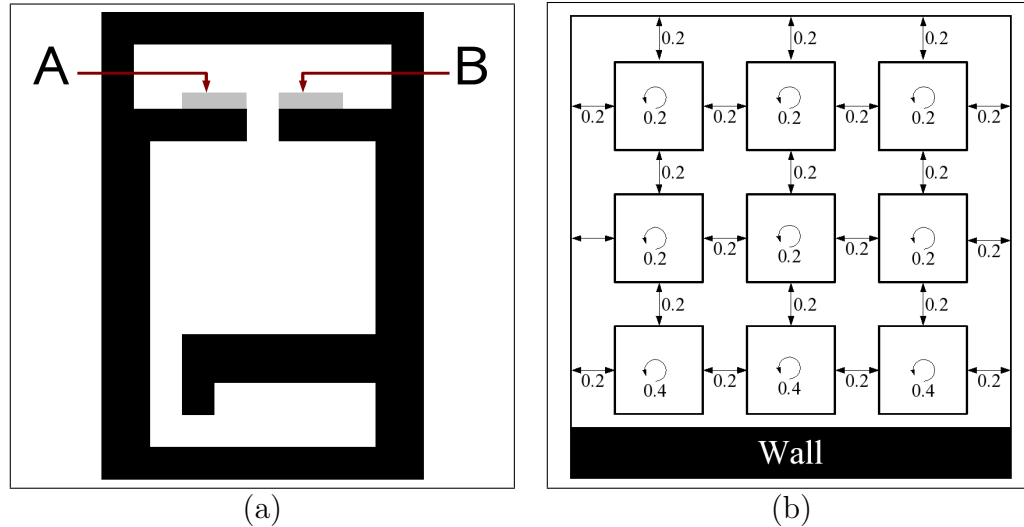


Figure 3.7: (a) Example world geometry. (b) Simple Markov Model.

figure 3.7. Now consider the grey cells, which are labeled with the letter A and B. In the simple Brownian Motion model, the cells from A and B use roughly the same transition function, which is mainly a random distribution to all neighboring cells (see right part of figure 3.7). A human on the other hand would, when standing in one of the cells of Block A, move with a high probability to the right (in direction of the door) and only with a very low probability to the left (given no ad hoc knowledge of the prior direction). A human standing in Block B would prefer to go to the left and would just in very unlikely cases walk to the right. This is due to the fact, that the human's motion intentions as modeled before is highly dependent on global geometry instead of local geometry. This can also be seen easily when one imagines the door between A and B closed, which would strongly reduce the desire in Block A to go to the right.

Resulting from these observations we conclude that the transition functions of different cells have also to be different from each other. The remaining question is how to find transition functions which leads to a plausible human



model. A simple solution would be to learn such transitions from observations. Unfortunately this would require more data, than can usually be acquired. Another solution is to use an already working model for learning the transitions. In the last section we used a sample based model, which lead to realistic distributions if the number of samples was sufficient. We can use this model to learn the needed transition model. Therefore, we simulate one sample about a very long time (letting it travel to more than 10 million random targets) and generate for every sub cell a statistic. An example for such a learned statistic can be seen in figure 3.8.

Please note that the here proposed model is deterministic, meaning a human

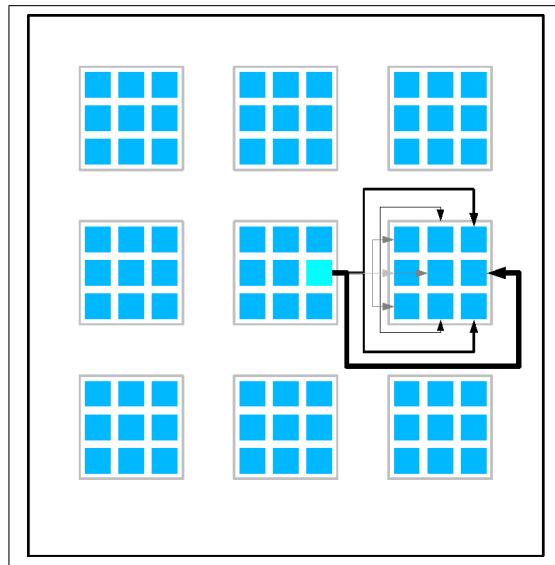


Figure 3.8: Transitions of the Improved Markov Model: Stronger lines mean a larger transition probability

with the intention of moving to the neighboring cell will always be in this cell after one time step. He might change his direction there, so every one of the 9 sub cells in the new cell can be his target. In a nondeterministic approach the statistic must also incorporate nondeterministic motions, resulting in 83 successors instead of 9.

Figure 3.9 shows simulation results of the so learned model. In this example

the discretization is set to cells of 1m x 1m, which corresponds to roughly 8000 cells. The current implementation allows in this scenario the simulation of up to 500 Time Steps in one second on a modern PC (Intel Pentium IV/3 GHz), therefore calculation time is almost negligible. As one can see, the results strongly resemble the distributions generated by the particle based model, in fact if we increase the number of particles, the resemblance gets stronger.

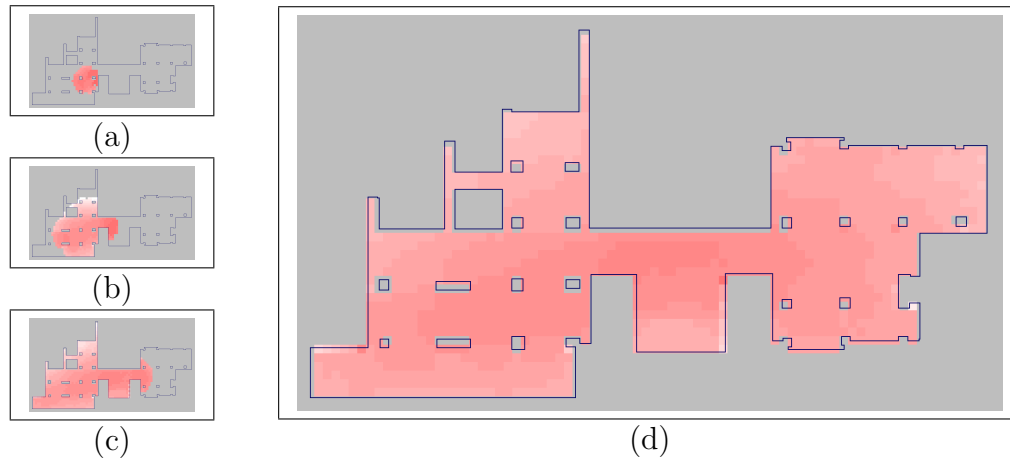


Figure 3.9: Improved Markov Model over time. (a) After 10 Timesteps (b) After 25 Timesteps (c) After 50 Timesteps (d) After 100 Timesteps

The so proposed model offers two advantages: It doesn't suffer from the effect of effective sample size depletion as the one from the last section and it can be calculated online in a limited and constant time frame.

### 3.1.2 Intruders with awareness

The second class of motion models proposes an intruder with limited knowledge of the pursuers' position and tries, according to the pursuit evasion paradigm, to avoid detection. The model has to balance between the intruder's behavior to reach its goal (again a certain position in space) and the

behavior to evade the pursuers. Therefore, the intruder reacts to the pursuers' position, when it enters a certain range to the pursuer. Normally, this range should be bigger than the sensor range of the robot (One can think of an intruder which can hear a robot in a certain distance and starts running in the opposite direction to avoid detection). While the implementation using particle filters can be done in a straightforward fashion, the complexity of such a behavior is beyond the capabilities of the simple and improved Markov Models. This is taken from the fact that even the improved Markov Model presumes intruders with a very limited memory (in fact the memory is reduced to the direction of travel). An intruder who evades a pursuer and after that re-adopts its old target must at least know its target before the evasive maneuver. The particle filter implementation can do this by remembering its target, evading the pursuer and, after leaving the area of the pursuer re-planning its path to the target. An example of the particle filter implementation can be seen in figure 3.10. As can be seen, the intruders avoid the robots in a way, that a single robot could block the whole corridor, so that no intruder could reach the empty part of the space, which is realistic for an intruder who is actively avoiding detection.

### 3.1.3 Intruders with perfect knowledge

The last class proposes an intruder, which not only knows the position of the pursuers, but also knows the pursuers' trajectories in the future. While this sounds unrealistic at first, it leads to worst case estimations of the pursuers' performance.

Pursuer algorithms based on the first and second model can lead to good performance on the average, but can fail when an intruder is smart enough to estimate the pursuers' behavior. Suppose a situation like in figure 3.11. A pursuer drives in circles around the pillar in the middle. We assume that intruder and pursuer have the same maximum speed. An intruder without or with limited awareness would be caught by the robot eventually, while an intruder with knowledge of the pursuer's behavior could outmaneuver him forever. A model based on this should lead to the result, that one robot could never catch the intruder in the scenario and that two robots are needed to achieve the goal.

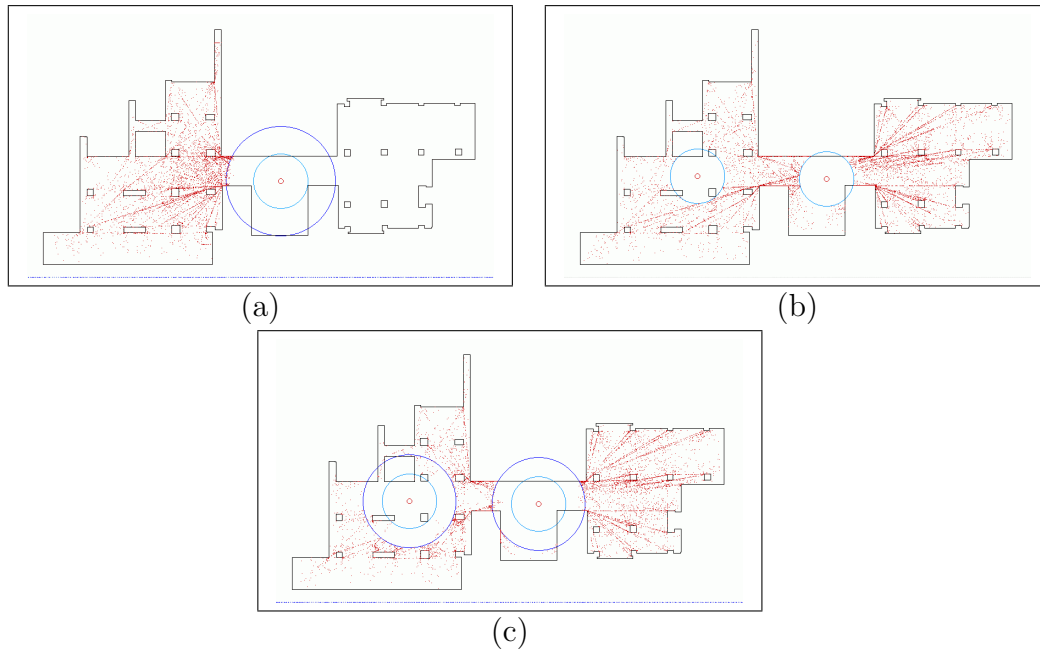


Figure 3.10: (a): Distribution of the intruder in presence of one robot and with awareness. (b): Distribution of the intruder in presence of two robots without awareness. (c): Distribution of the intruder in presence of two robots with awareness.

To achieve this, we change the modeling principle from scratch. In the previous section we modeled a distribution of one intruder over the full configuration space. Now we will model the probability of a grid cell, that a yet undetected intruder could be in it or not. While in the previous model the probability of an intruder being at a certain point in space always depended on the probability of all other cells, in this approach the probability values are mostly independent. For example, if the probability of one cell, that a yet undetected intruder could be in it, is reduced, this reduction does not affect the neighboring cell, at least not directly. The only dependency between two adjacent cells is that if the distributions are propagated over time, the probability value of a cell at time  $t$  depends on the values of the neighboring cells at time  $t - 1$ . Therefore time is discretized in (short) time steps and the

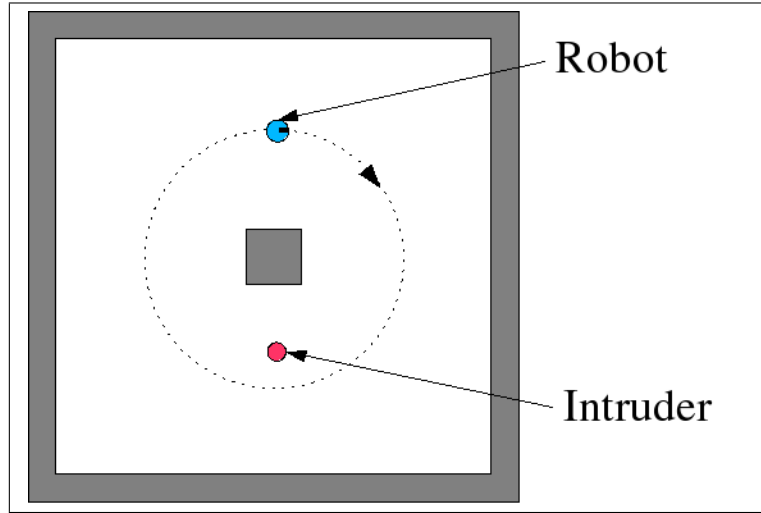


Figure 3.11: Example Situation

probability values can be updated by this formula:

$$P(C_{\text{cell}_x}^t) = \max \left\{ P(C_{\text{cell}_y}^{t-1}) : \text{dist}(\text{cell}_x, \text{cell}_y) \leq c \right\} \quad (3.1)$$

Here,  $c$  is the maximum distance the intruder can move within one time step.

It is important to note that we assume a closed world in this formula. The only explanation for an intruder being in a cell is, that he has either been in this cell before or has been in one of the neighboring cells at the last time step. A simulation of the model can be seen in figure 3.12. At time step 0 (start), we know for every cell that the probability of an undetected intruder being in this cell is very low (drawn in dark grey / green) with the exception of one cell, (in the second room on the lower side), where the probability is set to a high value (drawn in light gray / pink).  $c$  is set to 1, so an intruder can cross exactly one cell per time step. After 6 time steps, the area, where an undetected intruder could be, has increased, after 15 time steps, the area already covers part of the main corridor. After 100 time steps, the intruder could be anywhere. This form of expanding behavior is sometimes referred as a contamination model. It should be noted that contamination is normally discussed as a binary attribute, however in this context contamination can

be any value between 0 and 1, so following the definition of chapter 1, the correct term has to be contamination level model.

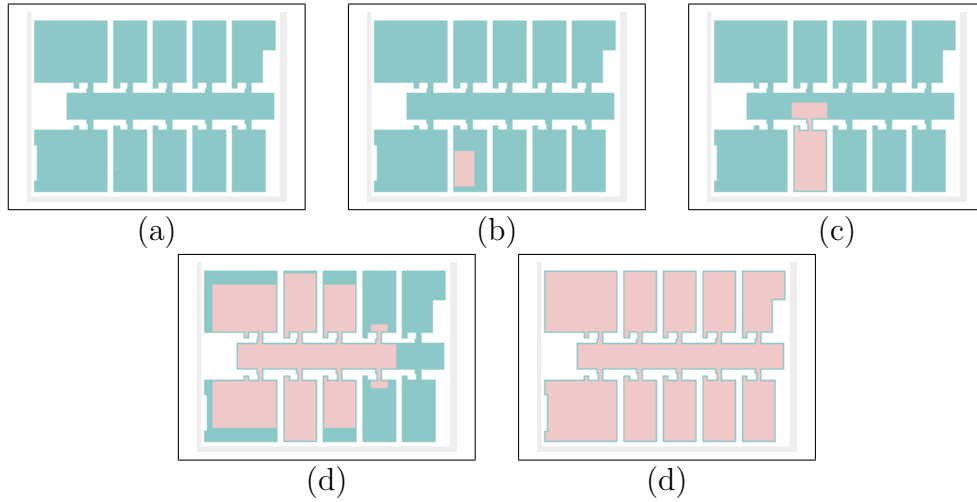


Figure 3.12: Contamination Model over time. (a) Start (b) After 6 Timesteps (c) After 15 Timesteps (d) After 50 Timesteps (e) After 100 Timesteps

## 3.2 Sensors

### 3.2.1 Sensor Models

To update the probability of the intruder models, a sensor model is needed to update the distributions. In our work, we will limit our research to the case of radial sensors with a full field of view, which are often referred as  $\infty$ -Searchers in the literature [SY92] [GLL<sup>+</sup>99]. However, the proposed models would also work with sensors with a limited field of view, as long as they are able to detect humans, but this would complicate the already difficult planning further and possibly beyond real-time-solutions.

The probability of detection of an intruder by such a sensor depends on a number of constraints such as local geometry, distance between the sensor and the intruder, the shape of the intruder as well as the underlying physical

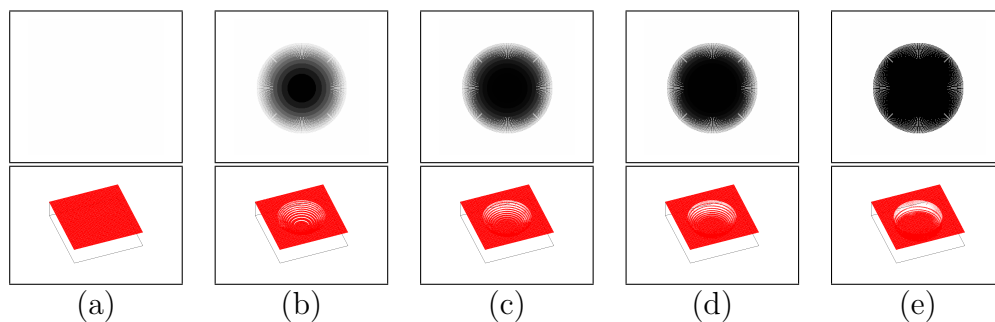


Figure 3.13: Changes through negative sensor readings in a contamination based model. (a) Without any sensor input. (b) After one negative sensor reading. (c) After two negative sensor readings. (d) After three negative sensor readings. (e) After four negative sensor readings.

principle the sensor is based on, since a real sensor is always only an approximation to a  $\infty$ -Searcher. Given an exact localization<sup>1</sup> in a given map, all these constraints can be combined in one suitable function  $f$ , which is monotonically decreasing from 1 with growing distance until a certain maximum range, where its value reaches 0. Depending on the actual sensor<sup>2</sup>, different functions can be used for  $f$ , examples can be seen in figure 3.14. Depending on the scan rate (scans per time), all plotted functions lead to roughly the same results. So while all these functions can be used, we used the linear function (the red line in the middle) for our experiments.

Taken together, we assume that each robot is equipped with a sensor that, given a clear line of sight, detects the intruder with a probability that only depends on the distance between robot and intruder. Let  $\mathcal{F}$  denote the free space of the environment and  $\mathcal{O}$  the space occupied by obstacles. We define the mutual visibility of two points  $l_1 \in \mathcal{F}$  and  $l_2 \in \mathcal{F}$  as a function

<sup>1</sup>Exact localization is necessary to reduce the constraint for the local geometry. Consider an intruder standing at a wall. With exact localization, the probability to detect him is only dependent on the distance between sensor and intruder. If the detection algorithm has to incorporate a flawed localization, it would be much more difficult to find such an intruder, because of the difficulty to distinguish the wall from the intruder. In such a case, the sensor model would become far more complex and difficult to compute.

<sup>2</sup>For example, a sensor could be very reliable on short distances and then deteriorate fast, while another sensor would lose its reliability much more slowly.

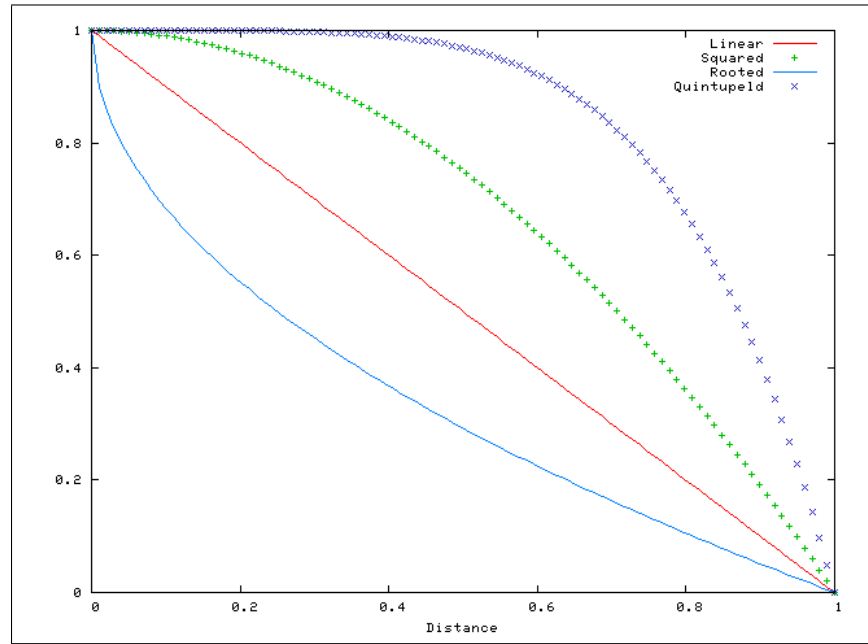


Figure 3.14: Different functions for the detection probability of a sensor dependent on range to the target

$$v(l_1, l_2) = \begin{cases} 1 & \text{if } \nexists l_3 \in \mathcal{O} : \\ & \|l_1 - l_2\|_2 = \|l_1 - l_3\|_2 + \|l_2 - l_3\|_2 \\ 0 & \text{else.} \end{cases}$$

Given the suitable probability function  $f$  which describes how the detection probability decreases based on the distance between a robot's position  $l_r$  and the intruder's position  $l_i$ , we express the probability that a single sensor measurement detects the intruder by

$$P(\text{Detect}|l_i, l_r) = v(l_i, l_r) \cdot f(\|l_i - l_r\|_2), \quad (3.2)$$

### 3.2.2 Sensor Integration

In the last section we have developed different models for humans, which can be classified in two main groups. In the first group, discussed in subsections 3.1.1 to 3.1.2, we discussed a global model, where we calculated one distri-



bution over the full space. We will call these models global human models, since they calculate one global model.

In subsection 3.1.3, we discussed a model, where every cell in space is a probability variable itself. We will call these models local human models, since the dependence between the variables is mainly local.

### Sensor integration in local human models

In order to evaluate the different search algorithms, we need to calculate the probability that a yet undetected intruder is located on a location in the environment, given the complete sequence of measurements taken by the searchers. We will use the standard grid based space decomposition as before. Given the sequence of sensor measurements  $L_{1:T} = \{L_1, \dots, L_T\}$  observed up to time  $T$ , the probability can be written as  $P(C_{xy}^T | L_{1:T})$ . All measurements are negative detections in our context. To calculate the term, we start by setting up the fraction between the term and its opposite event, which is often referred as the odds of an event. The odds approach yields the advantage that by applying Bayes rule to both the numerator and the denominator, the standardizing term, which is very difficult to calculate, cancels out. The use of Bayes Rule with the Background evidence of  $L_{1:T-1}$  leads to:

$$\frac{P(C_{xy}^T | L_{1:T})}{P(\neg C_{xy}^T | L_{1:T})} = \frac{P(L_T | C_{xy}^T, L_{1:T-1}) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T, L_{1:T-1}) \cdot P(\neg C_{xy}^T | L_{1:T-1})}$$

The probability of a measurement, given the probability of an intruder being in the cell, is conditionally independent of the older measurements. Therefore, we can simplify the terms to:

$$\frac{P(C_{xy}^T | L_{1:T})}{P(\neg C_{xy}^T | L_{1:T})} = \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})} \quad (3.3)$$

Further derivation leads to:

$$\begin{aligned} \frac{P(C_{xy}^T | L_{1:T})}{1 - P(C_{xy}^T | L_{1:T})} &= \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})} \\ P(C_{xy}^T | L_{1:T}) &= \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})} \cdot (1 - P(C_{xy}^T | L_{1:T})) \end{aligned}$$

$$P(C_{xy}^T | L_{1:T}) = \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})} - P(C_{xy}^T | L_{1:T}) \cdot \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})}$$

$$P(C_{xy}^T | L_{1:T}) + P(C_{xy}^T | L_{1:T}) \cdot \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})} = \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})}$$

$$P(C_{xy}^T | L_{1:T}) \cdot \left(1 + \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})}\right) = \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})}$$

$$P(C_{xy}^T | L_{1:T}) = \frac{\frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})}}{\left(1 + \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})}\right)}$$

$$P(C_{xy}^T | L_{1:T}) = \frac{\frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})}}{\frac{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1}) + P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})}}$$

$$P(C_{xy}^T | L_{1:T}) = \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1}) \cdot P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})}{\left(P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1}) + P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})\right) \cdot P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1})}$$

$$P(C_{xy}^T | L_{1:T}) = \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{\left(P(L_T | \neg C_{xy}^T) \cdot P(\neg C_{xy}^T | L_{1:T-1}) + P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})\right)}$$

To simplify things further, we assume that there are no false positive detections and set  $P(L_T | \neg C_{xy}^T)$  to 1, leading to:

$$P(C_{xy}^T | L_{1:T}) = \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{1 - P(C_{xy}^T | L_{1:T-1}) + P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}$$

$$P(C_{xy}^T | L_{1:T}) = \frac{P(L_T | C_{xy}^T) \cdot P(C_{xy}^T | L_{1:T-1})}{1 - P(C_{xy}^T | L_{1:T-1}) \cdot (P(L_T | C_{xy}^T) - 1)} \quad (3.4)$$

This formula can be used to update the belief state for each cell after each measurement based on the belief state before. Naturally, the first measurement requires a prior. This prior can be any value between 0 and 1, but

must never be 1, since this would lead to infinite odds and render the measurements useless, since we would surely know of an intruder in that cell. A prior of 0 is possible, meaning a cell, where we surely know that no intruder could be in it.

In a normal setting, where we do not know anything about an intruder in an area, we would initialize all cells with a prior of 0.99. While this formula is accurate, it is often desired to quickly calculate this probability update at the cost of accepting a small error. A possible solution to this is the use of the formula:

$$P^*(C_{xy}^T | L_{1:T}) = P(L_T | C_{xy}^T) \cdot P^*(C_{xy}^T | L_{1:T-1}) \quad (3.5)$$

The differences (and therefore the error of  $P^*$ ) of both functions can be seen in figure 3.15<sup>3</sup>. One can see that the difference between both formulas is quite small, in fact the influence of the sensor model is much higher than the choice between (3.4) and (3.5).

### Sensor integration in global human models

The integration in global human models can directly be derived from the previous section with relative ease. Before doing so, one has to understand the modelling difference between both approaches. In the local model, every cell had an independent probability value for an undetected intruder being there. No statement is made about the number of intruders, there could be none, one or any number of intruders in the area. In the global model, there is exactly one intruder in the area, he never leaves it and is at every time step in a specific cell. In the local model, the probability values of a given cell only depend on measurements of the cell, hence the only explanation for a change is either a measurement of the cell itself or a movement. In the global model, the values can change, if another cell is measured. For example, consider a world consisting of two cells with an equal distribution between those two, i.e. the probability of both cells is 0.5. Now consider a (negative) measurement of the first cell. Since we know for sure that an intruder is in

---

<sup>3</sup>Please note that the functions themselves are discrete, they are only drawn with lines to make them more comprehensive.

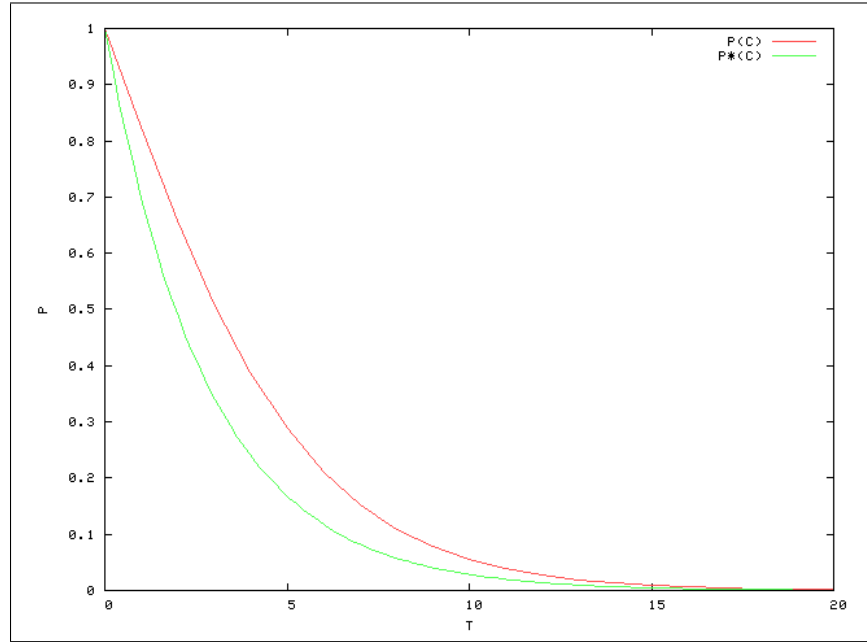


Figure 3.15: Example of the Probability update of a cell, which is consecutively scanned with a constant sensor model of  $P(L_T | C_{x,y}^T) = 0.7$ .

the area, the probability of the second cell has to increase. Therefore, the distribution has to be normalized after each sensor integration. This can be done by redefining the formulas from the last section:

$$\begin{aligned}
 P_g(C_{xy}^T | L_{1:T}) &= \alpha \cdot \frac{P_g(L_T | C_{xy}^T) \cdot P_g(C_{xy}^T | L_{1:T-1})}{1 - P_g(C_{xy}^T | L_{1:T-1}) \cdot (P_g(L_T | C_{xy}^T) - 1)} \\
 &= \frac{P_g(L_T | C_{xy}^T) \cdot P_g(C_{xy}^T | L_{1:T-1})}{1 - P_g(C_{xy}^T | L_{1:T-1}) \cdot (P_g(L_T | C_{xy}^T) - 1)} \\
 &= \sum_{rs} \frac{P_g(L_T | C_{rs}^T) \cdot P_g(C_{rs}^T | L_{1:T-1})}{1 - P_g(C_{rs}^T | L_{1:T-1}) \cdot (P_g(L_T | C_{rs}^T) - 1)} \quad (3.6)
 \end{aligned}$$

This is the formula to exactly determine the global probability values. Similar to our approach in the last section, we also give a faster, approximate

formulation:

$$\begin{aligned}
 P_g^*(C_{xy}^T | L_{1:T}) &= \alpha \cdot P_g(L_T | C_{xy}^T) \cdot P_g^*(C_{xy}^T | L_{1:T-1}) \\
 &= \frac{P_g(L_T | C_{xy}^T) \cdot P_g^*(C_{xy}^T | L_{1:T-1})}{\sum_{rs} P_g(L_T | C_{rs}^T) \cdot P_g^*(C_{rs}^T | L_{1:T-1})} \quad (3.7)
 \end{aligned}$$

### The Normalizer

The normalizer from the last section can also serve to calculate an estimate of the probability to catch an intruder at a given time step. We will come back to this in the discussion of evaluation techniques (section 4.1.1).

### 3.2.3 Real World Sensors

The sensor model used in the previous sections has been abstract. This offers the advantage that the discussed algorithms later in this work can be used with different sensors, as long as they fit the abstract description. For an actual implementation we have to specify a real sensor.

#### Laser Scanners

Since the searchers have to actively move through their surroundings, sensors for navigation and localization are needed, independent of the question which sensor to use for the detection of intruders. As stated in the introduction (1.3.2) our sensor of choice is the SICK-LMS Laser-Scanner, which is a reliable and precise sensor found on many robotic platforms. To meet the requirement of an  $\infty$ -searcher, we have to mount 2 scanners back-to-back on one platform under the neglecting of the blanking gap between the scanners. In the before-last section, we defined a general suitable function  $f$ , which gives the probability of detecting an intruder at a given range, which naturally depends on the sensor. While detecting and tracking persons in laser scans is not in the scope of this paper (see [SBFC03][SBFC01] for more details on the issue), we have to briefly discuss the idea how to detect a person in a scan.

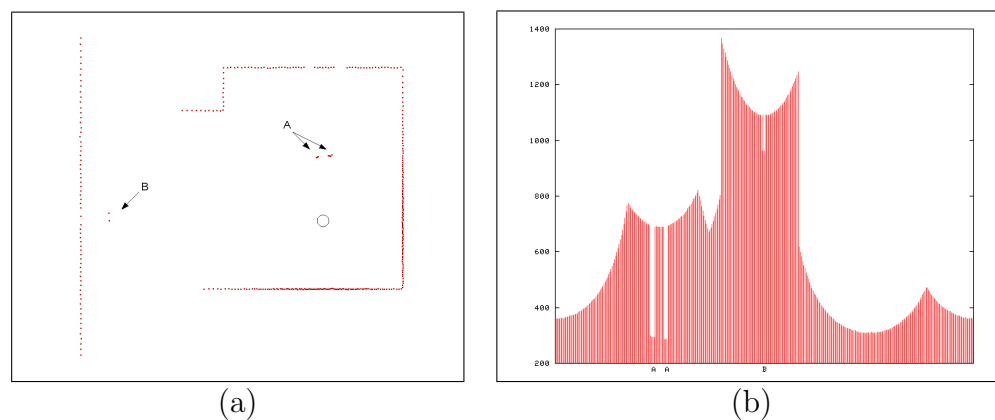


Figure 3.16: Example of a laser scan. The left part shows the laser scan in cartesian representation with 2 persons labeled A and B, the right part shows the same scan as pure data.

A standard SICK Scanner (PLS or LMS) uses an angular resolution between  $0.5^\circ$  and  $1^\circ$  per beam and is in many robotic systems mounted at a height of about 35 cm from the ground to allow collision free navigation. At this height, only the legs and knees of an intruder are seen by the sensor. So to successfully detect a person in the scan, one has to decide if the reflection of a laser beam was caused by the environment or the legs or knees of a person.

The basic idea is to search the laser scan for specific features, which are caused by human legs. Depending on the distance to the sensor, a leg causes a small group of neighboring beams to be roughly the same range, whereas the derivatives at the edges of this group are usually high. To better cope for false positives (also called false alarms), one can also take the background map into account and exclude all features, which are in the direct vicinity of an obstacle. Human legs have a diameter of roughly 15 cm. If we need at least three laser beam hits to define an alarm, the intruder has to be no more than 4.3 m from the sensor to always raise an alarm, given the SICK resolution is set to  $1^\circ$ . If we use a resolution of  $0.5^\circ$ , the intruder has to be in the range of 8.6 m to be successfully detected. Based on this observations we decided to take 5 m as the maximum range for the function  $f$ . An example can be seen

in figure 3.16. The scan contains two persons, which are labeled in the left part as A and B. Person A can easily be detected, since the features show a small group of scan-end-points with similar range and a high derivative at the endpoints. Person B cannot be successfully detected, due to the range between robot and person, only one beam is reflected from each leg. Since a single reflection could also be a sensor error, no alarm can be raised.

While the sensor proves to be a good choice for navigation, it has some limitations in the case of detecting intruders, which can become a problem if an intruder knows about the specific weaknesses of the sensor.

- The sensor can be fooled by black fabric, since the reflected light becomes too weak to be detected. We found this effect very rarely with black trousers.
- The intruder could evade the laser beam by stepping on an object, for example a table.
- The intruder could press himself against a wall, which makes it very difficult to discriminate him from sensor error.

### IR Cameras

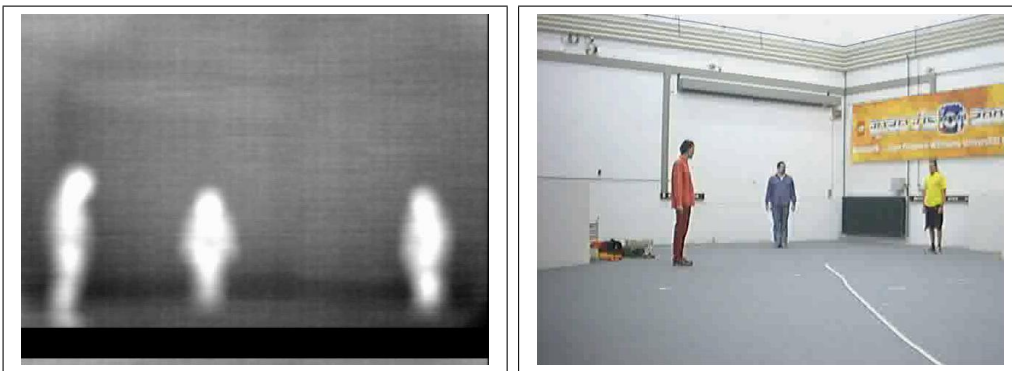


Figure 3.17: Example of a thermographic picture. The left part shows the actual IR-picture, the right part shows the same scene taken by a webcam.<sup>5</sup>

An alternative to the Laser Scanner is the use of infrared sensitive cameras, also often referred as thermographic cameras. IR-cameras offer the advantage to detect humans in direct line of the sensor quite easily, as long as the environment is colder than the human itself (which is usually the case since human body temperature is normally around  $37^{\circ}$  Celsius.). An example picture of such a camera can be seen in figure 3.17<sup>4 5</sup>. As one can see, the three human shapes can be easily subtracted from the background, making the detection of humans a simple task in comparison with a laserscanner or a normal camera.

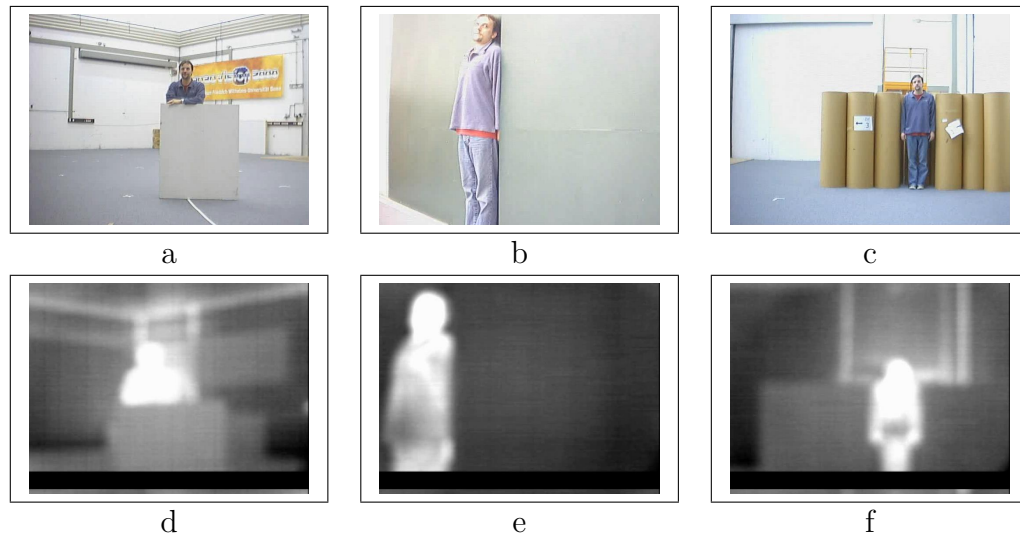


Figure 3.18: More thermographic examples: (a)-(c): Example situations shown by a camera. (d)-(f) The same situations taken from a thermocam.<sup>5</sup>

The weaknesses of the laser scanner, which were discussed in the last subsection, also do not apply for the thermographic camera. The upper three pictures in figure 3.18 present typical situations, where the Laserscanner fails to detect a human. In the first example, the human is partly hidden behind

<sup>4</sup>While this picture is not omnidirectional, omnidirectional cameras are not that difficult to build.

<sup>5</sup> Pictures by courtesy of Bernd Brügemann, FGAN



---

an obstacle, in the second the human pressed himself against a wall, making it very difficult to find him and in the last example, the human shoved part of the environment to make himself indistinguishable from the obstacle behind him. In situation (a) and (c), the laser scanner is naturally unable to find the human, while in the second situation, it is at least very difficult. The lower part of figure 3.18 shows the thermographic images of the same situations. In all three pictures, the human is easily to be found.

While it is also possible to actively avoid detection by a thermographic camera, it is a lot more difficult as in the case of the laser scanner. The disadvantage of these cameras is their high price and high energy consumption, due to the necessity of active cooling. They are also not suited for navigational purposes, since they are unable to detect the geometry of objects of the same temperature (see figure 3.17).



# Chapter 4

## Metrics and Simple Search Methods

In this chapter we will discuss metrics to analyze and compare different algorithms for intruder searching, which we will use in later chapters for the evaluation of the proposed methods. We will also give a number of structurally different maps to show the generality of our approaches.

After giving the metrics, we will also propose some simple search methods to allow a better understanding of the problem and the metrics themselves. This data will also allow us to compare the more sophisticated solutions, which we will present in later chapters, with simple and naive methods.

For the sake of completeness we have included the metric entropy as well as the search method of simple random movement, which both turned out to be unsuitable, as metric or search method respectively.

### 4.1 Metrics

#### 4.1.1 Probability of catching the intruder

One important metric in the context of security systems is to look for the average time the system needs to "catch" an intruder, given one is there.

Based on the models from chapter 3, this can be calculated in a straightforward manner. At each sensor-scan of the system, the probability of an in-

truder (who is correctly represented by the human model) being seen within the sensor can be derived. The opposite event is the probability of an intruder not being seen, which can be joined over time, leading to the formula:

$$P(t) = 1 - \left( \prod_{i \leq t} (1 - p_i) \right) \quad (4.1)$$

where  $p_i$  is the probability of the intruder being seen by the  $i$ -th sensor scan. In the probabilistic context one has to set a limit, when an intruder is considered "caught". This leads to the following term for the time needed:

$$b(t) = \begin{cases} 0 & \text{if } P(t) \geq c \\ 1 & \text{else} \end{cases} \quad (4.2)$$

$$Q = \operatorname{argmin}_t (b(t) + P(t)) \quad (4.3)$$

An Example for this metric can be seen in Figure 4.1. In this example

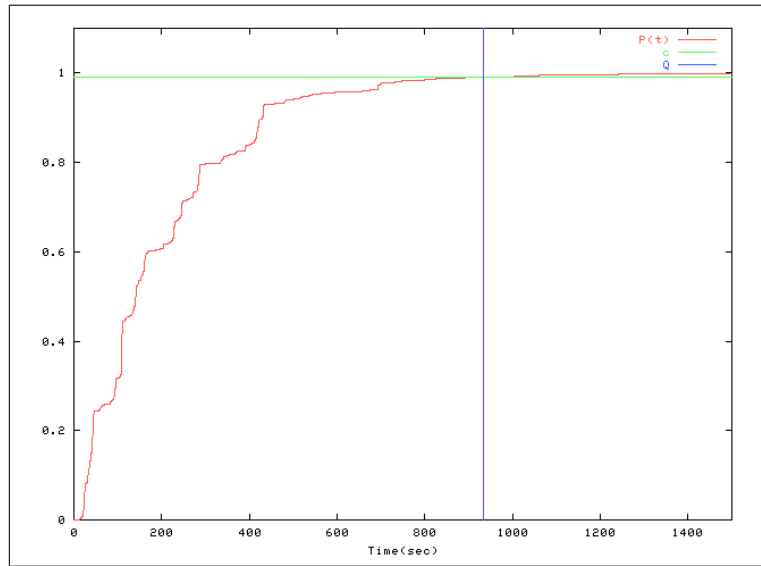


Figure 4.1: Example for "Catch" Probability

$c$  is set to 0.99, leading to a  $Q$  of 933 seconds, which means that in this setting the system needs about 15.5 minutes to "catch" an intruder with a

probability of 99 percent.

### 4.1.2 Contamination

The most common metric found in the context of pursuit evasion games is contamination. Contamination is a binary concept of a place or area, which can either be contaminated or free. Since our work is of probabilistic nature, the term needs clarification. We would like to define a place or area as free, if the possibility that a yet unseen intruder could still be in the place or area is less than  $c$ <sup>1</sup>. If the probability is  $c$  or higher, we define the place or area to be contaminated. The probability of a yet undetected intruder being in a cell is called the contamination level of the cell and has already been briefly discussed in section 1.3.

One can think of two evaluation metrics based upon this concept. The first is the size of the contaminated area in proportion to the complete area to be searched. The second is the time that a system needs to reduce the contaminated area to 0.

### 4.1.3 Entropy

Another metric one might think of is the entropy of the intruder distribution. Naturally, this metric only makes sense for the case of global intruder models, since in local human models, the level of entropy is equivalent to the number of uncontaminated cells. But even in the case of a global model, the metric is only of very limited use, as long as the goal is to see the intruder and not to restrict his presence in a small space.

## 4.2 Experimental Setup

### 4.2.1 Maps

To analyze the proposed algorithms under different boundary conditions, six different maps were used for the experiments, ranging from rather simple

---

<sup>1</sup>The value can be chosen almost arbitrarily, as long as the value is between 0 and 1.

surroundings to complex scenarios taken from real world maps.

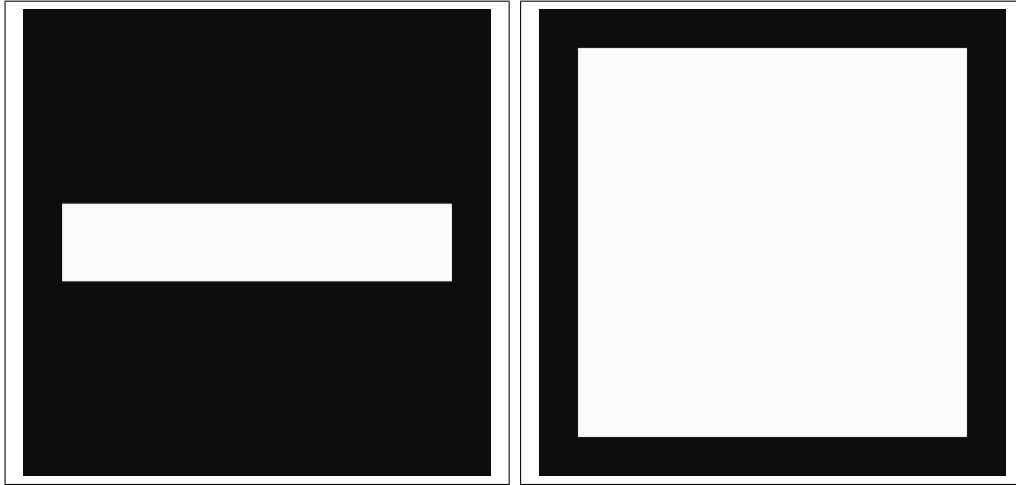


Figure 4.2: Simple Corridor / Empty Hall

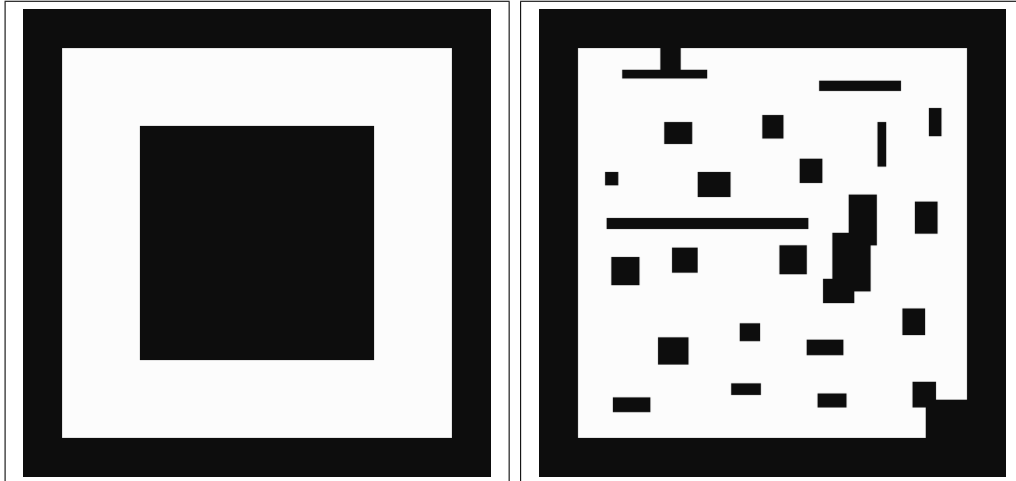


Figure 4.3: Hall with pillar / Cluttered Map

These maps are shown in figure 4.2, 4.3 and 4.4. While the first four are pure simulation maps, the the latter ones are derived from an existing building.

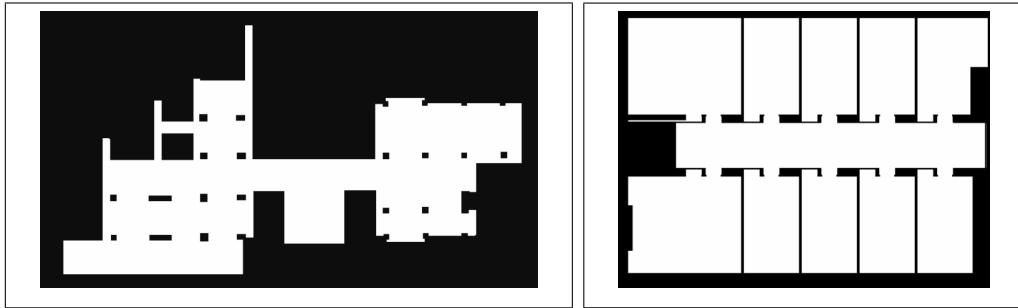


Figure 4.4: University Building Bonn - Ground floor / First Floor

## 4.3 Simple Search Methods

### 4.3.1 Static Observers

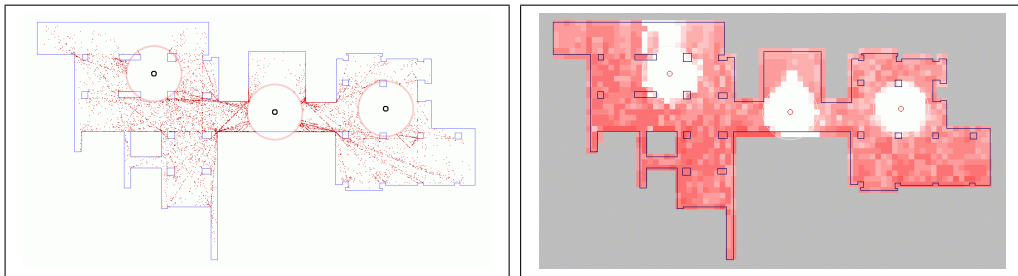


Figure 4.5: Probability density distribution during an experiment with three static observers. Left: Distribution calculated with a particle filter. Right: Distribution calculated with a histogram based filter.

The most simple strategy for the pursuers is to just stay in fixed positions and wait for the intruders to enter their field of view. Many of today's security systems (for example cameras, motion detectors) are based on this scheme. Since there is no movement, there is no need for planning, coordination or control, except for the direction of the sensor in terms of sensors with a triangular field of view, which are beyond the scope of this manuscript. One advantage of this approach is the segmentation of the space for the intruder, which cannot reach certain points in space without entering the sensor range

of a pursuer. The disadvantage is that since there is no active searching an intruder can stay undetected for an arbitrary amount of time. If he is already located in the same segment as his goal location, he can reach it without any danger of being detected. Finding optimal positions for observers is a very difficult problem in itself and its details are beyond the scope of this manuscript. We used a greedy strategy based upon the particle density taken from the model of section 3.1.1, which causes the group to be placed on "choke points" with a high probability of catching unaware intruders. An example of such a group of positions can be seen in figure 4.5, where a group of three robots are set up to guard the map. Evaluating the method for intruders with awareness (limited or perfect) is only of limited use, since no intruder with awareness would enter the FOV of one of the observers and the observers have no means of moving on their own. In this case, the probability to catch an intruder is always zero. The contamination also doesn't change over time and becomes constant after the first time step. So, we limit the evaluation of static observers to the case of unaware intruders. We did a number of experiments with different group sizes and maps and found the systems behavior to always be almost the same, regardless of the used map or proposed intruder speeds. After a short startup phase the system enters a loop with a constant probability of catching an intruder. An example of such an experiment can be seen in figure 4.5. In the left part, we used a particle based model, while the right shows the same positions for a Markov model. Both produce the same results.

### 4.3.2 Random Walking Observers

Given a large area where static sensors are unable to watch the whole area at once, there are always blind spots, in which an intruder could stay for an unlimited amount of time. So, the observers have to move themselves to expand their fields of view and to catch stationary or moving intruders. The easiest strategy for this is to move the observers in a random pattern through the area, so that every point in free space is, at some point in time, in the observers' fields of view. The main advantage of this strategy compared to the static observers is the constraint on the intruder to move, if he wants



to stay undetected. The main disadvantage is the lack of the above mentioned segmentation, giving the intruder the possibility to reach almost any point in space undetected, given the observers take a disadvantageous route at random. Coordination is, as in the case of static observers, very limited and reduced to the necessity of collision avoidance.

Analogous to the discussion in section 3.1.1 the term of random movement needs further specification. In principle, a random strategy could be any (even the most sophisticated) strategy for moving the searchers, which includes some (even insignificant) random element. This is obviously not suitable for any form of comparison and would render the specific term random strategy almost useless. Therefore, we limit our focus at this point to the case of simple random strategies, specifically the strategies already discussed in section 3.1.1 (Brownian motion, simple random walk and random way point). We did a number of tests with different parameter settings and team sizes and evaluated these using the prior discussed metrics.

In case of the contamination metric, results proved all three methods to be unsuitable for cleaning the area. Even a group size of more than twice the minimum necessary number of searchers (see next chapter) was unable to clean out all contamination from the area in our tests, regardless of the chosen random method. However, we have to add that this is naturally not the case for unlimited time, since under unlimited time conditions, the robots' plans would include a perfect decontamination plan just by coincidence. However, in limited time (our simulation experiments usually last 2-3 hours of simulation time), we never found even a very large group to clean out the area.

For the catch-probability the situation is different, since even disadvantageous routes have a probability to catch an intruder. Again, we did a number of experiments with all three methods and compared the resulting catch-probabilities over time. Typical results can be seen in figure 4.6. For this experiment we used the groundfloor map (figure 4.4/Left) with a sensor range of 5 meters. As one can see, due to the random nature of the methods, the performance over time (even when strongly smoothed) shows a great variance. While the overall performance, if measured and averaged over more

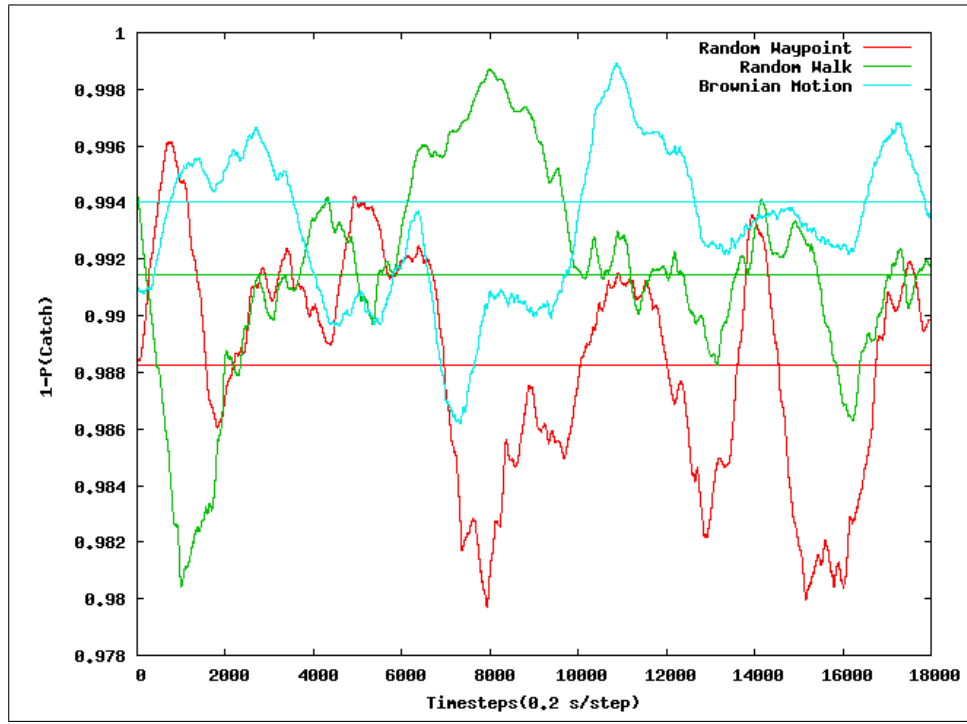


Figure 4.6: Results of different methods for three random walking pursuers. Red line shows the performance of pursuers moving through a random waypoint model, green represents simple random walk and light blue represents pursuers doing Brownian motion. Performance is measured in  $1-p_i$ , so lower values are better. Straight lines represent the average for the method. Shown time frame is one hour.

than one hour, shows the random way point method outperforming Brownian Motion and Simple Random Walk, all three methods show long phases of poor performance. Compared to static methods, only the random way point method was able to perform slightly better and only if measured over a long time.

Following these results, we decided to exclude the random search methods from further examination and comparison. The main reason for this was the limited usability of the methods in a real scenario. We think that one of the prospects of a multi robot security system should be a good average

performance. While we cannot rule out the possibility, that a random system can possibly outperform even a far more sophisticated system, we think the possibility of the system to fail over longer time periods is unacceptable.

### 4.3.3 Fixed Walkarounds

The last method, which concludes the simple search techniques, is the following of a fixed roundtrip. This technique is often found in normal human security applications, where a watchman is sent on an a priori determined path, for an example through a museum. The route of the watchman is often the closest roundtrip, which visits a set of important points. The advantage is that each possible intruder location is checked in a fixed time interval, so fixed intruders or intruders which stay on one location for a long time, are detected. The disadvantage is, that if the tour is known by the intruder (see section 3.1.3), it can be used to avoid detection completely (see figure 3.11 for an example). An instance of the problem, based upon polygonal 2-dimensional maps with a  $\infty$ -searcher (a watchman whose omnidirectional vision is only blocked by obstacles) is known as the optimal-watchman-route (OWR) problem. This problem is proven to be NP-hard for general polygons [CN86].

In our case, the searcher's abilities are additionally reduced through a range limitation, which extends the original OWR problem. The resulting problem is called the d-sweeper problem [Nta92] and is naturally also NP-hard. One approximative solution to the problem is superimposing a regular grid of points over the free space of the map and solving a TSP problem using these points. While this solution works and produces a usable roundtrip, the solution is often unnecessarily long and complex to calculate. Results can be improved by using a grid, which is better customized for the underlying map structure than a regular grid. Such a grid can be calculated by the use of a decomposition algorithm, which we will present in details in the next chapter. At this point it shall suffice to explain that the algorithm tries to find a minimal vertex-covering of the area with the constraint that no point of space is more than a constant distance from a vertex. The resulting amount of points is (depending on the map structure) a lot smaller than the amount

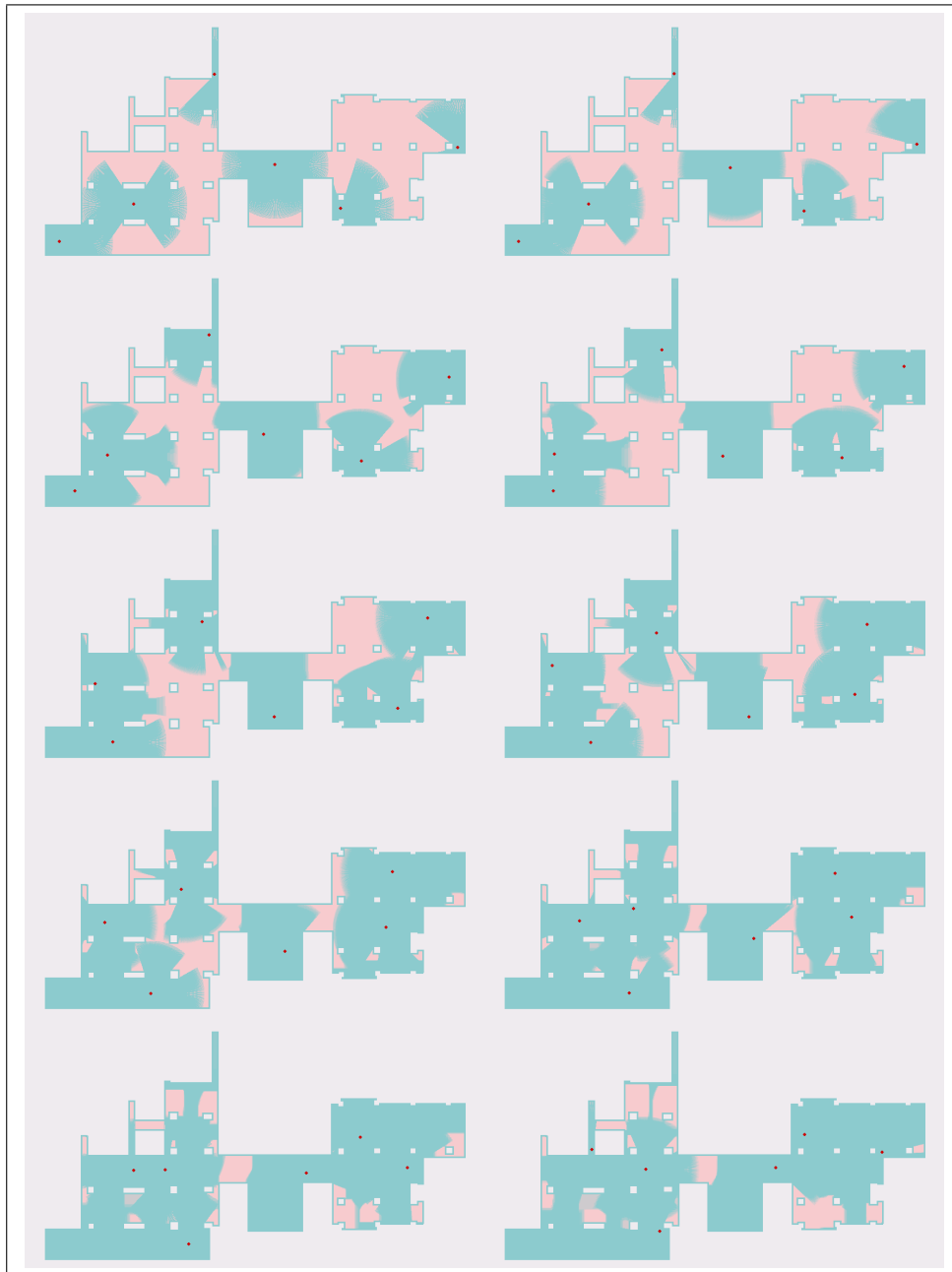


Figure 4.7: Example of a roundtrip experiment with 6 robots. Flow is line by line, every row represents 20 time steps of the simulation system.

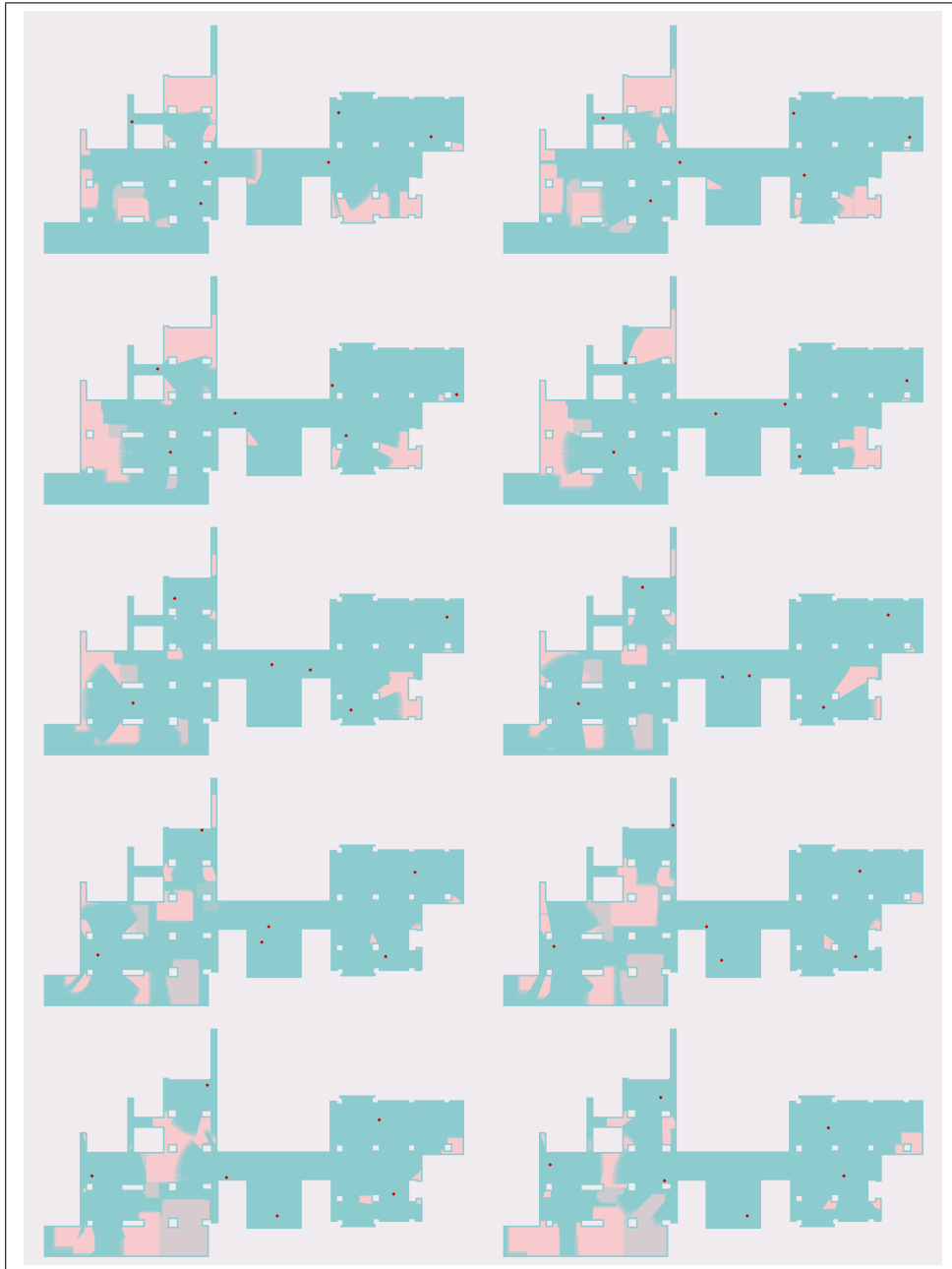


Figure 4.8: Example of a roundtrip experiment with 6 robots. Flow is line by line, every row represents 20 time steps of the simulation system.

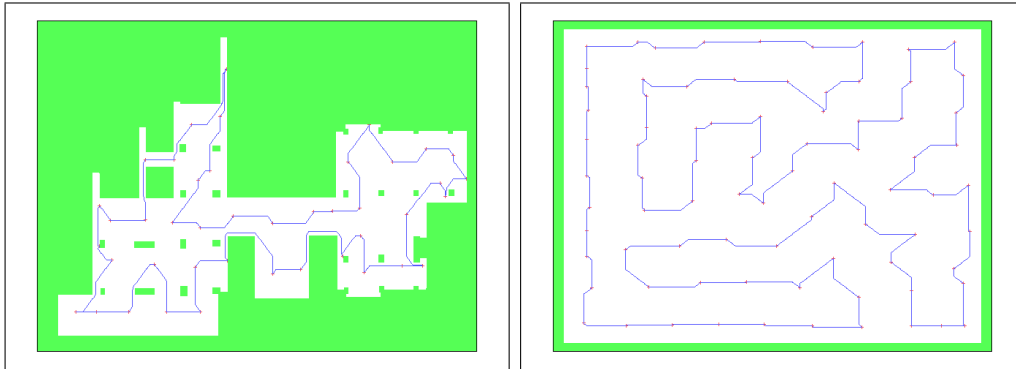


Figure 4.9: 2 Examples for Roundtrips. Left: Ground floor University Building / Right: Empty Map

of a regular grid. While TSP is also NP-hard (even in the strong sense), we used a practical heuristic for solving the problem, which is known to generate almost perfect solutions for small amounts of vertices. [HEL00]. The results for two example maps can be seen in figure 4.9.

### Multiple Robots

While the roundtrip is the usual solution in normal security applications, it only covers the case of one searcher. To extend the solution to a group of robots, two strategies are possible, which result in similar system behavior, but have different levels of complexity. The first is to split the free space (or the grid) prior to solving the TSP in equally sized bins (one bin for each pursuer) and solve the TSP for each bin. In this case each pursuer searches his private area for intruders. The disadvantage of this method is, that the selection of the bins is a very complex problem of its own. The alternative is using the path of the whole space and dividing the searching robots on this path. Since both systems behave similar and the latter is much easier to implement and test, we choose it for the following experiments. As before, we did a number of experiments for different group sizes and different relative speeds of pursuers and intruder.

### Evaluation

Again, we start by applying the contamination metric. An example of such an experiment can be seen in figures 4.7 and 4.8. At the start of the experiment, the 6 robots are divided among the roundtrip path. After that a planning module steers the robots on the roundtrip, while keeping the distance between the robots in a certain interval, to keep the robots separated on the roundtrip. The regions painted in dark blue represent cleared area, while the regions painted in pink are considered contaminated. As one can see, the contaminated regions are small compared to the cleared space, but at no point in time the whole area becomes clear. A clever intruder could evade capture at all times. Testing different relations between the speed of the robots and the intruder, we found that even a speed ratio of 4 to 1 between pursuers and intruder would not suffice to reduce the contamination to 0. We had to increase this ratio to 16 to 1 to achieve that goal. Simply speaking, to guarantee an intruder in this scenario to be found with only 6 robots, we would have to build robots, which could move 16 times faster than humans and still offer reliable navigation.

To evaluate the technique with the probability to catch metric, we also did a number of experiments with several team sizes and different ratios between intruder and pursuers' speed. In contrast to the randomized methods the variance of the values is very low, so the values are much better suited for comparison with other techniques. As one would expect, the size of the team turned out to be directly proportional to the probability to catch an intruder (at least for reasonable small team sizes), so in case of intruder searching by applying a roundtrip, the catch probability can be doubled by doubling the number of pursuers. Examining different speed ratios lead to more interesting (and at firsthand unexpected) results, which are plotted in figure 4.10. In this case, we did a roundtrip with three robots traveling about 30-40 cm/s, which is a realistic speed for navigation with a B21 robot. We examined catch-probabilities for intruders traveling slower than the robots, intruders with the same speed and intruders moving faster than the robots. We expected the probability to catch to be lowest with the slow moving intruders and to be highest with the fast moving intruders (analogously to the static pursuers). Interestingly, the lowest probability to catch was calculated

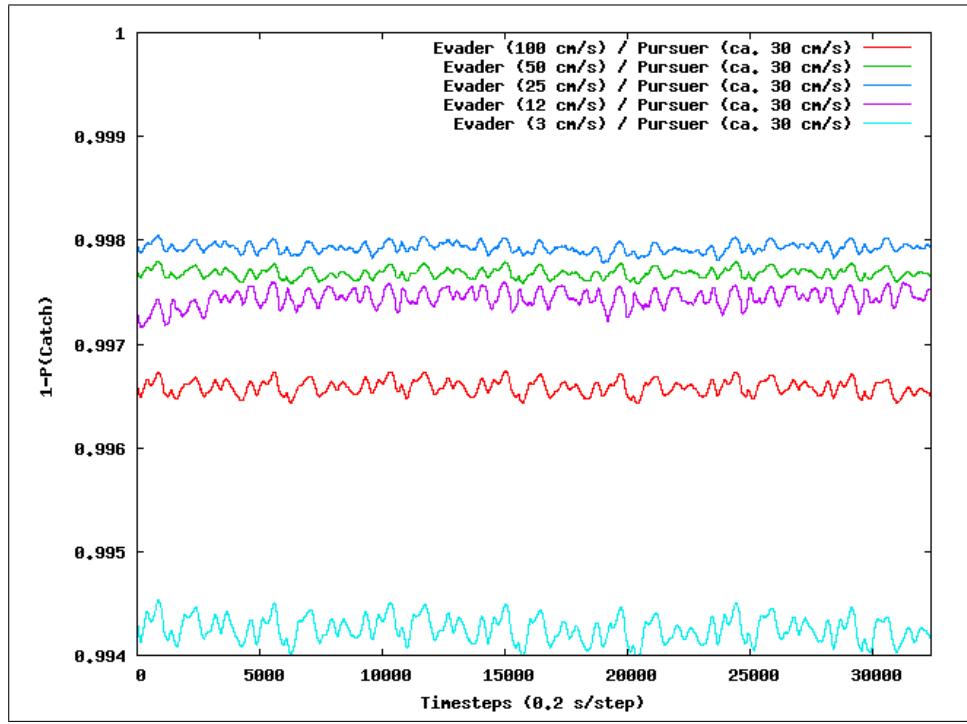


Figure 4.10: Performance of three pursuers on a roundtrip with different speed settings for the intruder. Red line shows a fast intruder (100 cm/s), green a moderate (50 cm/s), blue a slow (25 cm/s), violet a very slow (12 cm/s) and turquoise an almost standing intruder (3 cm/s). Performance is measured in  $1-p_i$ , so lower values are better. The system performance is worst, if the intruder speed is almost the same as the system speed.

for the intruders moving at the same speed as the robots, while slower and faster intruders have a higher probability to be caught. While this seems to be unexpected at first, the explanation is straightforward: In this case, the pursuers show the same "flow" as the probability distribution of the intruder. If a pursuer moves faster than the intruder, every move of the pursuer shifts some part of the distribution in its sensor radius, if the intruder is faster than the pursuer, every move of the intruder moves some part of the distribution in the sensor radius of the robot from the backside. In case of similar speeds, both effects are at a minimum and the sensor lies almost all time in a gap



of the distribution. From this observation one can conclude, that in case of an intruder unaware of pursuit, a system should avoid choosing the speed of the intruder for its pursuers.

Apart from this observation, the results for speeds faster than the pursuers (in our case, the more realistic speeds for real human behavior) show an increase in the catch probability with increased intruder speed, however, it should be noted, that the speedup is sub-linear.



# Chapter 5

## Contamination Based Searching

In the last chapter, geometric information was used to calculate the shortest round trip. This path does not take any sensor model or human model into account, especially not the possibility of the intruder to move. In this chapter, we want to use these models and generate plans and actions based on them. We especially want to generate plans, which "guarantee"<sup>1</sup> the intruder to be found after plan execution. Parts of this work have already been published, for reference see [MRS05].

### 5.1 Space Decomposition

To formulate planning problems, one has to specify the planning space. The canonical planning space in the context of grid maps is to restrict the allowed positions for pursuers to grid cells, therefore avoiding the problem of dealing with continuous and infinite planning spaces. To account for the intruder, the state space is the Cartesian product from these positions and the possible distributions of the intruder. So, the state space for a planning problem with  $n$  pursuers and one intruder would be:

---

<sup>1</sup>Guarantee is here not meant in its exact verbatim meaning, since in a probabilistic context the probability of 1 is never reached. Instead we simply mean a very high probability to catch the intruder like 0.99 for example.

$$State_t = (P_{1t} \times P_{2t} \times \dots \times P_{nt}) \times Bel(D_{It}).$$

An action in this space would be a movement of a pursuer to a neighboring cell ( $P_{rt+1} = P_{rt} + \delta_{rt}$ ). This state space is far too big even for very short time intervals and small maps, rendering planning based on this intractable. Therefore, we have to reduce the planning space significantly, which we will do by reducing it to a graph and generating plans for pursuers moving on this graph.

### 5.1.1 General outline

The basic idea is to project the problem from a general 2-dimensional search problem to a graph search problem, which itself is solved by an A\* planner. Therefore, we construct a set of vertices, so that each point of free space (every point where an intruder could be) is in sensor range of at least one vertex. The vertices themselves are taken from the configuration space of the pursuers. In other words, the whole space would be covered by sensors if we could place a pursuer on every vertex. Such a set can always be found as long as there are no points in the intruder's configuration space which cannot be seen from the pursuers' configuration space.

After adding edges between the vertices (section 5.1.3) we have constructed a connected, undirected graph, which is a subset of the original problem but with a much smaller complexity. While still being difficult to solve, searching this subset can successfully be done by a smart A\* planner, as long as the original problem was already solvable. It is important to note that this would also be a solution for the general 2-dimensional problem.

### 5.1.2 Computing the vertices

We first have to determine a set of vertices, which satisfy the constraint that every point of the free space  $\mathcal{F}$  of the environment is in the field of view of at least one vertex. Naturally, we want to cover the complete free space with a small number of vertices. Already for the special case of infinite sensor range, minimizing this quantity is an instance of the well known art gallery problem,

which is known to be NP-Complete [O'R87]. Instead of computing an optimal solution, we employ a randomized optimization algorithm, which starts with an empty vertex set and expands this set with vertices, which overlook space not yet covered by the vertices of the current set. Due to the nature of this randomized approach this often leads to unnecessary big vertex sets, with unnecessary redundancy. To cope with this problem we implemented a number of optimizations, which lead to good results in practice:

- We prefer to select vertices which overlook greater parts of the environment. This can be done by randomly selecting a large number of candidates and selecting the vertex which overlooks the largest part.
- After the whole environment can be seen from the vertices, all unnecessary (redundant) vertices are removed.
- During the selection all vertices are joggled to improve their distribution over the environment (hill climbing).
- The whole process is completely restarted with an empty set, as long as no set with a sufficiently small number of vertices is found at the end.

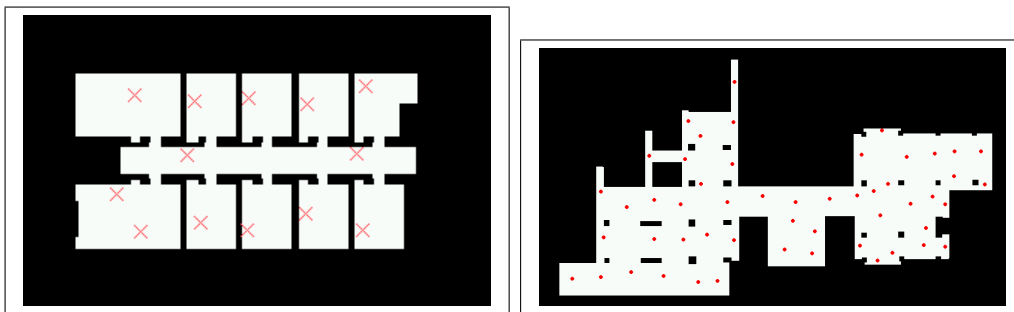


Figure 5.1: Example Vertex Sets

Two examples of such vertex sets are depicted in Figure 5.1. Each point in space that can be seen from at least one vertex is associated with the nearest visible vertex based on euclidian distance. The space associated to a vertex

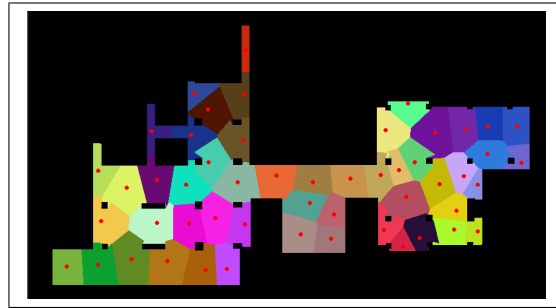


Figure 5.2: Example decomposition

is called the region of the vertex. An example for such a decomposition can be seen in figure 5.2. It should be noted that the regions are not solely generated based on euclidian distance but can also depend on visibility. For example look at the hatched region in figure 5.3, which belongs to the region of vertex A, even if the euclidian distance to vertex  $c$  is smaller.

### 5.1.3 Computing the edges

Now we have to build a graph based on the vertex set. For this purpose, we connect two vertices by an edge, if their regions are adjacent to each other (see figure 5.3). Sometimes borders between two regions are interrupted by occlusion. In this case, we add one edge for each continuous part of the border, so that the corresponding vertices are connected by multiple edges (This is illustrated in figure 5.3 with the dashed line in the lower part of the drawing. The vertices A and B are connected by two edges.). The resulting graph itself is undirected and connected. Later we will also need a cost function for the edges, so we define the costs as the time that one of the robots would need to traverse the edge from one vertex to the other. These costs are calculated by a MDP planner [TBB<sup>+</sup>98] based on the geometric map of the environment. The paths computed this way are also used for navigation. Note that two edges connecting the same two vertices can have different costs, because the path of the robot has to cross the part of the border corresponding to the edge.

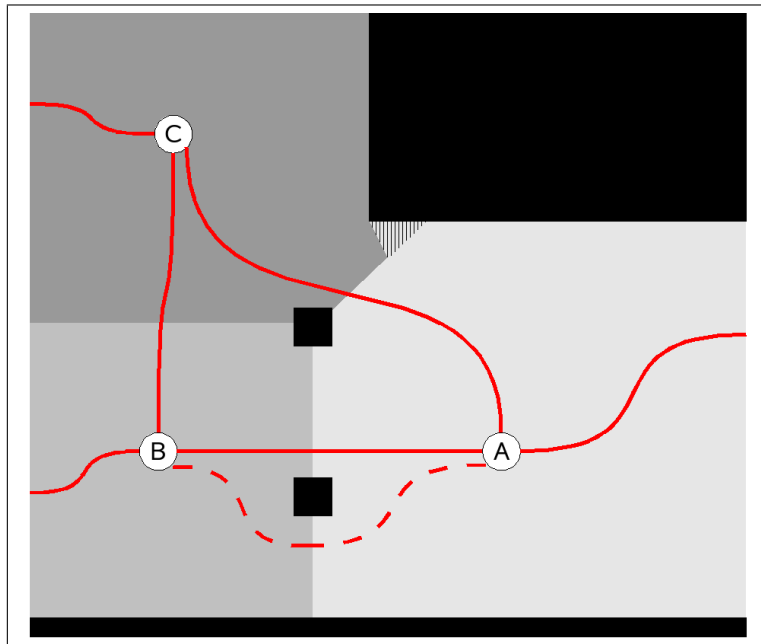


Figure 5.3: Three vertices and the corresponding edges. Obstacles are black.

#### 5.1.4 Final Graph

The result of adding the edges is the travel graph, an example can be seen in figure 5.4. In the left part one can see the projected paths the robots have to take when traveling from one node to the next. The right part of the figure shows the resulting final graph.

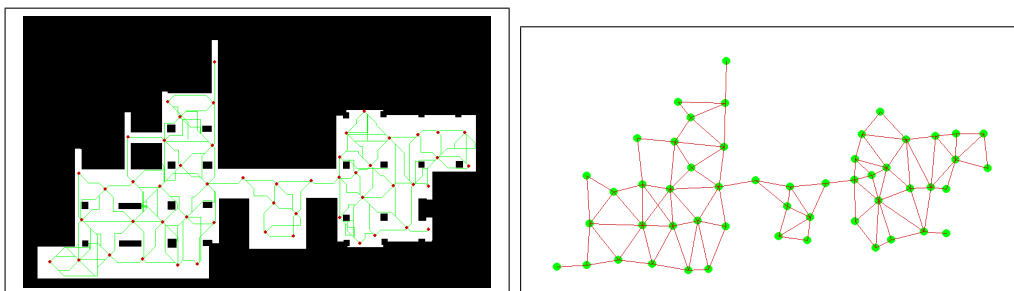


Figure 5.4: Graph with resulting edges / Final Graph

## 5.2 Graph Decomposition

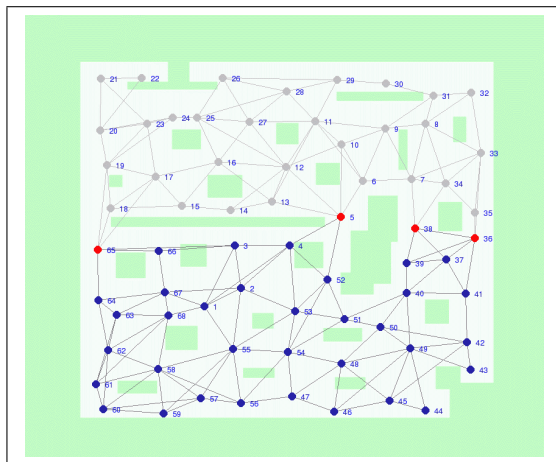


Figure 5.5: Example Split

Although this procedure reduces the complexity of the search space significantly, large or complex areas still lead to graphs with many vertices, so planning in such a graph may still be computationally infeasible in practice. For example, consider a search graph of 30 vertices and a group of three robots. Let this graph be simple, so that no node has a degree of more than two. Even if we assume an almost optimal level of parallelism, the naive search tree will have at least a depth of 10 and a branching factor of 8 (every robot has in any step the choice of between his two neighbors), resulting in a size of almost  $8^{10}$  nodes. Finding solutions for maps with 100 or even 200 vertices would be unacceptable.

However, large real-world environments usually consist of multiple subdivisions whose interconnections are relatively simple and easy to guard, e.g. rooms in a building connected by a few doors and hallways. A reasonable attempt to solve such a planning problem would be to search all rooms independently and consecutively, and place guards to prevent an intruder from reentering already cleared areas.

Based on this we solve large problem instances by a divide and conquer approach, that splits the graph into subgraphs. These subgraphs can be



searched independently, sacrificing a few robots to guard the borders between cleared and contaminated areas and gaining a huge reduction of search complexity in turn. An example of such a split can be seen in figure 5.5. (Note that this approach works even if the above mentioned intuitive assumption above about the topology of the environment does not hold. Then, in order to find the intruder, the robots have to form some sort of sweep line that sweeps across the search space.) Thus, a split through the graph can be considered as an intermediary search state dividing the graph into a cleared and a contaminated part with a sweep line in between, and independent planning is equivalent to pruning the search tree upon reaching the split state. The

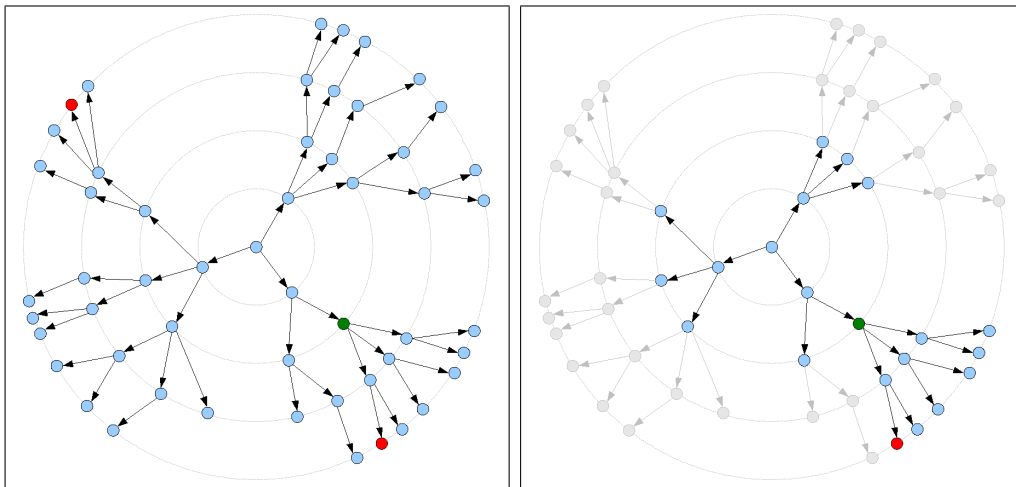


Figure 5.6: Complexity reduction through the determination of an intermediate state: Left figure shows the normal search tree. Blue nodes represent search nodes, red nodes represent solutions, green is an intermediate state. Right tree shows the reduced search tree, which results from first searching from the start to the intermediate state and then searching from the intermediate state to a solution.

principle is shown in figure 5.6. If we can identify an intermediate state/split, we can avoid doing an expensive, complete search (shown left), but instead conduct two cheaper searches (shown right).

Naturally, the number of possible splits in a given search problem is quite large. However, we found two attributes of splits, which strongly affect their

usefulness. Firstly, the best speedup for the search can be achieved by a split, which lies exactly in the middle between the start and the solution. Therefore, a split should divide the graph in roughly two parts of the same size. Secondly, a split should be as general as possible, meaning that a split with less nodes is better than a split with more nodes. Since both requirements can oppose each other, a useful split has to balance between both requirements.

The algorithm used to create such a split works as follows: First, all possible border lines in the graph are generated and the resulting connected components (or subgraphs) are calculated. Then, these split candidates  $\mathcal{S} = \{S_1, \dots, S_n\}$  are rated using the heuristic function

$$f(S) = c \cdot l + \sum_{i=1}^k \left| n_i - \frac{N}{2} \right|,$$

where  $l$  denotes the length of the border, i.e. the number of vertices which interconnect the subgraphs,  $k$  the number of subgraphs,  $n_i$  the sizes of the subgraphs and  $N$  the size of the complete graph. The balance between the requirement of a minimal sized split and a split which lies at most in the middle between start and solution is done by the parameter  $c$ . We found that setting the parameter to  $\frac{5}{2}$  lead to the best results in the typical graphs we used. The best candidate is determined by

$$S_{\text{best}} = \operatorname{argmin}_{S \in \mathcal{S}} [f(S)].$$

Splits that evaluate above the threshold  $\frac{3}{2}N$  are ignored to prevent degeneration phenomena like single vertices being cut off the rest of the graph.

Even if no separation points exist, the splitting still improves search performance as it allows to discard portions of the search tree upon reaching a split line as an intermediary goal. Thus it allows to solve larger problems.

## 5.3 A\* Planning

The planning process itself is carried out using the A\* algorithm. The state of a planning node consists of the vector of robots positions (a robot can be on a node or an edge), the contamination state of all vertices, and time.<sup>2</sup> So the planning space is:

$$State_t = (P_{1t} \times P_{2t} \times \dots \times P_{nt}) \times Bel(D_{It}) \times T_t.$$

The starting node of the planner uses a vector of initial robot positions, which can be chosen arbitrarily, and the goal state is a vector of desired robot positions and a completely cleared graph. An action is either

- Choosing a new target for a robot that stands on a vertex
- Choosing for a robot to wait for a few seconds

Whenever an action is assigned to every robot (moving or waiting), time is forwarded until a new decision is necessary. During each of these phases, the contamination is calculated. A vertex is contaminated, if there is no robot standing on it and there is an edge with no robot on it leading to a contaminated vertex. It is important to note that vertices which are cut off by splits can also contaminate vertices in the local planning space, if they are not already cleared. Remember that we needed to add additional edges between two vertices, if the border between their regions is not continuous (due to occlusion). If one robot travels from one vertex to another one, the other edge could still contaminate the starting vertex, if the target vertex was already contaminated. In such a situation a second robot is always necessary to guard the starting point.

In the next subsections, we describe three optimizations to speed up the whole process.

### 5.3.1 Recontamination

Lapaugh showed in [LaP93] that in graph searching, recontamination is not necessary to find a solution. This result is based on a slightly different model,

---

<sup>2</sup>We will discuss alternative planning spaces later.

whose goal is set to clearing edges instead of nodes and which also allows a pursuer to jump from one node to another, non adjacent node.

The result cannot be directly transferred to our formulation of the problem, since we can construct graphs, where a jump from one node to a non adjacent node is necessary to keep the number of searchers minimal (see section 5.8.3 for an example). However, we found that the assumption of recontamination being unnecessary does hold for almost all practical cases. Almost always when we were able to find a complete decontamination plan for a number of robots, we also found a decontamination plan for the same number of robots, which does not incorporate any recontamination. So we usually skip all plans, in which vertices are recontaminated. This leads to a tremendous speed up in searching bigger graphs.

### 5.3.2 Heuristics

The optimality of  $A^*$  requires a heuristic function to be admissible, which requires the function to never overestimate the costs for reaching the goal. While it is not difficult to find heuristics, which are trivially admissible (the 0 function for example), we were not able to come up with a heuristic which allowed efficient searching, while still being admissible. However, we came up with a solution, which was mostly admissible, but allowed efficient and goal-directed searching. It shall be noted, that using a non-admissible heuristic causes the  $A^*$  algorithm to lose its optimality but not its completeness.

Our search is directed by a combination of graph clearance achieved and the distance to the next uncleared node. The heuristic function for a given search state  $s$  is

$$h(s) = \sum_{i=1}^R \min_{n \in \mathbf{C}} [d(n, p_i)] + \frac{|\mathbf{C}|}{R} \min_{n \neq n' \in \mathbf{N}} [d(n, n')],$$

where  $R$  is the number of robots,  $\mathbf{N}$  the set of graph nodes,  $d(\cdot, \cdot)$  the distance measure,  $p_i$  the current position of the robot  $i$ , and  $\mathbf{C} \subset \mathbf{N}$  the set of nodes that are currently contaminated. This function prefers search states, in which the robots are moving towards the border line between cleared and contaminated nodes and which subsequently expands the cleared area. Both

behaviors are essential for fast graph clearance and keep the search state expansion overhead low, especially in combination with the recontamination assumption and the hashing of expanded states, that is explained next.

### 5.3.3 State Hashing

Another optimization, which is necessary especially in the context of multi-robot planning is the detection of states, which are equal up to a permutation of the robots. For example consider a planning node, in which robot 1 decided to move to point A and robot 2 decided to move to point B. When expanding the starting node, the planner would also produce a planning node with robot 1 moving to point B and robot 2 moving to point A. Since we assume that all robots have equal capabilities, both of these planning nodes refer to exactly the same state. If a solution for the problem exists, it can be found either way, so one node can be omitted. To do so, we use a hash function, which takes the contamination and robot state into account. If the hash value of a just expanded planning node collides with an older one, it is only expanded further if both states are really different.

## 5.4 Merging of plans

The last part of the algorithm is merging the plans for the subgraphs (which were formed by splits) to a full plan. Since a plan of a subgraph always ends with robots placed on all vertices of a split, the rest of the robots can move freely. The next subset always starts with all robots on the split, so a small intermediate "plan" has to be generated, which moves the free robots to the starting position of the plan for the next subgraph. Because the robots already on the split are not moving, no recontamination can happen making these small plans very easy to generate.

## 5.5 The complete contamination based planner

To summarize, the whole coordination process is carried in the following 6 steps:

1. Generate a set of vertices  $V$ , so that every point in free space is visible from at least one vertex.
2. Add edges to the graph, so that every distinct border between two adjacent regions corresponds to one edge.
3. Divide the graph into subsets, until the size of the subsets is below a threshold.
4. Define for every subset a starting state and a finishing state.
5. Use the A\* planner to find a solution for clearing each subset, beginning with the starting and ending with the finishing state.
6. Merge all subset plans to one full plan.

## 5.6 Complexity and Scalability

### 5.6.1 Planning Complexity

Determining the complexity of an A\* implementation is often very difficult. While it can be proven, that A\* is optimally efficient in the absence of other information than the given heuristic, the complexity is still dependent on the underlying problem and the heuristic itself. For example, A\* can be used to find a solution for the traveling salesperson problem (TSP) and since TSP is known to be NP-complete and the question if  $P=NP$  is still not solved yet, it is at this moment impossible to give precise complexity bounds for such an implementation of A\* (at least without solving the  $P=NP$  problem

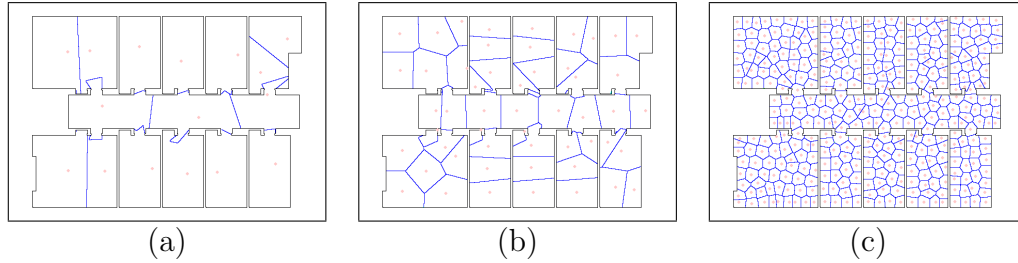


Figure 5.7: Decomposition of a floor map with different scan ranges: (a) Scan Range 1000cm (b) Scan Range 500 cm (c) Scan Range 150 cm

simultaneously). As stated in section 2.1.2 the problem of determining the number of searchers to search a given graph is also NP-complete [MHG<sup>+</sup>88, LaP93] and it is easy to see, that the problem of finding a concrete plan to search a graph is at least as difficult as determining the necessary number of searchers.

However, even if we cannot prove a precise upper bound, we can still provide worst-case estimates of the algorithm. We assume therefore that in the worst of cases the heuristic does not help the search at all, therefore degenerating the A\* search to simple Uniform-cost search (UCS). The time and space complexity of UCS is known to be  $O(b^{1+\lceil C^*/\epsilon \rceil})$ , where  $b$  is the effective branching factor,  $C^*$  is the depth of a solution and  $\epsilon$  is the lowest possible cost between two actions [RN95]. When assuming that no action takes less than one second and that  $C^*$  is much larger than 1, this can be roughly simplified to  $O(b^{C^*})$

The branching factor itself depends on possible actions in a given situation. Therefore it is the product between the number of robots  $R$  and the possible actions of each robot. The number of possible actions is the average branching factor of the underlying graph. This is also a worst case estimate since the product of  $n$  numbers with a fixed average of  $x$  is always maximum if all numbers are set to  $x$  (see Appendix / Theorem 1 for a detailed explanation).

$$b = R \cdot b_G$$

Based on this we can give our complexity estimate as:

$$f \in O((R \cdot b_G)^{C^*+1}) \quad (5.1)$$

Unfortunately this result has more theoretical than practical value. In a realistic scenario, one often deals with 5-6 robots, a graph branching factor of 2-3 and costs of around 1000 seconds. This would lead to a worst case planning estimation of at least  $10^{1000}$  planning steps. However, in practice the algorithm often found a solution with between 10,000 - 1,000,000 search nodes with the use of the above mentioned heuristics.

## 5.7 Alternative Planning Space

Based upon the observations made in the last section and also from some test runs with the algorithm, we got to the idea for a smaller planning space with similar behavior. Our idea was based on three observations:

- The planning space is often unnecessarily complicated by the number of robots. Since the number of robots directly influences the branching factor of the search algorithm, a high number makes the search unnecessary difficult.
- In every step of the algorithm, the graph is always partitioned into two sections, a contaminated section and a free one <sup>3</sup>. Both are separated by robots standing on a node or traveling an edge. The algorithm expands some sort of wave front over the contaminated nodes (see section 5.9.1 for examples).
- When we move a single robot at a time, we can omit the concept of travel time for an edge. If a complete plan exists that clears the graph with parallel movement of the pursuers, there also exists one which cleans the graph, where only one robot moves at a time.

So, instead of planning movements for actual robots, we would like to plan a wave front to cross the graph. A wave front is a vector of graph-nodes,

---

<sup>3</sup>The sections don't have to be necessarily connected.



which separate the cleared space from the contaminated (for initialization the starting node is considered a wave front node at the beginning and is itself considered cleared). An action is either the movement of one such wave front node to a new (contaminated) node without the opening of a rift in the front, the emerging of a new wave front node from an old one (therefore expanding the width of the front) or the deletion of an old and obsolete node (contracting the front).

We also use an A\* implementation to do the search but with different functions for the path cost function  $g$  and the heuristic function  $h$ . We set  $g$  as the width of the wave front, which assures that solutions, which require a smaller front are evaluated first. The calculation of a good heuristic function  $h$  is easy, since we know fairly precisely the number of movements to the goal (since 2 of 3 possible actions clear exactly one contaminated node). To combine both function, we also have to make  $g$  "outweigh"  $h$ , so that regardless of the distance to the goal, solutions with smaller fronts are preferred. This can be done by dividing  $h$  by the number of nodes. So, we calculate  $f$  as:

$$f = \underbrace{Front\_Width}_g + \frac{Goal\_Distance}{\underbrace{|G|}_h} \quad (5.2)$$

This approach offers two advantages: First, this approach gives an optimal solution in respect to the number of robots needed. While the first approach could either give a plan for a given number of robots or fail, this approach always leads to a solution and in the process determines the minimal number of robots necessary. Second, the worst case complexity is much lower than that of the first approach. The planning depth is no longer a function of seconds (please note that the following  $C_{2*}$  is different from the  $C_*$  above), but of planning steps and the branching factor of the plan is determined by the minimum number of robots. Thus,

$$f \in O((R_{min} \cdot b_G)^{C_{2*}+1}). \quad (5.3)$$

However, while this approach is minimal in the number of robots, it is also fixed on this minimum, it cannot make use of any extra robots to speed

up the search. Another disadvantage is that while a successful run of the planning algorithm guarantees a solution, the concrete solution in form of a plan for actual robots still has to be determined. While this is not difficult, it adds another necessary step to the system. Finally, the biggest strength of the second planning space, that it optimizes planning steps instead of time, is also its biggest weakness, since it does not take actual travel times into account and can therefore generate plans, which take unnecessarily long to execute. It depends on the map as well as the size of the group to determine which planning space one should use. Roughly speaking, one should always try for the first planning space, which generates better and faster plans and should reserve the latter for very complex cases, where the first one fails.

## 5.8 Implementation Details

### 5.8.1 Nondeterministic Movement

One of the major problems in the field of robot motion planning is imprecision of physical hardware. Also, the localization of a robot during the execution of the plan is always only an estimate of the real position. Therefore, one cannot expect a robot to follow a given plan from cell to cell as planned. In the above discussed algorithm the property of letting no intruder "slip" past the searching robots (and therefore for its correctness) is dependent on two factors:

- A robot standing on a vertex in the graph can see the full region of the vertex.
- A robot traveling on an edge from one vertex to another, can always maintain visibility of the full frontier it crosses.

The problem of these claims can be seen in figure 5.8. In the left part one can see a (deterministic) plan for cleaning the hallway. If the robot could travel exactly as planned, no intruder can cross the hallway in the opposite direction without entering the field of view of the robot. In a real robot system a robot would more likely travel the trajectory of the right part of figure 5.8. We can expect the system to roughly follow the plan but not

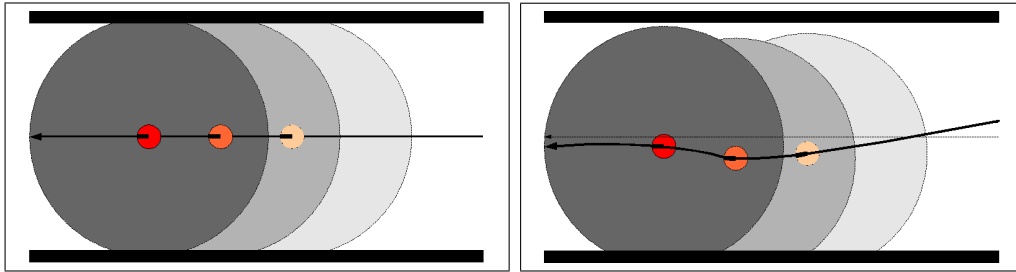


Figure 5.8: Robot moving in a floor from right to left. (a) Deterministic Motion: No Intruder can pass the robot. (b) Nondeterministic Motion: An Intruder could pass the robot without being detected (by sneaking along the upper wall).

precisely as planned. The planner has therefore to take this nondeterminism into account. This can be done by subtracting the maximal possible deviation, which depends on the implementation of the collision avoidance or low level planner and the actual robot, from the effective scanning radius. This deviation has to be estimated in advance.

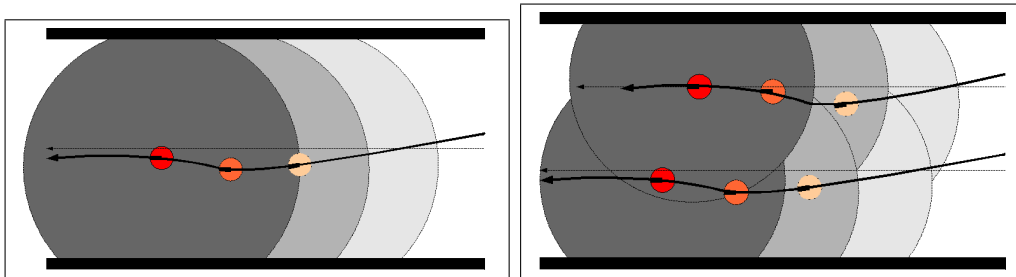


Figure 5.9: Robot(s) moving in a floor from right to left. (a) A robot with a larger scan radius prevents any intruder from passing the robot without being detected. (b) Two robots with each a smaller scanning radius also prevent any intruder from passing the robots.

In the example of figure 5.8 the deviation of the robot's path to the planned path is about 10 percent of the scanning range. The solution can be to improve the effective scanning range of the robot to compensate for the motion (see left part of figure 5.9) or to use a smaller effective scanning range

	0 sec	30 sec	60 sec	90 sec	120 sec
Robot 1	A	B	C	D	E
Robot 2	F	G	H	I	J

Table 5.1: Example Plan

for the planner, resulting in a plan, in which two robots have to cross the floor, to prevent an intruder from passing through (see right part of figure 5.9).

### 5.8.2 Time Delays

Another common problem, while using nondeterministic robots, is that the time a robot needs to cross an edge is only a rough estimate. A robot may be faster or slower than planned. The first case is easy to handle, if a robot arrives early at a node, it can simply wait for its next command, the plan is still correct. The latter case is far more difficult, since a delayed arrival can open gaps in the wave front. An example of such a situation can be seen in figure 5.10, whereas the corresponding plan is seen in table 5.1. Robot 1 has the plan to start at vertex A, crossing B, C, D and finally reaching point E. Robot 2 starts at F and wants to go to J, while traversing G,H and I in the process. Each edge is planned to take 30 seconds.

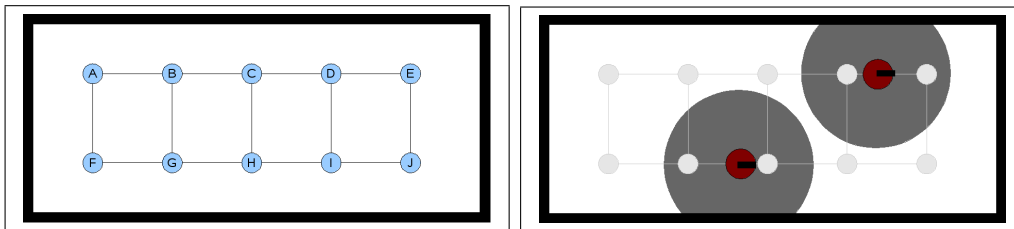


Figure 5.10: Example Situation for a two robot sweep with one robot delayed

In a deterministic scenario the graph would be cleared after the run. But if robot 2 gets delayed for some reason (for example through a wrong sensor echo, forcing him to brake in an effort to avoid a collision) and robot 1 leaves

point D before robot 2 has reached H, a gap is opened, allowing an intruder to slip through. The plan is no longer correct. (In our early experiments, we found this problem to occur in almost every instance, especially in large graphs with more than 300 nodes.)

We found two solutions for the problem: The first one is to overestimate the costs of traversing an edge in such a way, that the problem doesn't occur at all. While this works and shows correct behavior, the running time of the experiment becomes unnecessary large. In our current system, we found situations, where a robot needs about three times the normal time to traverse an edge if everything goes wrong. So, to still guarantee a correct plan, the time for the experiment would also be tripled, most of the time, the system would stand still. This is clearly not acceptable. A better solution is to stop the system only if necessary. So, the execution layer of the system has to check if a robot gets delayed and forces the other robots to wait, until the robot has made up its delay. This works quite well and although delays happen in almost every experiment, they are still a rare occurrence compared to the un-delayed tours. In practice the additional time needed to cope with delays is less than 2 percent of the planned time.

### 5.8.3 Choosing the starting nodes

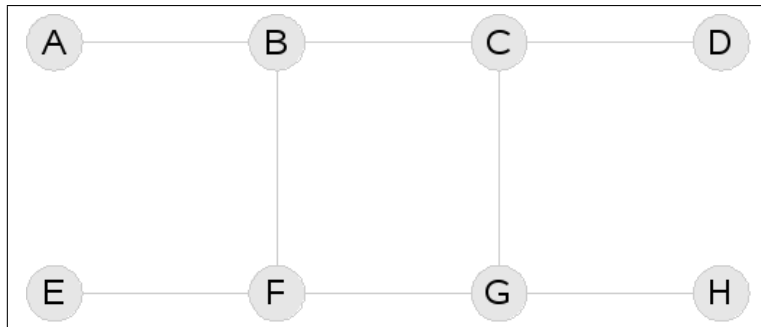


Figure 5.11: Example search graph

In subsection 5.3.1 we assumed that the theoretical result from Lapaugh [LaP93], that recontamination is not necessary to search a graph, does also hold for

our formulation of the problem. This idea leads to a tremendous speedup of the search, larger problems ( $>50$  vertices) are often intractable without this assumption.

Consider the graph in figure 5.11. The search number of this graph is 2, one possible solution is to start with robot #1 at Node A and robot #2 at Node E, and let robot #1 move from A to D, while robot #2 moves from E to H. After three moves of each robot the graph is cleared. So there exists a plan for two robots which clears the graph without recontamination. However, there is no plan without recontamination, that lets both robots start at the same location and does not incorporate the jumping of a robot to a non adjacent node. So this graph is an example, where the assumption does not hold. Based on this, there are three alternatives to cope with the problem:

- We could drop the prohibition of recontamination. As we have stated before, the prohibition is needed to terminate the search in reasonable time. So, while still remaining correct and complete, the search would become impractical.
- We could allow the robots to jump to non adjacent nodes. This would greatly increase the branching factor of the search and make the method completely impractical for any non-trivial search space.
- We could guess a good startup configuration with a high probability that a plan without recontamination exists from this startup configuration. We would therefore sacrifice the optimality of the resulting plan in terms of minimal number of robots needed.

From these three alternatives, only the third proves to be practical. From our experiments, we learned that a good startup configuration is often placing all robots on one node at the outer boundary of the graph. In the above example, this leads to a plan with three pursuers, which is clearly suboptimal, since a plan with only two robots was shown to exist. In practice, we found that this problem becomes more and more negligible with larger robot groups, however choosing a better startup configuration can, dependent on the structure of the graph, sometimes save one or two robots.

## 5.9 Experiments

### 5.9.1 Time based planning

We conducted numerous experiments with the described system using different maps, group sizes and parameter settings like intruder or pursuer speed. To make things comparable, we used the same simulation system as described in section 4.2. While naturally showing map-specific variations, all experiments lead to very similar results, so we would like to confine the discussion to two representative examples.

For the first example, we used our common ground floor map (figure 4.4), since we believe it to be a good example for real applications. We have already shown an example graph resulting from the decomposition of the map (see figure 5.4). We choose the middle of the map as the starting point and choose a group size of five pursuers.

The resulting experiment can be seen in figures 5.12 and 5.13. The figures show the experiment line by line, starting from the starting situation in the upper left to the end situation in the lower right. As one can see, the robots start decontaminating the right part first, while a single robot is set to guard the corridor in the middle. After the right part is cleared the robots return to the middle and go for the left part, while still guarding the corridor leading to the already cleared right part. In the whole experiment, the only observable recontamination is due to the discretization of the movement.

For the second example, we choose a slightly more artificial but more challenging map, which is an empty, quadratic hall of  $50 \times 50$  meters (see figure 4.2). The decomposition of this map leads to the graph in figure 5.14. While the decomposition is comparably easy, the resulting graph is one of the most difficult to plan on, because of its high connectivity. We use a team of 8 robots and let them all start in the lower right corner. The experiment can be seen in figure 5.15. As one can see, the robots immediately form a sweep line which walks over the space, always separating contaminated from free areas. It should be noted at this point, that this behavior is directly emerging from the planner and the assumed restriction of forbidden recontamination.

Apart from the maps we also tried different parameter settings, for example different intruder speeds, different sensor ranges and slower robots. While

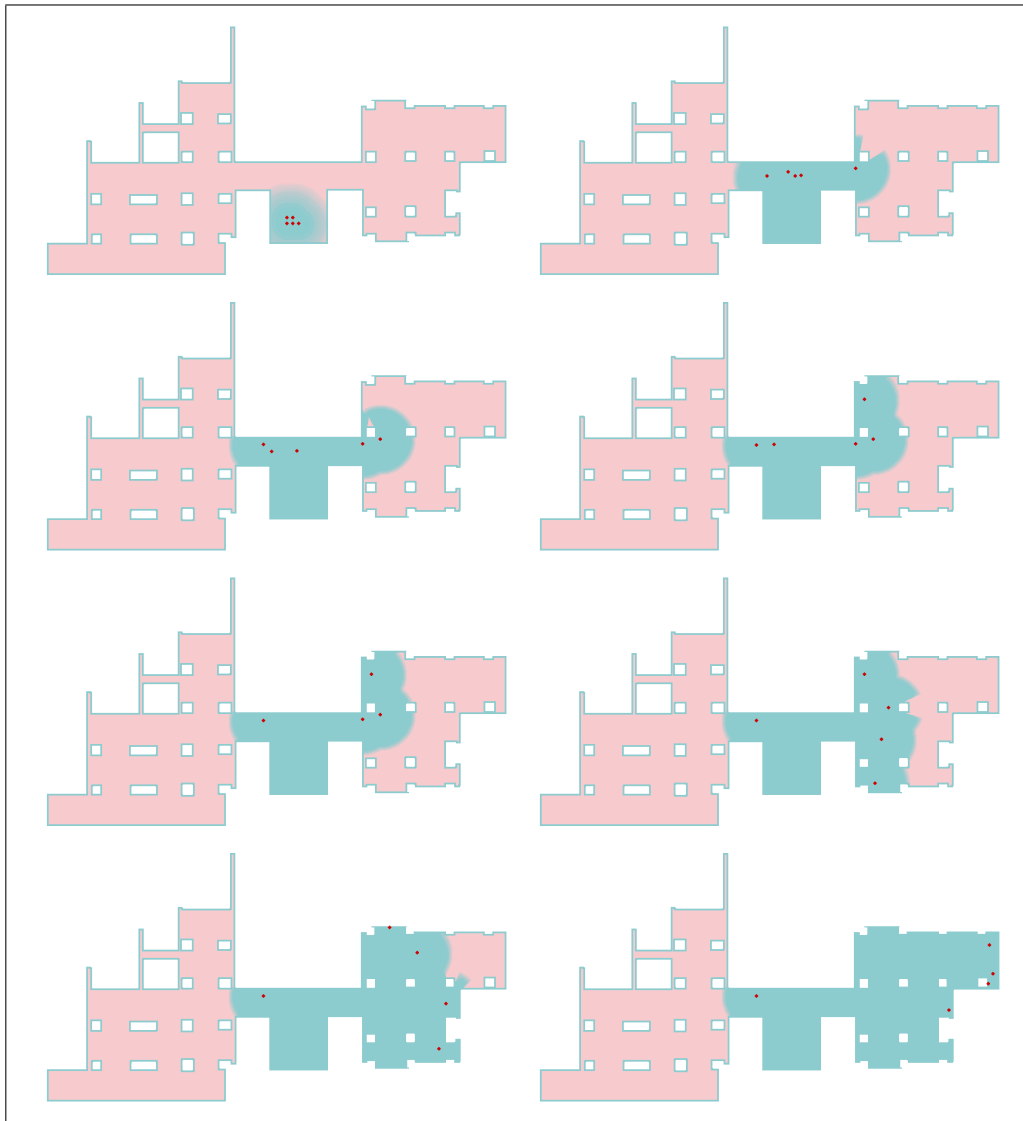


Figure 5.12: Example of an experiment with 5 robots. Flow is line by line, every row represents 150 time steps of the simulation system.

robot speed does not affect the quality of the outcome (only the time needed to travel the map), changing the speed of the intruder to an extreme value can lead to failure. This originates from the fact that a very fast intruder



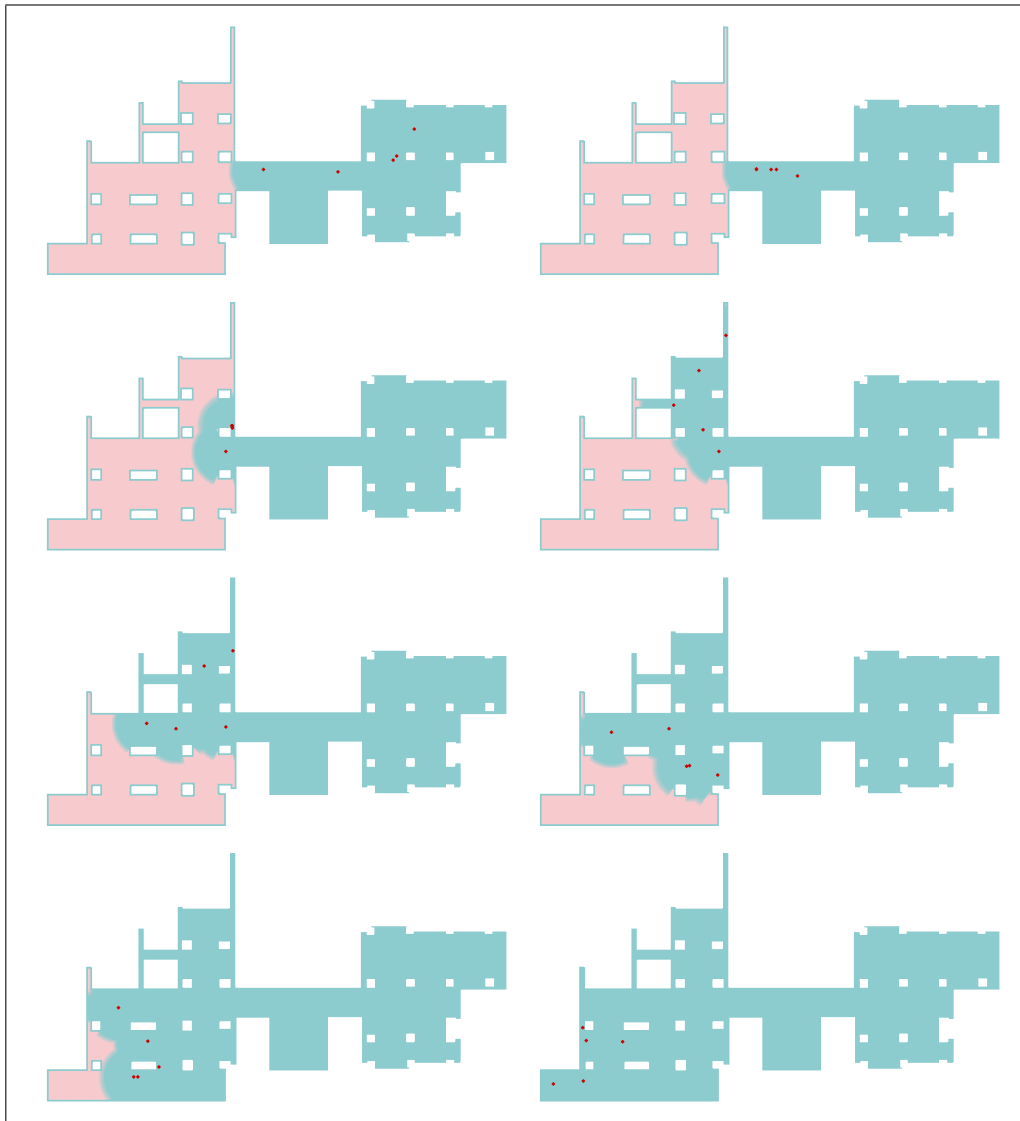


Figure 5.13: Example of an experiment with 5 robots. Flow is line by line, every row represents 150 time steps of the simulation system.

can simply cross the field of view of a pursuer between two scans. However, while finding this problem during simulation runs, we don't consider it to be a realistic problem. If we assume a scanning frequency of 10Hz (which is a

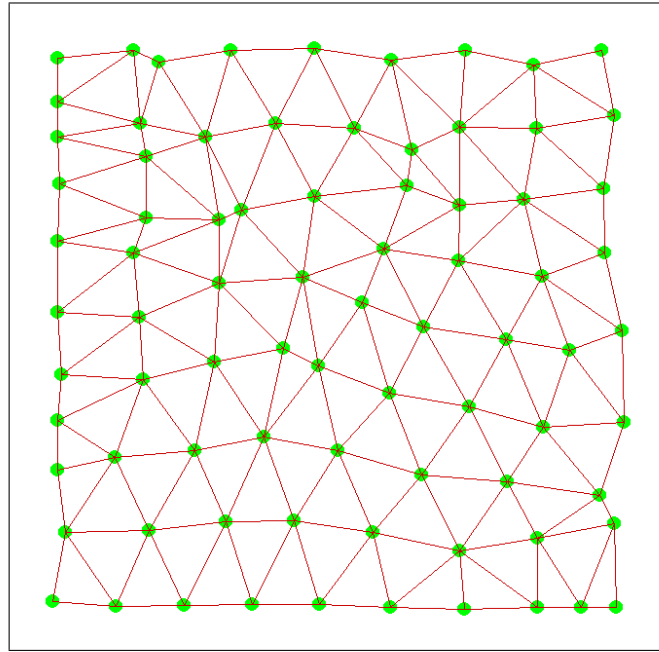


Figure 5.14: Decomposition of the Empty Map

rather low value for modern sensors) , an intruder had to have a speed of more than 10 m/s to cross the outer limits of the field between two scans. Since an LMS SICK scanner generates up to 75 scans per second, we think this problem to be negligible. Changing the sensor range also does not lead to different results in quality, especially since this can simply be seen as a different map.

### 5.9.2 Time Based Planning II

We also conducted some experiments with real robots. Real robot experiments, especially experiments with groups of robots, come with a number of problems, which usually do not arise in a simulation system. Especially the tendency to system failure grows exponentially with the number of used robots. Therefore, we will use these experiments as a proof of concept and

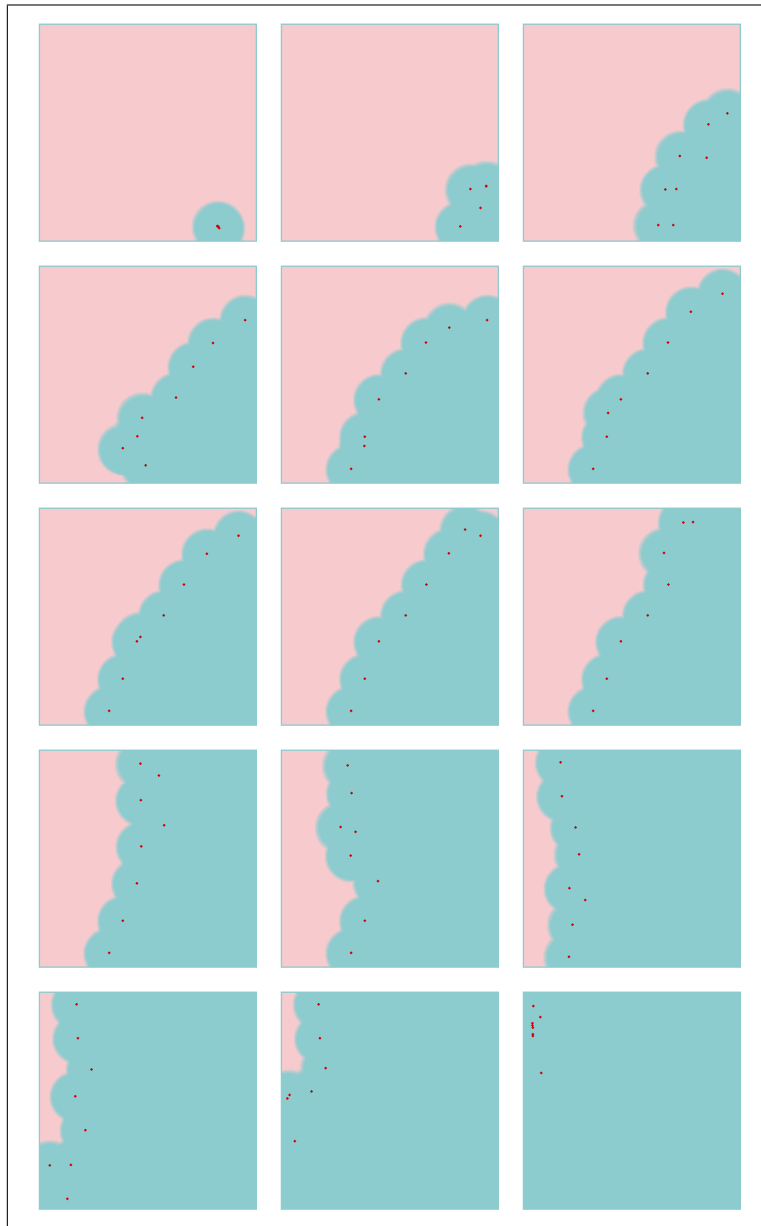


Figure 5.15: Example of an experiment with 8 robots. Flow is line by line, every row represents 150 time steps of the simulation system.

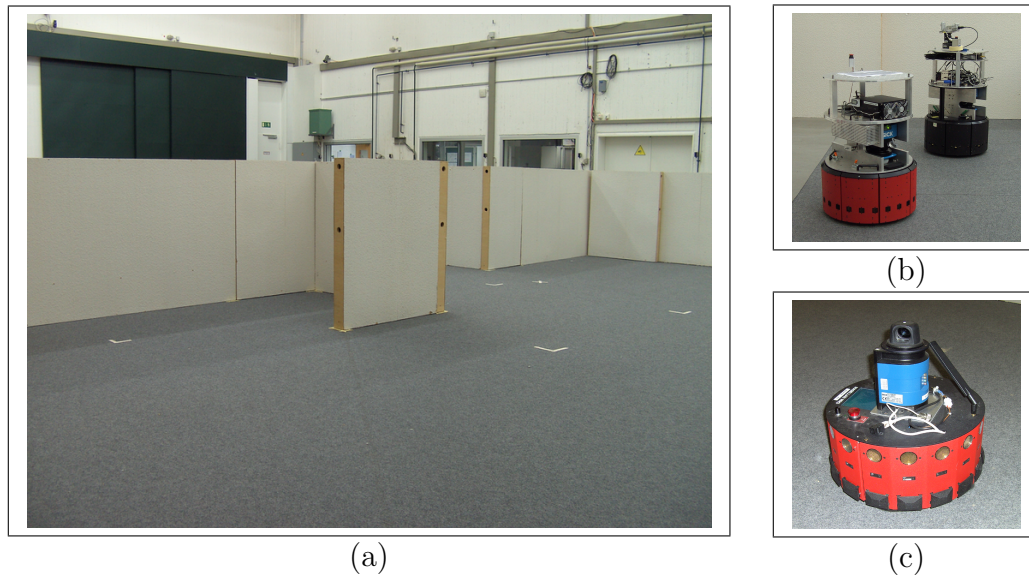


Figure 5.16: Robot proving ground and robots (a) Proving ground (b) Robots Bluecher and Clausewitz (B21 chassis) (c) Robot Moltke (Magellan chassis)

leave the comparisons between different methods and parameter settings to the simulation experiments.

We did the experiments in the robot proving ground of the Research Establishment for Applied Science (FGAN) in Bonn. The proving ground is an empty, rectangular hall of roughly 18x14 meters, with cameras at the ceiling which ease experiment documentation. Wooden Wallparts of different sizes can be used to generate different scenarios. Figure 5.16 (a) shows a photo taken inside the proving ground.

For the first experiment, we set up the scenario shown in figure 5.17 (a). After doing an exploration of the environment, the map shown in figure 5.17 (b) was created. Figure 5.17 (c) shows the resulting search graph. Our algorithm determined that this graph can be searched with two robots, if both robots start in the room with the three nodes. So we choose a group of two B21 robots, which are shown in figure 5.16 (b). Both use typical SICK laser scanners mounted in a back-to-back manner, which allows an (almost)

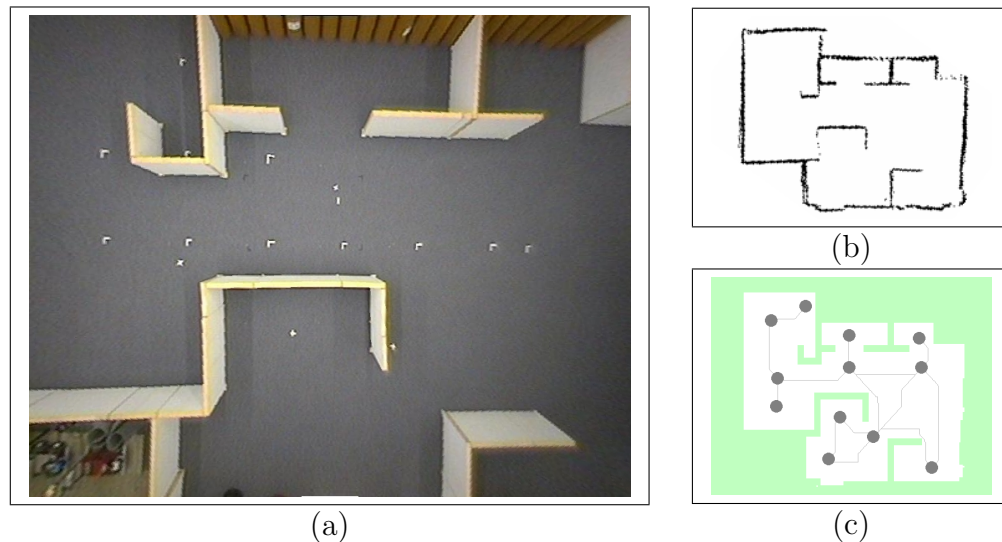


Figure 5.17: First Experiment in Time Based Planning. (a) Situation Overview (b) Localization / Robot Map (c) Resulting Search Graph

360 degree field of view. We did a small number of experiments, which all showed roughly the same results. We show one of them in figure 5.18. The flow is largely self-explanatory, the robots clear the right part before the left part.

For the second experiment, we set up a more difficult scenario, shown in figure 5.19 (a). As before, the robot map and the resulting graph is shown in 5.19 (b) and (c). The complexity of the environment demands a third robot in the team. In lack of a third B21 platform, we used a smaller Magellan robot (see figure 5.16 (c)) with a slightly different laser range scanner. Apart from the fact, that the navigation of the Magellan system is slightly worse than that of the B21s, we did not notice any difference in handling. The experiment itself is shown in figure 5.20. The flow is again largely self-explanatory, one should notice the typical sentinel behavior of the Magellan robot (snapshots 4-9), which guards the middle part of the map, while the other two robots clear the right part.

Unfortunately, the experiment failed in the last 20 seconds. The naviga-

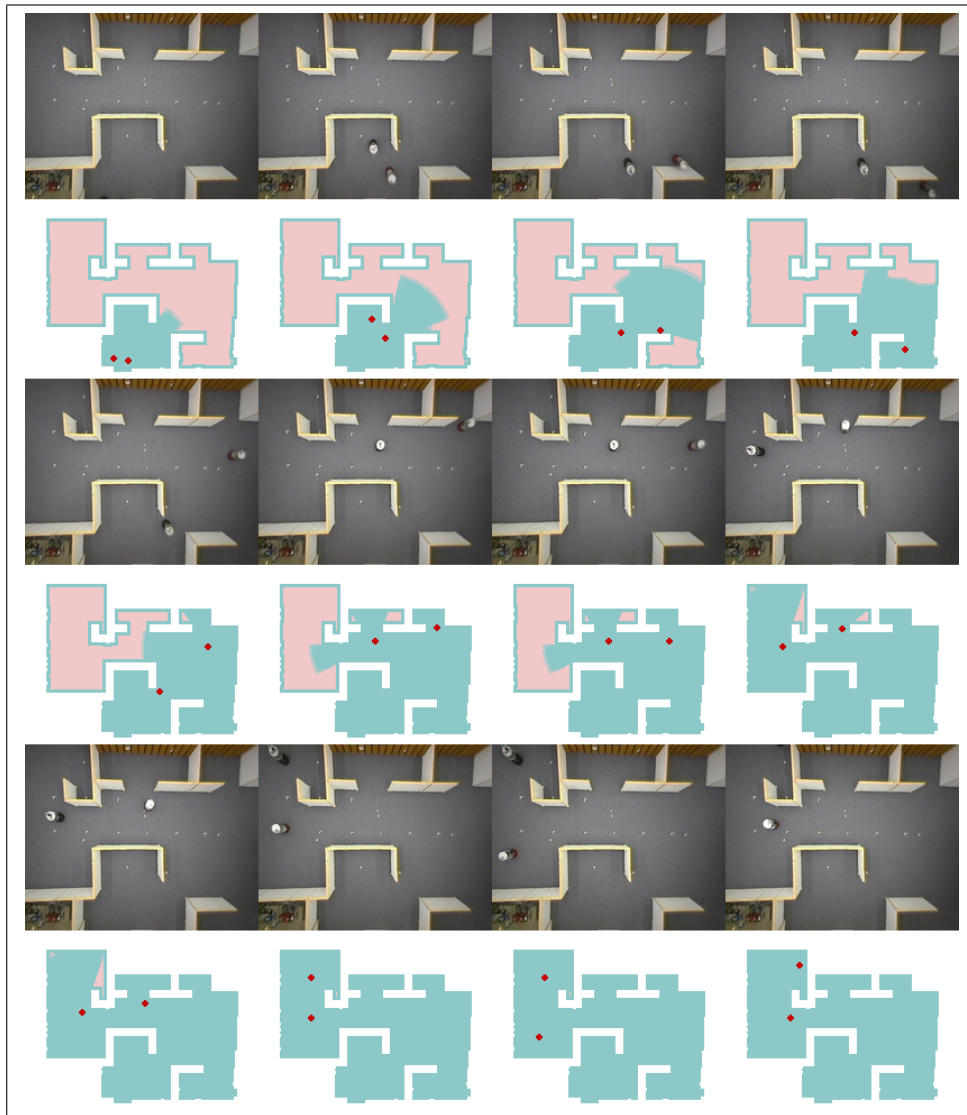


Figure 5.18: Example experiment for two robots. Shown are 12 snapshots during the experiments, flow is line by line. Every snapshot shows the situation as seen by the camera, as well as the contamination state calculated.

tion system failed to reach the last node and stopped roughly 80 cm before the given target. This leads to a very small area left contaminated at the

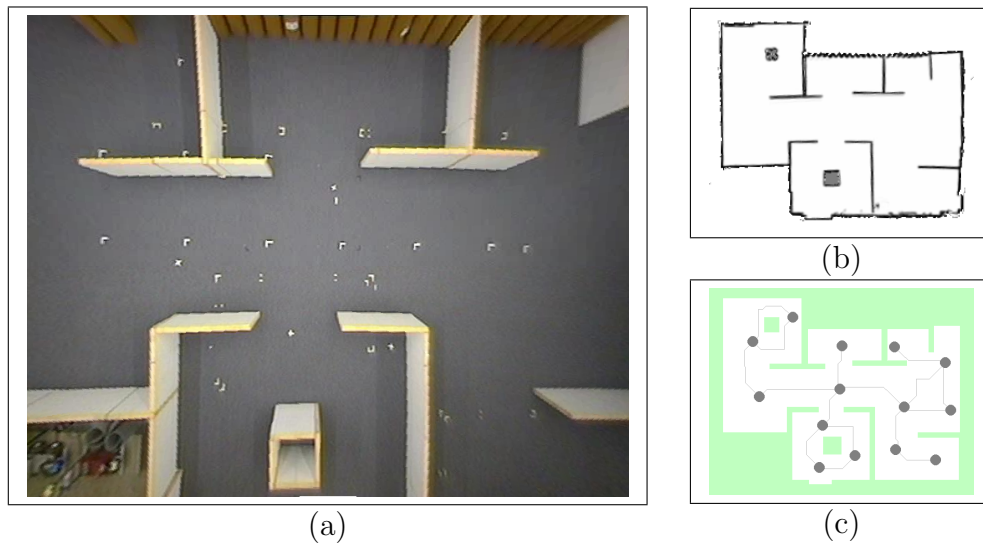


Figure 5.19: Second Experiment in Time Based Planning. (a) Situation Overview (b) Localization / Robot Map (c) Resulting Search Graph

end (see last snapshot).



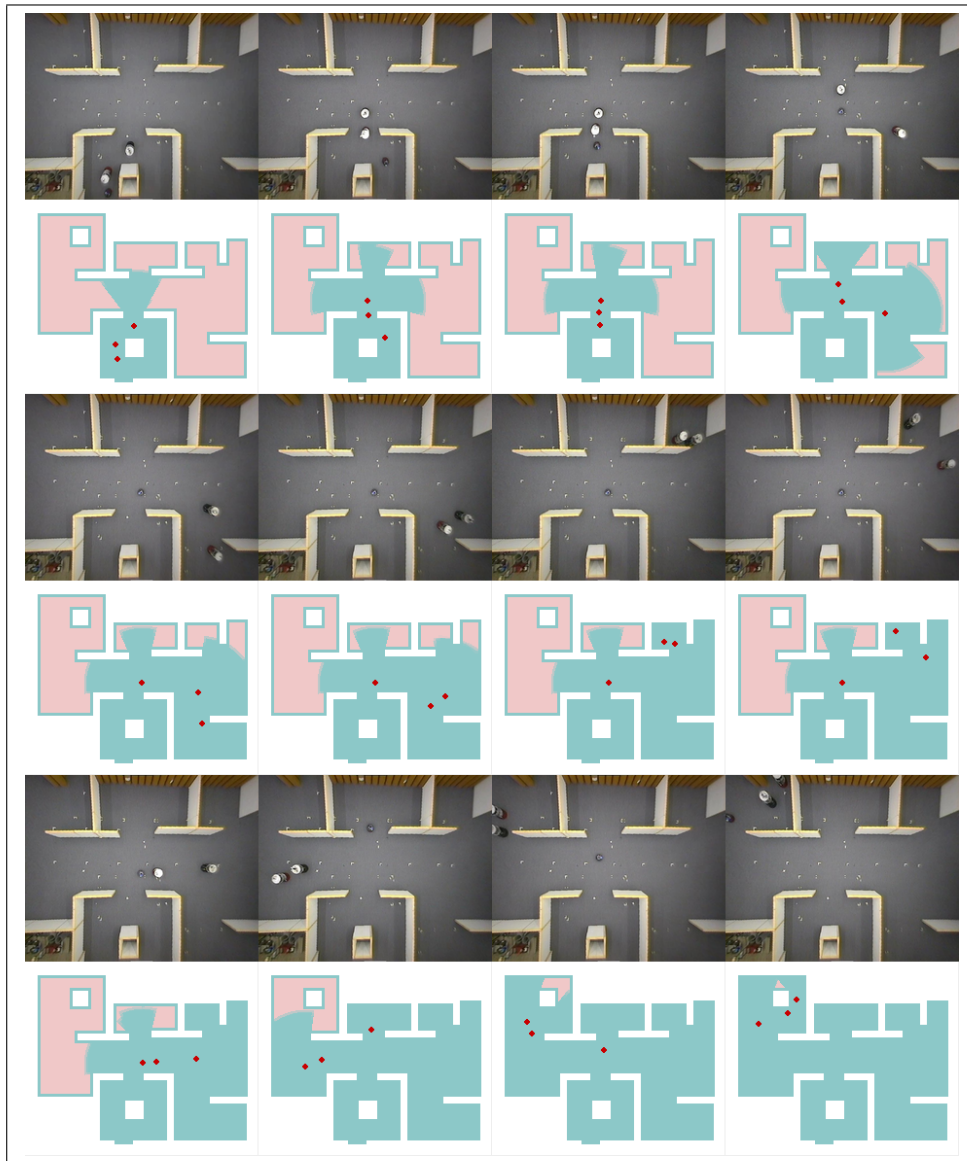


Figure 5.20: Example experiment for three robots. Shown are 12 snapshots during the experiments, flow is line by line. Every snapshot shows the situation as seen by the camera, as well as the contamination state calculated.

### 5.9.3 Wavefront Expansion

We also did some experiments using the wavefront expansion technique. For the purpose of comparison, we used the same settings as in the experiments



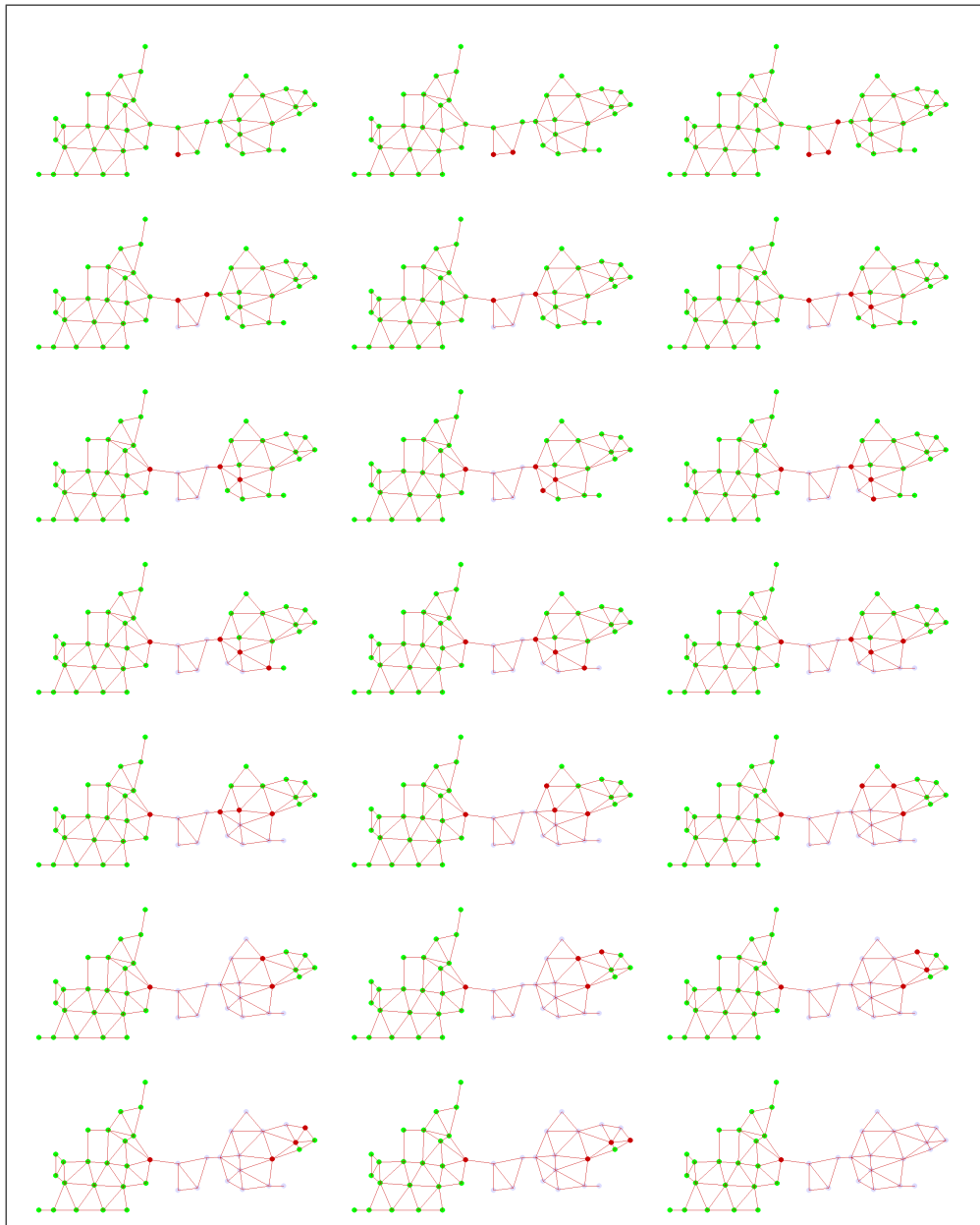


Figure 5.21: Example run for the wavefront expansion technique. Flow is line by line. At the beginning all vertices, except the starting vertex are considered contaminated (green). Every planning step expands the wavefront to decontaminate one vertex. Wavefront is shown as red vertices.

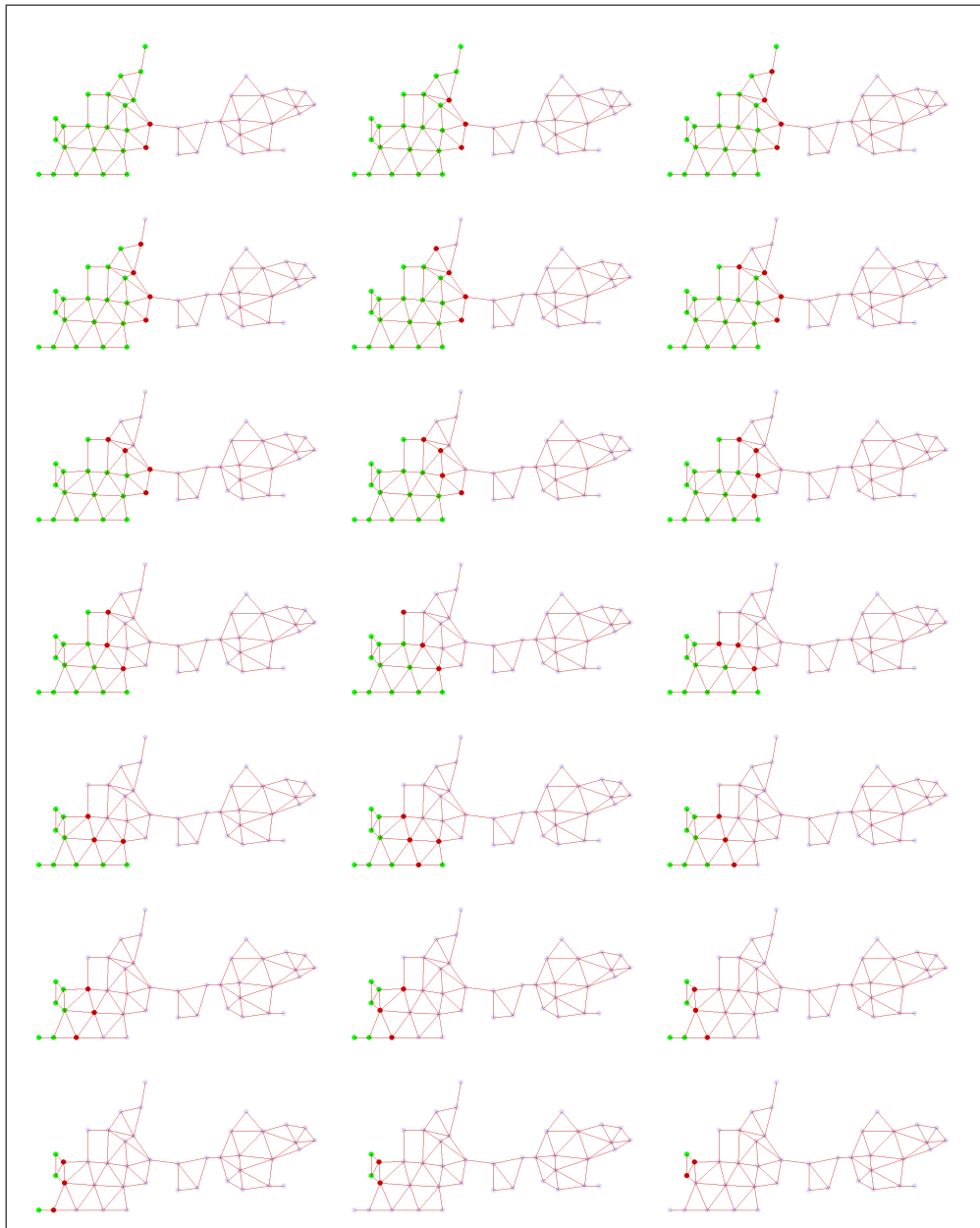


Figure 5.22: Example run for the wavefront expansion technique. Flow is line by line. At the beginning all vertices, except the starting vertex are considered contaminated (green). Every planning step expands the wavefront to decontaminate one vertex. Wavefront is shown as red vertices.

presented before. Interestingly, the experiments lead to almost the same solutions as the time based A\* planner, which is somewhat surprising, given that the planning spaces are completely different. An example can be seen in figures 5.21 and 5.22.<sup>4</sup> As before, the robots start in the middle of the map, clear the right part while guarding the middle and finally clear the rest of the graph.

This observation held for almost all maps we tested, the solutions differed only in minor aspects.

## 5.10 Advanced Problems

To examine the strengths and weaknesses of the graph decomposition method and especially the planning component, we have generated a number of more difficult problems in terms of size and complexity. We generated a collection of five huge artificial maps, each sized of  $100 \times 100$  meters, each with a completely different setting of obstacle structure. These maps can be seen in figure 5.23. Adding to this, we introduce a floor plan of an existing hospital building which is roughly  $100 \times 60$  meters in dimensions and consists of almost 100 distinguishable rooms.

### Artificial Map One - Spiral Office

The first map to test can be seen in figure 5.23 (a). The structure is an artificial office building, which houses about 60 rooms around a spiral floor. The structure is an exaggeration of an ordinary office building with the idea in mind of creating a very "deep" problem in terms of the necessary search tree. The so created building contains free space of 801153 cells, translating to roughly  $8000 m^2$ , where an intruder could hide. To generate the graph we assumed an effective scanning range of 5 meters for the pursuers. In this case the space decomposition lead to graphs of around 300 vertices to cover the full space, depending on optimization time and number of restarts. An example of a so generated graph can be seen in figure 5.25. We tested

---

<sup>4</sup>Since we have already shown in the last section how the execution of a plan looks like for the intruders distribution, this time we would like to present the underlying graph.

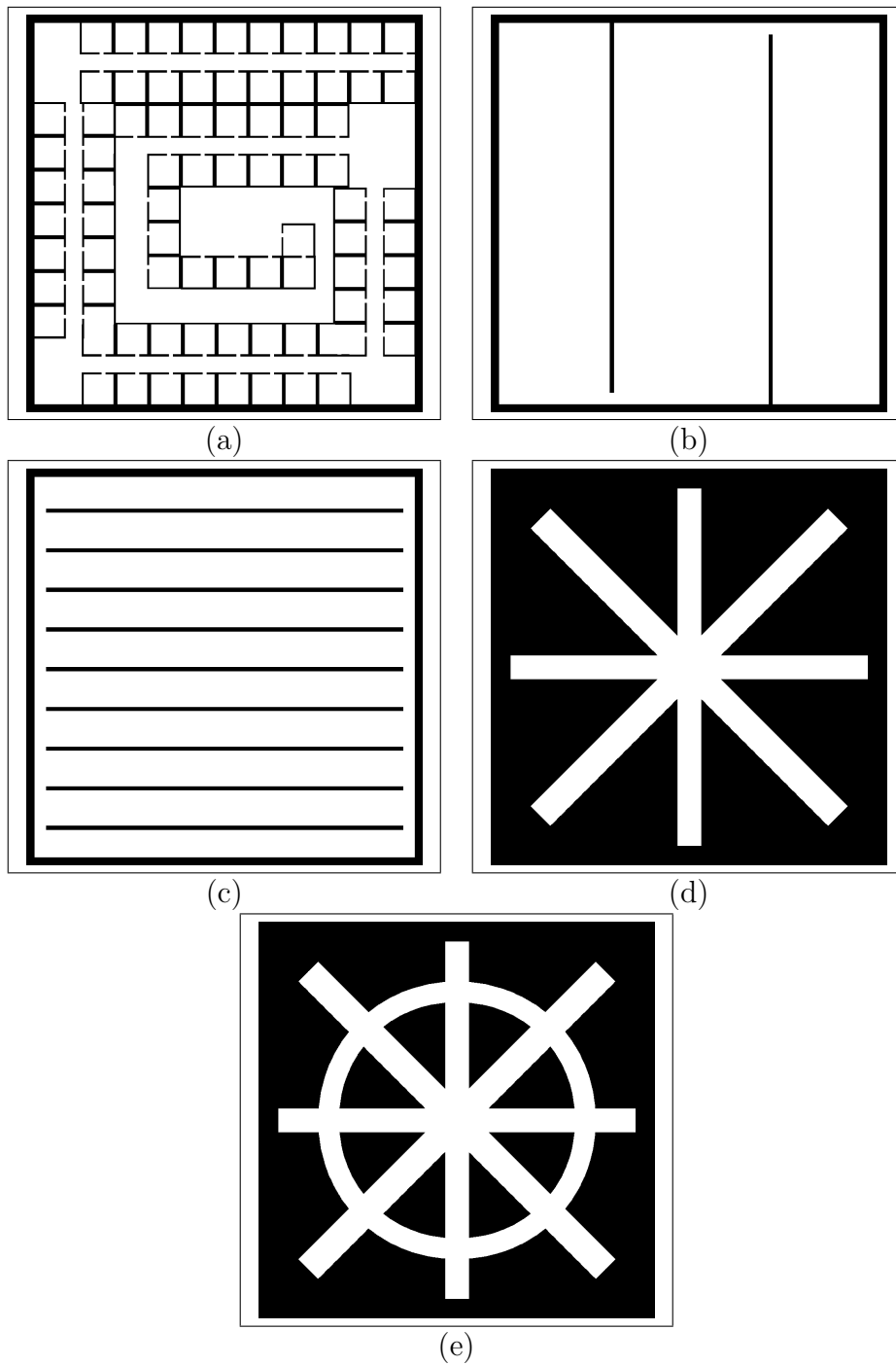


Figure 5.23: Advanced Test Maps.

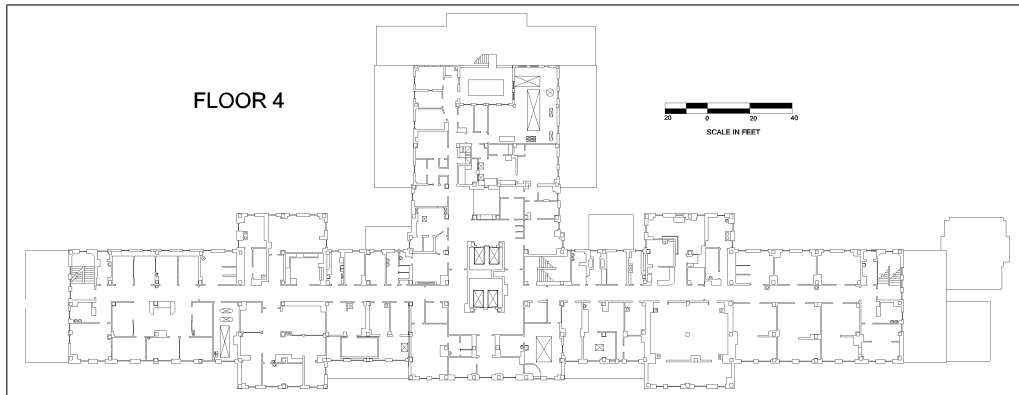


Figure 5.24: Floor plan Hospital Building

the normal A\* planner as well as the wave front technique with the shown graph. We tested different team sizes as well as different starting locations. The results are the following:

The normal A\* planner finds no solution for less than four robots. The planner usually gives up after 10-15 seconds. With four robots a solution can be found if the robots start in the middle of the map. Outside the middle five robots are necessary for a complete plan. The planner usually takes 10-40 seconds for a full plan (with the expansion of 5.000-15.000 planning nodes) and the plan itself is usually around 26000 robot cell-movements, which translates to a search time of 108 minutes under the assumption of a robot speed of 40 *cm/s*.

Contrary to the normal A\* planner, the wave front technique does not only give up on three robots, it also proves through exhaustive search that no solution for three robots exists. This can be done in less than two seconds per starting point. So the search number of the graph is four. With starting points in the middle, a solution for four robots is usually found in less than five seconds (with the expansion of 70.000 (wave front) planning nodes). For some points outside the middle it can also be proven by exhaustive search that no solution for four robots exist, if all robots have to start at this point and recontamination is forbidden (see section 5.8.3 for discussion on this anomaly).

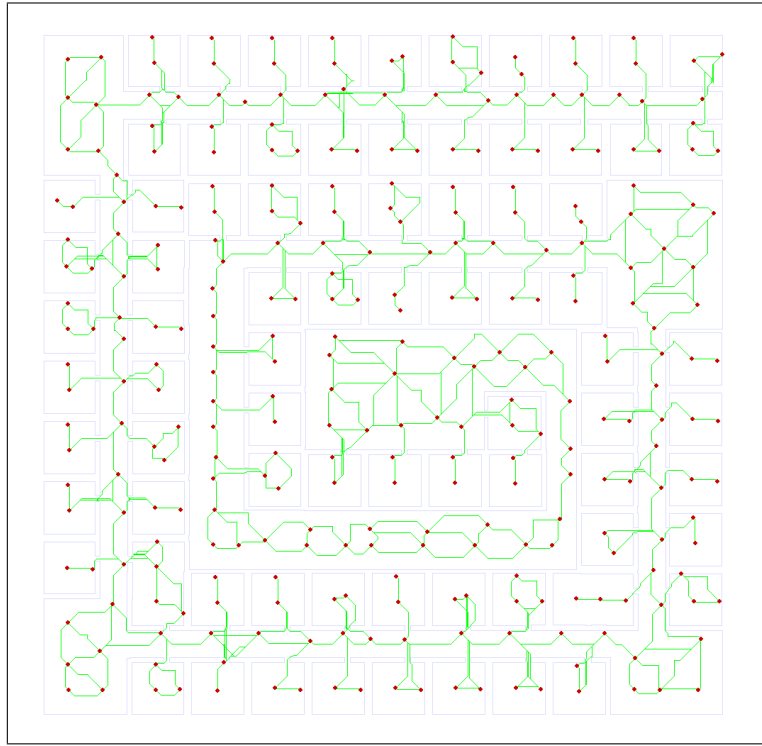


Figure 5.25: Graph decomposition of the spiral office map

Judging those results, both planners do not have any problems with the depth of the problem. While the A\* planner generates better solutions, the wave front planner can prove for some starting instances that a solution with less than 5 robots does not exist.

### Artificial Map Two - Divided Hall

The second map to test can be seen in figure 5.23 (b). The map shows a large hall divided by two walls into three sections. The sections are connected through small openings in the dividing walls. The map is huge and consists of 859664 empty cells, which corresponds to nearly  $8600 m^2$ . But even if the map is bigger in size than the first, the decomposition is able to find a complete covering with 180 vertices due to the open nature of the free space. Again, we tested both techniques with different starting locations.

The normal A\* planner was able to generate plans with 6 robots in less than 5 seconds, plans with 5 robots could not be successfully generated. The plans had a length of approximately 6500 robot movements, which means a search time of 27 minutes. The wave front technique shows similar results and can also prove through exhaustive search that no solution with less than 6 robots exists.

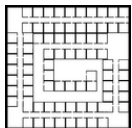
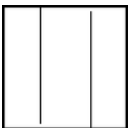



### **Artificial Map Three - Stack of hallways**

The third map to test can be seen in figure 5.23 (c). The size is 829466 empty cells ( $\approx 8300m^2$ ) and the decomposition leads to 250 vertices. The idea of this map was to generate a problem, which can either be solved by a small group in a long time or with a large group in short time, something we also would like to discuss in the following section about the use of additional robots. The time based planner generates a plan with four robots in less than five seconds. In this case the robots would need around 14.000 moves, which corresponds to roughly one hour. The wave front technique behaves in the same way, but also proves that a solution with less than four robots doesn't exist.



### **Artificial Maps Four and Five - Star Shapes**

The last two maps can be seen in figure 5.23 (d) & (e). While being very small in size ( $1900 m^2/2520 m^2$ ), the complexity, especially of the second map, is comparable to the other discussed maps. The search number of the generated graphs is 3 and 5 respectively and both planning mechanisms found a solution in less than 10 seconds.

## Results

Map	(a)	(b)	(c)	(d)	(e)
					
Description	Circular	Divided	Stack	Star	Star 2
Size	8000 $m^2$	8600 $m^2$	8300 $m^2$	1900 $m^2$	2520 $m^2$
Sense Range	5 m	5 m	5 m	5 m	5 m
#Vertices	300	180	250	50	75
Search Number	4	6	4	3	5
Calc Time A*	<20 s	<5 s	<10 s	< 5 s	< 10 s
Calc Time WF	<10 s	<10	<10 s	< 5 s	< 10 s
Proven Bound	4	6	4	3	5
Plan Time	108 min	27 min	60 min	63 min	78 min

## Results Hospital Map

Map	Hospital	Hospital
		
Size	10,000 $m^2$	10,000 $m^2$
Sense Range	5 m	3 m
#Vertices	450	800
Search Number	6	7
Calc Time A*	10 min	Fail
Calc Time WF	10 s	30 s
Proven Bound	-	-
Plan Time	150 min	-



### Hospital Map

The last map to test is a floor plan of an exiting hospital building. The floor consists of almost 1 million free cells, which corresponds to almost  $10.000\text{ m}^2$  (= 1 hectare) of free space. Due to the complex nature and all the small walls and rooms, the decomposition leads to graphs of no less than 450 vertices, which is therefore the most complex structure we analyzed. The complexity took both planners to their limit. The A\* based planner is able to find a solution for a team of six robots, but needs more than 1,2 million planning nodes to find it, which takes around 10 minutes of computation time on a modern computer (P4/3GHz). While this time is still small compared to the plans execution time (2,5 hours), the increase in calculation time compared to the other examples is tremendous. It should also be added that this result can only be achieved by choosing a good starting node, for many nodes no plan with 6 robots can be found at all. On the other hand, if the number of robots is increased, for example to 8, a solution can be found much faster, mostly in less than 20 seconds. The wave front technique however, is able to find a solution for 6 robots within seconds (from almost every border node), but is unable to prove that 6 is the lower limit of the map, since the search space for 5 robots is already too big to be searched exhaustively. The map clearly shows the advantages of the wave front technique in graphs of high complexity. To finally characterize the limits of the planner, we extended the last experiment by changing the range parameter of the pursuing robots. We changed the maximal detection range of the robots from 5m to 3m, which lead to a graph of 800 vertices. The wave front technique was able to find a solution in 30 seconds which uses 7 robots. The normal time based planner was unable to find a solution within acceptable time.

## 5.11 Additional Robots

In the context of the previously discussed graph decomposition techniques, perhaps the most central question was, how many robots or pursuers are needed to guarantee an intruder to be found in limited time. Naturally, if the number of available robots is lower than this limit, an intruder could hide

forever, at least if he knows the search pattern of the pursuers in advance.

Contrary to this, the question arises, how the use of additional robots, which are not necessarily needed, affect the outcome of the system. For the case of the wave front expansion technique, this question is only partly reasonable, since this technique does not generate actual plans and is therefore not able to make use of additional robots. Therefore we restrict the question to the time based A\* planner.

We did a number of experiments for three representative maps (artificial maps (a) and (c) as well as the hospital map) with different team sizes (ranging from 4 to 50 pursuers). The results can be seen in figure 5.26.

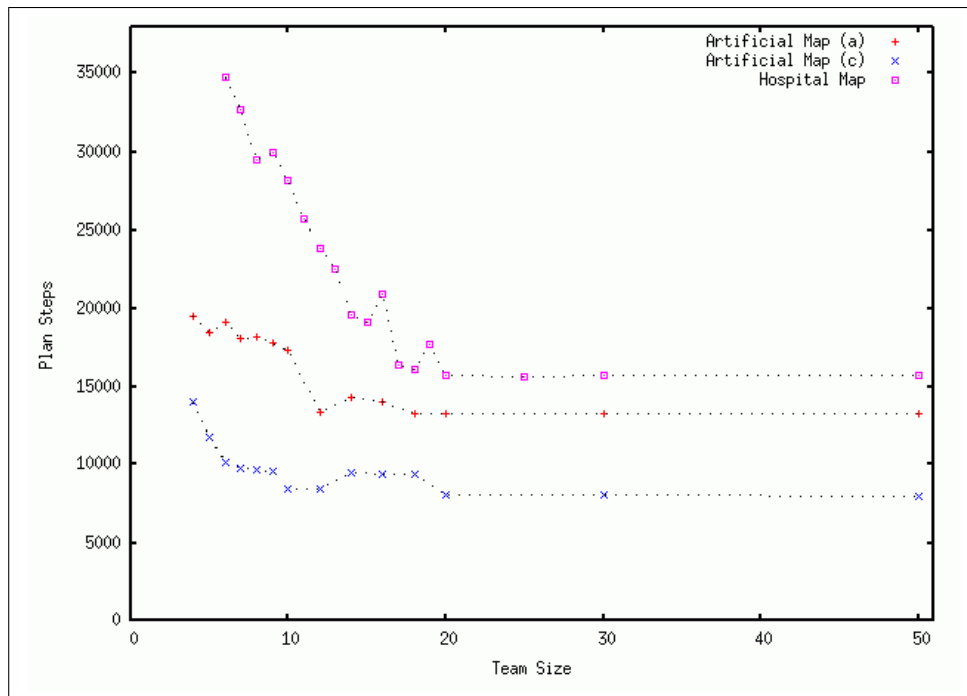


Figure 5.26: Time to clear a given map with different group sizes

For all three maps, a speedup can be achieved by using a larger team. This effect is especially strong for a small number of additional robots (mostly up to three) and gets weaker for larger numbers. At a certain point (depending on the map, from 5-15 additional robots) the search time reaches its minimum

and increasing the team size does not provide any further speedup. When one looks at the used maps, the results can easily be explained. In the minimal team size of 4, the artificial map (a) requires one robot to stay in the floor, while the other three check the rooms for intruders. The system has always to wait for the rooms to be cleared and therefore the first robot has often to wait. If additional robots can be used, there is no need to wait, so the main limit is its time to reach from one end of the floor to the other. Artificial Map (c) shows similar results, but from its structure reaches its limit at a team size of approximately 10 robots (4 robots minimum size plus 6 additional robots). Finally, the hospital map shows the best results for increasing the team size. If the team size is increased from 6 to 20, the whole space can be searched in almost 40% of the original time.

To summarize this, it can be stated, that the use of a small number of additional robots can speedup the search process to a certain extent, however the speedup itself is strongly dependent on the structure of the map.

## 5.12 Critic & Directions of research

We have already discussed some minor weaknesses of the described method, for example the still open question of how to generate optimal decompositions of the search space or the non-admissible heuristic function. We also sacrificed the optimality in the number of robots needed with the exclusion of plans, which require recontamination. While these problems exist, they are usually not an issue and can be coped with application of more computing time. However, the underlying model assumption still has a major weakness, which cannot be coped with so easily. To cover the worst case scenario, we assumed an intruder with perfect knowledge, who does not only know the current positions of all the pursuers, but also their position in the future. Such an intruder is realistic in case of deterministic pursuers, where an intruder could compute the paths of the pursuers in advance. However, the assumption gets weak in case of nondeterministic agents. To clarify this, we will give an example, which can be seen in figure 5.27. In this situation, the pursuer starts in the middle and has to clear the full space. If we assume an intruder with perfect knowledge (and at least the same speed as the

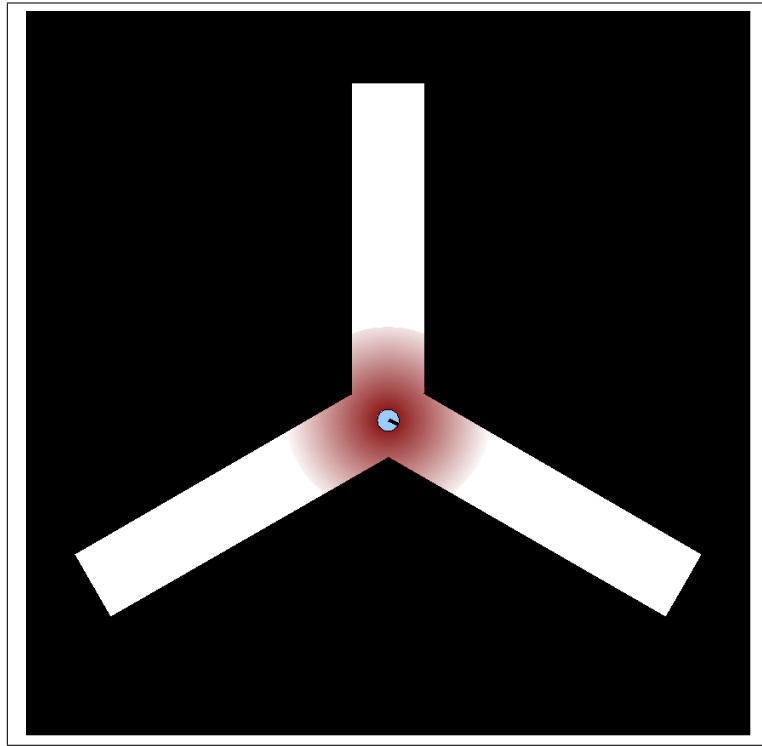


Figure 5.27: Example Game

pursuer), even of the future, one robot is not enough to clear the map. This is simple to see: Whenever the robot returns to the junction in the middle, the intruder knows which corridor the robot will take next and hides in the other. In differential game theory, a Nash-Equilibrium [Nas50] is a concept in which both players would not change their own strategy, given that they know the strategy of the opponent. Note, that there is no Nash-Equilibrium in the game just described.

In the real world, normal intruders can be smart, but usually cannot determine the future. In this case, the pursuer can catch the evader by using a simple trick: Whenever he returns to the junction from one corridor, he simply chooses randomly one of the two remaining corridors and searches it completely. When he doesn't find the intruder, he returns to the junction and chooses again. The chances of finding an intruder is 50 percent for every

choice, so by repeating this procedure long enough, the probability to catch the intruder can be raised to any desired value. Naturally, this is not inside the scope of deterministic game theory and leads to Bayes-games, which is beyond the scope of this work. It will not be proven at this point, but interestingly enough, these two strategies form a Nash-Equilibrium.

This example illustrates, that a realistic intruder, regardless how clever, cannot outwit a random decision. Summing up the above, we can say, that the assumed model of perfect knowledge leads to reliable and good strategies for nondeterministic robots with deterministic decisions, but there could be strategies involving nondeterministic decisions, which need less pursuers to fulfill the task.

A future direction of research should be the modeling of smart evaders without knowledge of future decisions the pursuers are going to make.

As stated before, the heuristic function used in the time based A\* planner is not admissible at this time. This can lead to suboptimal solutions but offers tremendous speedups in terms of calculation time. Another improvement of the method would be to find a better, possible admissible heuristic function, with similar behavior as the non-admissible function.



# Chapter 6

## Greedy Methods

### 6.1 Greedy methods

In the last section, we discussed planning techniques which always "guarantee" an intruder to be found in limited time, regardless of his behavior. This was implicitly achieved in the construction of a sweep line, which traverses the state space of the intruder. Naturally, these techniques fail, if the number of robots is too small for the required sweep line. In such situations, it is impossible to catch an intruder with perfect knowledge, since he always knows where to hide from the pursuers. So the question arises, what a security system should do, if the number of robots is very small and the area to be searched is very complex. As stated before, there is no way of catching an intruder with perfect knowledge. The canonical answer is, that such a system should try to find the intruder with maximum likelihood. But we will see, that even defining what this goal exactly means is not as clear as it looks firsthand. Parts of this work have already been published, for reference see [MS06].

### 6.2 Problem Formulation

In section 3.1.1 we discussed human models with limited knowledge of pursuit in detail. The simulated probability distribution allowed us to calculate the probability of catching such an intruder in the current time step, if we

know the location of every robot.

Let  $R^t = \langle r_1^t, \dots, r_n^t \rangle$  be the vector of robot positions at a timestep  $t$  and  $H^t$  be a simulated distribution of human intruders in the map at timestep  $t$ . Also define  $\tilde{R}^t = \langle R^0, \dots, R^t \rangle$  as the vector of robot position vectors<sup>1</sup> up to time  $t$ .

The probability to catch an intruder at a certain timestep will be named as  $P_{Catch}^t$  and depends on the intruder distribution as well as the robot positions, as well as the sensor. We therefore define

$$P_{Catch}^t = f(R^t, H^t) \quad (6.1)$$

for a suitable sensor function  $f$ . With this function, we have a means to calculate the performance of a set of robot positions up to time  $t$ :

$$U^t = 1 - \left( \prod_{i=0}^t (1 - P_{Catch}^i) \right) \quad (6.2)$$

With the performance equation, we can derive the solution for the best plan  $\tilde{Q}^t$  and its performance  $q^t$ :

$$\tilde{Q}^t = \operatorname{argmax}_{\tilde{R}^t} (U^t) \quad (6.3)$$

$$q^t = \max_{\tilde{R}^t} (U^t) \quad (6.4)$$

The equation shows the prior mentioned ambiguity of the question what the robots should do, since it can only be answered for a fixed time horizon.

To clarify the argument we give a small example, which can be seen in figure 6.1. Let the probability distribution for this example be as in the following table:

	A	B	C
H	0.1	0.3	0.6

The time needed for the robot to travel from its starting location to location

<sup>1</sup>The set of robot positions can be associated to a plan from the start to timestep  $t$



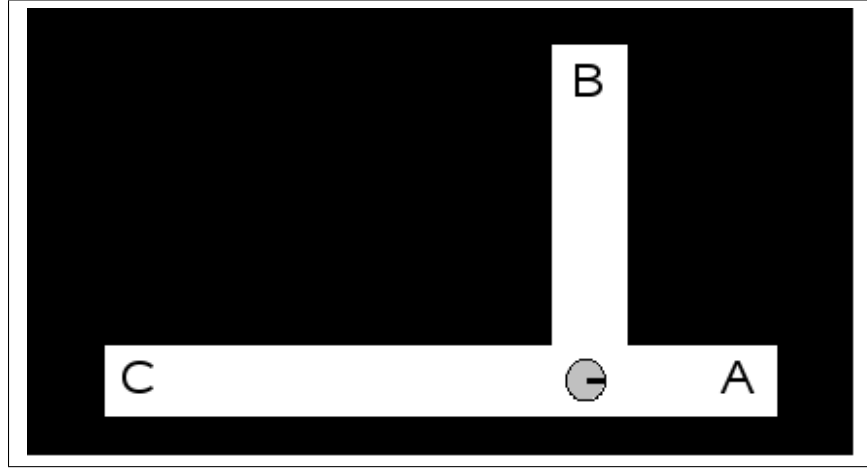


Figure 6.1: Example Decision Situation

A takes 1 timestep, to location B takes 2 timesteps and to location C takes 4 timesteps. To simplify things, we will assume, that the intruder doesn't move and the distribution is also not normalized after sensor integration<sup>2</sup>. It is easy to see that the best plan for the robot changes with the time horizon. If  $t$  is set to 1, the best plan is to move to location A. If  $t$  is set to 2, the best plan is to move to B and if  $t$  is set to 4, the best plan is to go to location C. For values greater than 2, the plans even lose their uniqueness, since one can find different plans with the same performance. The solutions are shown in this table:

Time horizon	$\tilde{Q}^t$	$q^t$
1	$\langle S \rangle, \langle A \rangle$	0.1
2	$\langle S \rangle, \langle S \Rightarrow B \rangle, \langle B \rangle$	0.3
3	$\langle S \rangle, \langle S \Rightarrow B \rangle, \langle B \rangle \langle B \rangle$ $\langle S \rangle, \langle S \Rightarrow B \rangle, \langle B \rangle \langle B \Rightarrow S \rangle$	0.3
4	$\langle S \rangle, \langle S \Rightarrow C \rangle, \langle S \Rightarrow C \rangle, \langle S \Rightarrow C \rangle, \langle C \rangle$	0.6

<sup>2</sup>Both assumptions are made to keep the numbers simple, dropping both assumptions does not change anything about the general conclusions.

As this example shows, the best solution highly depends on the chosen time horizon. A time (or planning) horizon of 4 steps can lead to completely different results than a time horizon of 3 steps.

One might argue at this point, that a larger time horizon is always superior to a smaller one, but even this is not necessarily true. In the example shown before, the best solution for 4 timesteps was to directly start moving from S to C. If the chosen time horizon is 6, the best solution would be to first move from S to A and then go to C. The solution is completely different from the one for 4 timesteps. This problem is very well known in game theory and a general solution is not known today.

## 6.3 General outline

If planning the solution of a problem becomes too complex, it is often suggested to generate good but suboptimal solutions with greedy methods. As in the case of solving the exploration problem, we would like to use an utility-based approach, since both problems have very similar internal constraints and can be solved by the same algorithms. In fact, intruder searching can be seen as an exploration problem with a shifting and decaying map.

### 6.3.1 Costs

The costs for reaching a target from a starting location can be directly taken from the exploration algorithm. A detailed description was already given in section 2.2.1.

### 6.3.2 Utility

In the exploration process, the utility of a target location was the expected information gain at that point. Naturally, the points with the highest information gain have been the frontier cells and so only frontier cells had to be evaluated as possible targets. In case of the pursuit-evasion-problem frontier cells do not exist. In this case all reachable cells have to be tested for their utility. Following the exploration approach, the utility of a cell shall be the

expected information gain, which is exactly the probability of catching an intruder, when doing one sensor scan at the specific location. While this utility can directly be derived from the simulated human model, it is a very expensive function in terms of calculation time, especially when taking a realistic sensor model into account. The problem is shown in the left part of

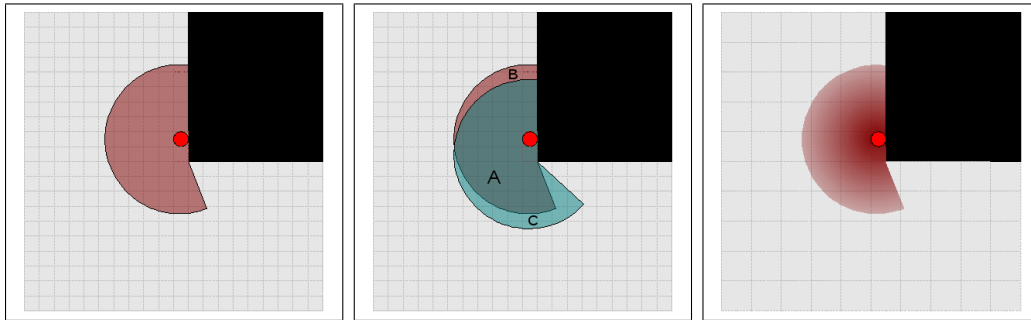


Figure 6.2: Utility calculation problem.

figure 6.2. If we want to calculate the utility of the cell we have to consider all cells, which are in range of a robot, standing on this cell and which are not blocked by obstacles. This region is shown in red. This region is mostly unique and it depends on local map geometry. Even if we consider the region of a cell to be calculated in advance, the online calculation of utility is very expensive. We developed two strategies to cope with this problem. The first one is to reduce the sensor model to a binary function, which only depends on range. In this case, the probability of catching an intruder, which is visible from a pursuer's sensor, is independent of the exact range, as long as it is within the maximal range of the sensor. In this case, we can take advantage of the local dependency of two neighboring cells. This is shown in the middle of figure 6.2. If we consider the cell marked with the red dot and the cell directly below, we can see that both regions have the area marked with an A in common, the only difference are the comparably small areas B and C. If the utility for the upper position has been calculated, the calculation of the lower can be done by subtracting the possibility to sense an intruder in region B and adding the possibility of sensing one in region C. The effect of this speedup increases with the level of discretization.

If we do not want to drop the sensor model (right part of figure 6.2, there is no advantage in local neighborhood, the only solution in this case is to use a smaller discretization.

### 6.3.3 Coordination

Even the coordination scheme can be taken from multi-robot-exploration. When one pursuer is set to a specific location, the information gain of all neighboring cells is decreased, depending on the sensors range.

## 6.4 The Algorithm

The algorithm is similar to the multi-robot exploration algorithm and is also divided in an outer and an inner part:

1. Initialize human distribution.
2. Execute **Target Point Selection Algorithm 2**.
3. Generate Plans for all assignments.
4. For  $\Delta$ -Timesteps do:
  - Execute robot plans.
  - Integrate robots sensors.
  - Integrate human motion.
5. GOTO 2

Multi Robot Searching Algorithm

1. Determine the set of all cells  $\mathbf{G}$ .
2. Compute for each robot  $i$  the costs ( $\mathbf{C}^i(g)$ ) for reaching every cell  $g \in \mathbf{G}$ .
3. Compute for each cell  $g$  in  $\mathbf{G}$  the probability to catch an intruder at the current time step and set  $\mathbf{U}(g)$  to it.
4. While there is one robot without a target point:
  - (a) Determine a robot  $i$  and a cell  $g$ , which satisfy:  
 $\langle i, g \rangle = \operatorname{argmax}_{\langle i', g' \rangle} U(g') - \alpha \cdot C^{i'}(g')$
  - (b) Add  $\langle i, g \rangle$  to the assignment.
  - (c) Reduce the utility of every cell  $g'$  according to:  
 $U(g') = U(g') \cdot (1 - f(\|g, g'\|_2))$

Target Point Selection Algorithm 2

## 6.5 Example

### 6.5.1 Target point selection

Figures 6.3 and 6.4 show an example of the Target Point Selection Algorithm. In the left part of figure 6.3, one can see the calculated costs for a robot in the left part of the map. As before, costs are calculated by a standard MDP planner. In the right one can see the calculated utility, based upon the human distribution<sup>3</sup>. Figure 6.4 shows the tradeoff between costs and utility. This tradeoff naturally depends on the chosen  $\alpha$ .

### 6.5.2 Multi Robot Searching Algorithm

We also want to show a typical experiment. As in other chapters, we use the ground floor map. In this example, the team of pursuers consists of three robots, which, as we have seen before, is a small team for this map. All

<sup>3</sup>In this case, the base for the utility was the distribution shown in figure 3.9(d)

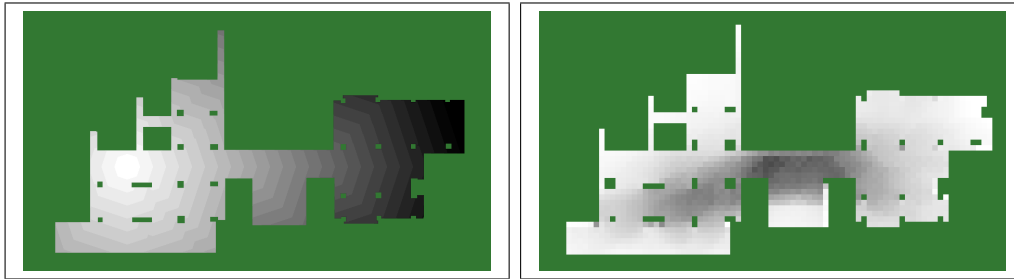


Figure 6.3: Costs and utility in the greedy search technique. Left: Costs for reaching any target from a robot standing in the left part of the map. Right: Utility of reaching a cell, based upon the human distribution.

three robots start in the middle of the map and have a maximum scanning range of 5 meters. The experiment is shown in figures 6.6 and 6.7, robots are painted as red circles. Depending on the probability of an intruder being in the cell, the cells are painted in a color range from white (low probability) to red (high probability). In very seldom cases, when the probability is very, very low a cell is sometimes painted in light grey, which means a value that can be treated as almost zero. The scale is logarithmic.

To give a better understanding of the course of action, we have also drawn the assignment computed by the Target Point Selection Algorithm in every situation. The targets are marked as green spots.

As one can see, the robots divide and start searching the area, while the distribution shifts in reaction to these movements. Based upon these observations it is difficult to evaluate the technique, apart from the almost trivial fact that the searchers are divided over the space, which can be seen, by looking at the trajectory of a similar experiment (figure 6.5).

## 6.6 Evaluation and Comparison

Contrary to the exploration problem, the evaluation of pursuit-evasion is far more difficult. While in the case of exploration, an experiment is always finished in limited time, the absolute time needed to do an exploration run can be compared. Pursuit evasion games, where an intruder is not guaranteed

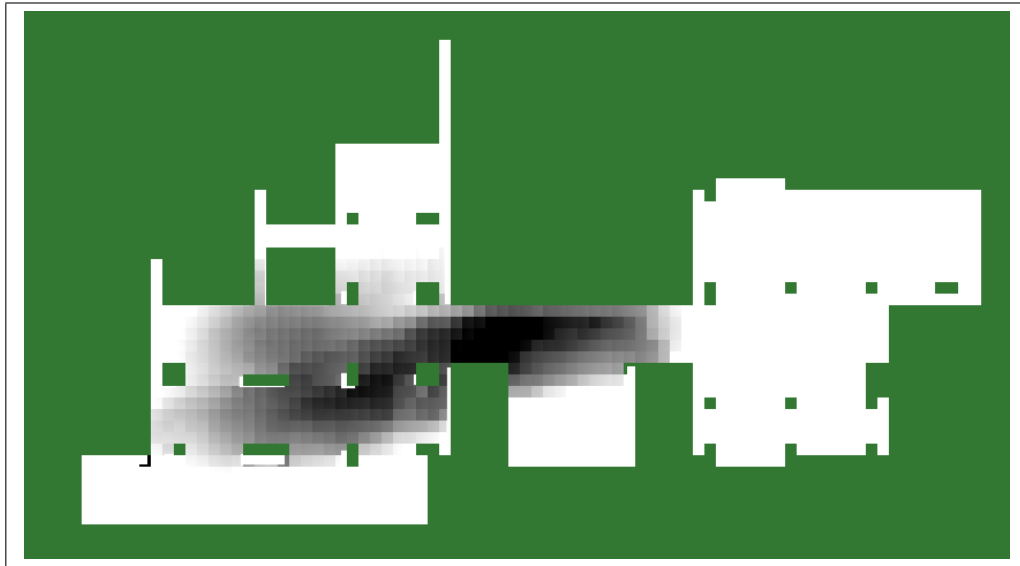


Figure 6.4: Tradeoff between utility and costs within the greedy search technique. Darker values represent cells with a better tradeoff.

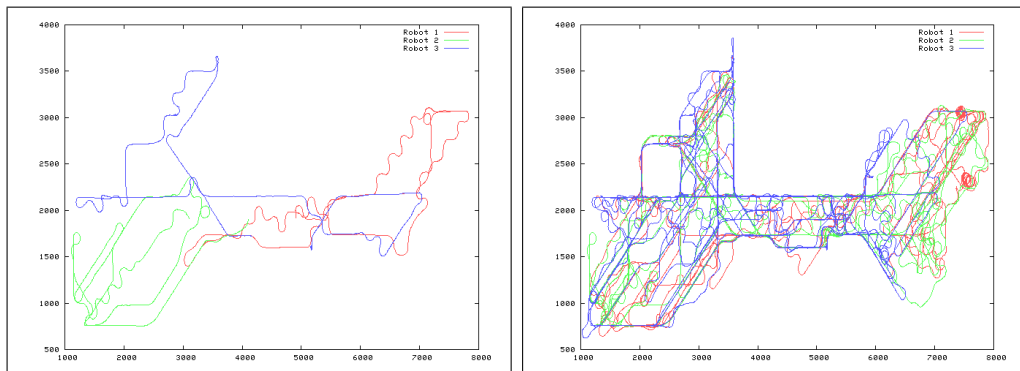


Figure 6.5: Robot Paths in a greedy search experiment. (a) Robot paths after  $\approx 6$  minutes. (b) Robot paths after  $\approx 1$  hour.

to be found, usually never end. If we apply the contamination metrics from section 4.1.2 and choose a robot group which is smaller than the search-number of the map, it is easy to see that the algorithm can never reach zero contamination. So we have to restrict us to the alternative metric, which is

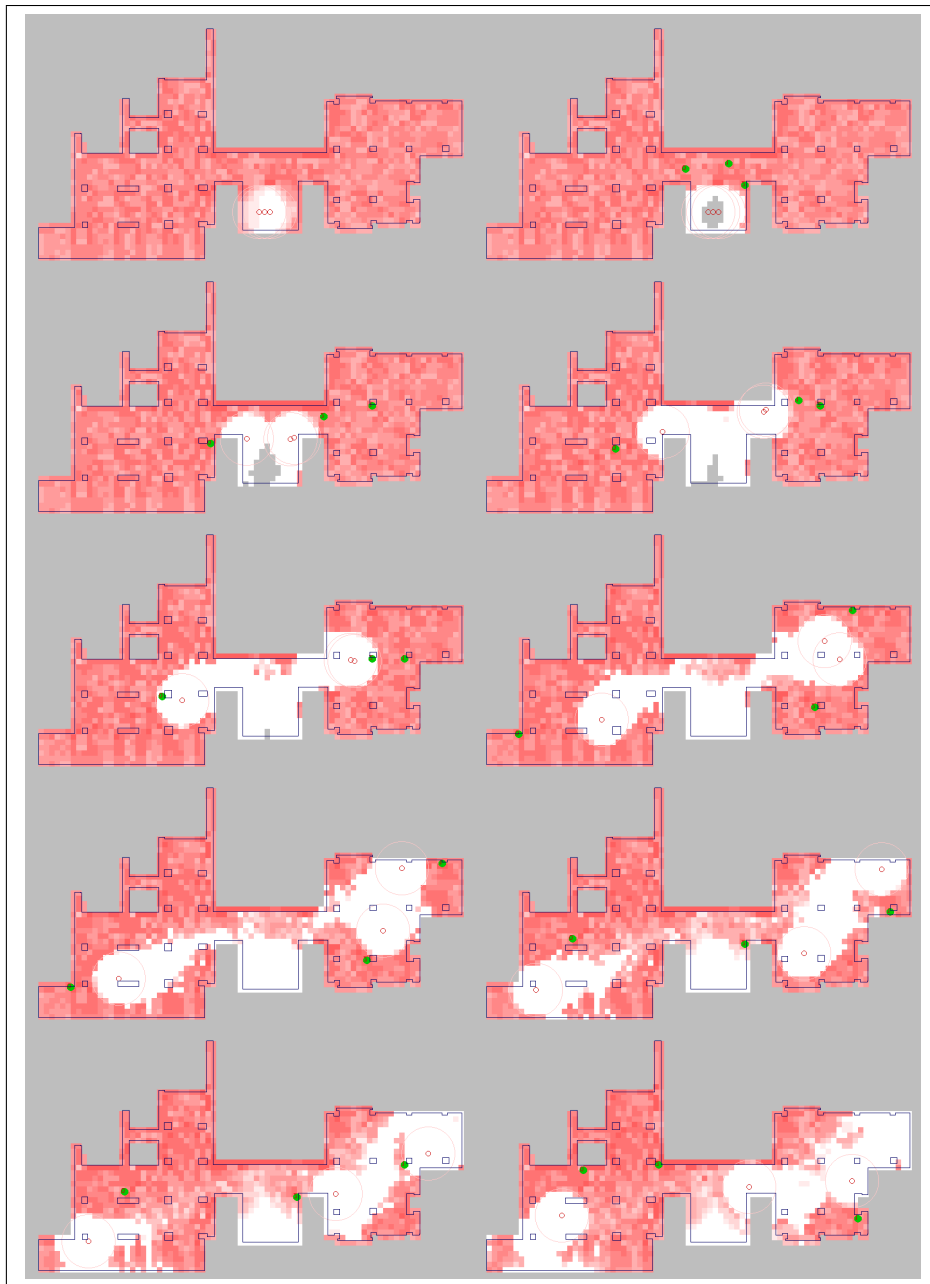


Figure 6.6: Greedy search experiment. Flow is line by line. Colors white to red show the probability of an intruder being at the cell. Red circles mark the robots, green circles mark the robots target assignments.



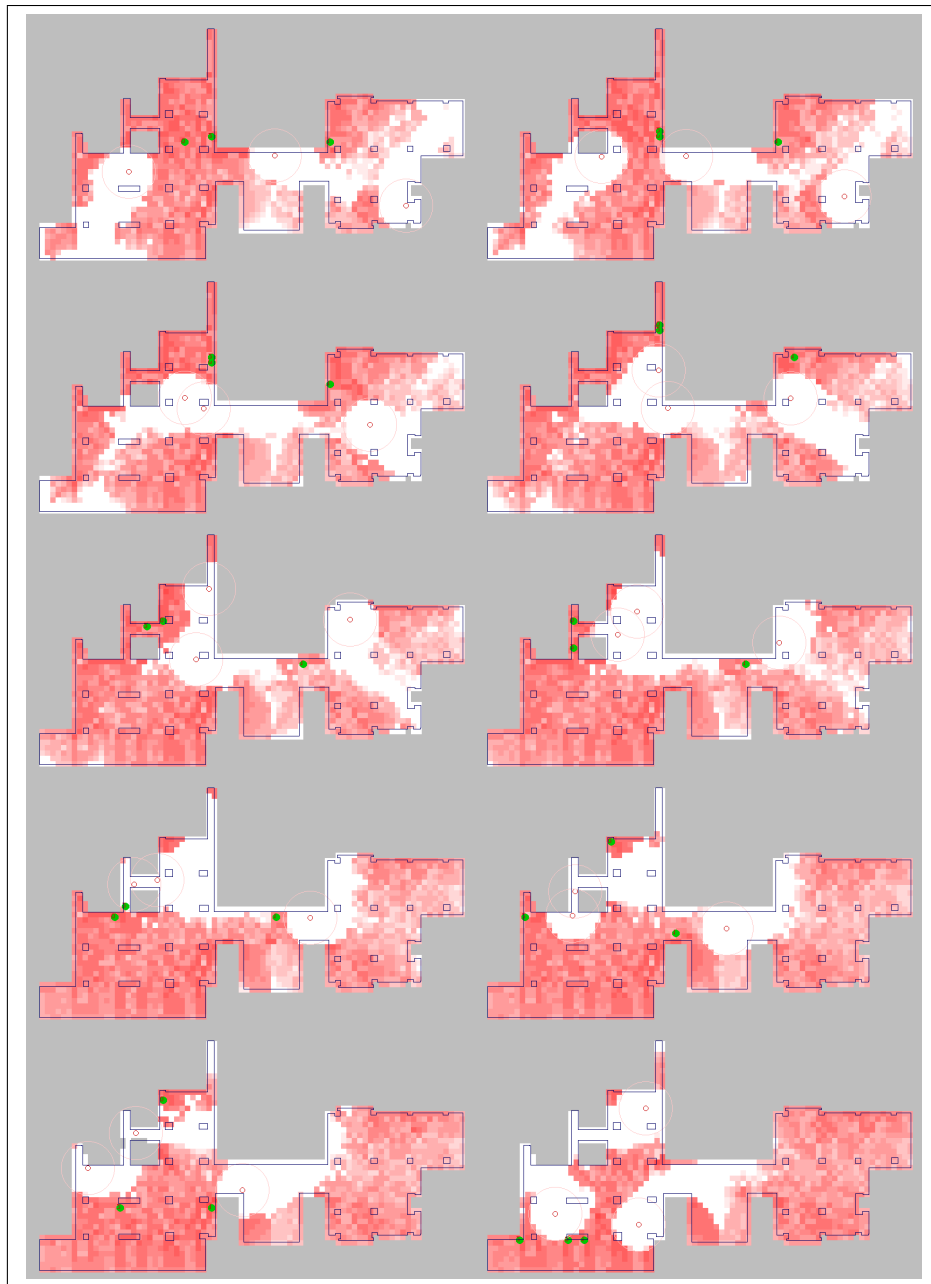


Figure 6.7: Greedy search experiment. Flow is line by line. Colors white to red show the probability of an intruder being at the cell. Red circles mark the robots, green circles mark the robots target assignments.

based upon the probability of catching an intruder with imperfect knowledge (see section 4.1.1). Unfortunately, this metric is rather weak in itself, since it requires a model, which may model a given intruder very accurately or not. In this case, the best evaluation is the probability of catching an intruder, who behaves as expected from the model, in a given time.

Another problem is the fact that the greedy algorithm is based upon exactly the same model as the evaluation, which undesirably links the evaluation method to the search method. This can not be prevented, since when we change the evaluation model to a better intruder model, the search method becomes unfairly handicapped unless we also change its model to the better one. It is easy to see, that there is no sense in using different models for the search and the evaluation.

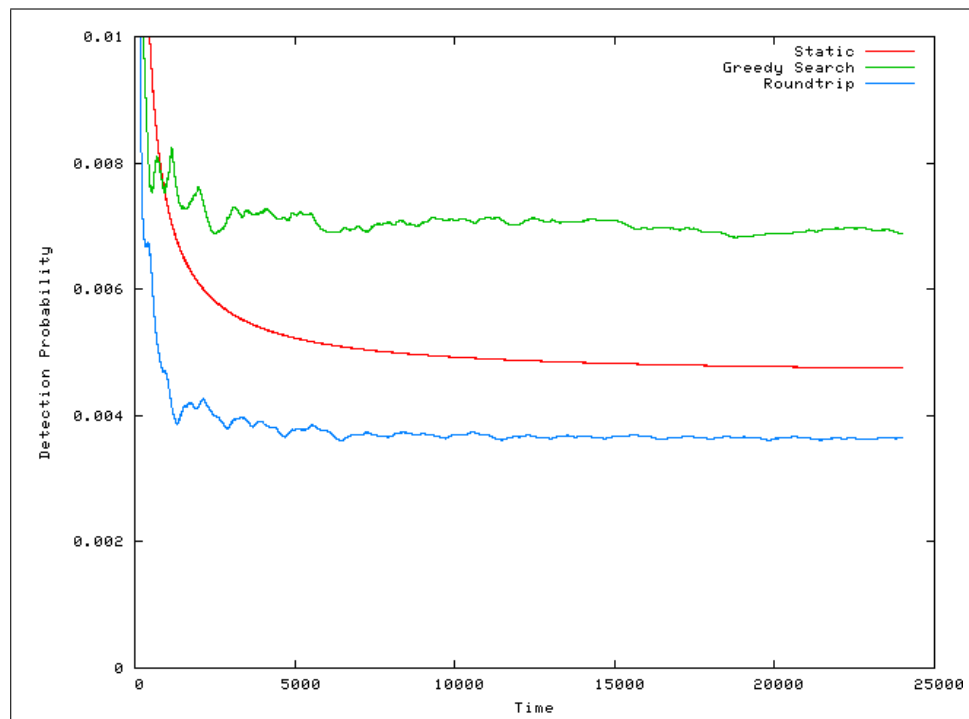


Figure 6.8: Performance of three different methods of intruder searching with three pursuers. Performance is strongly smoothed for better comparison.

The probability of catching an intruder at a given time can be directly

taken from the global human model simulator, since it is the counterpart of the model's normalizer (see subsection 3.2.2).

Figure 6.8 shows these probabilities for the above described example experiment. The green line represents the values for the described method. As one can see, the probability to catch is comparatively high at the beginning of the experiment (since the pursuers are dropped at the starting point therefore "catching" many intruders). After a short drop phase, the probability reaches a stable value of 0.007, which translates to a probability of 0.7 percent to catch an intruder at a given time step, which is the performance of the technique as a whole. This performance metric allows a direct comparison with simpler methods. In the same figure the performance of two other methods is also shown. The most simple method of placing static observers is shown in red, while the divided round-trip of three robots is shown in blue. As one can see, the performance of the greedy method is significantly better than the performance of the simple methods. In section 4.3 we have already discussed that the performance of a technique is strongly dependent on the ratio between intruder speed and pursuers' speed. In the setting of the just discussed experiments, the intruder speed was set comparatively high, which is the same as setting a slow speed for the pursuers, which results in an advantage for the group of static observers. If we increase the speed of the pursuers, we would expect this advantage to decrease. This is exactly what happens and can be seen in figure 6.9. In the right part, one can see the results just discussed. In the left part, the intruder speed was much slower (one sixth of the original speed), therefore the performance of the static system is strongly reduced. The ranking of the simple methods is inverted (see discussion in 4.3.3). More important is the fact that the greedy system outperforms the simpler methods regardless of the speed ratio, as long as the ratio is in realistic bounds.

## 6.7 Critic & Directions of research

As stated in the last section, one problem of the presented technique is the correlation between the evaluation function and the direction of the greedy search. Simply speaking, if the underlying movement model turns out to

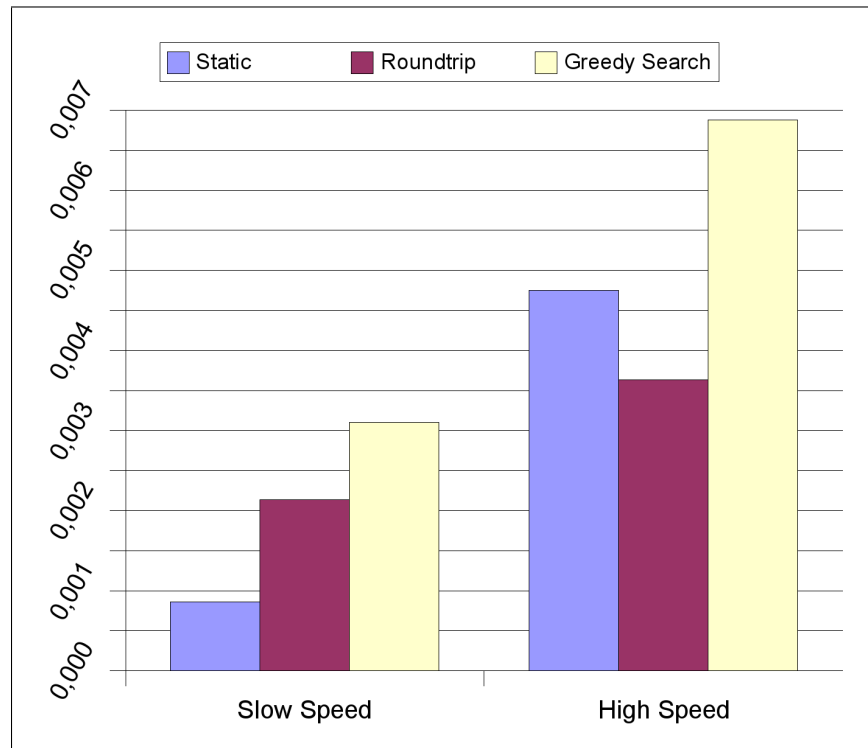


Figure 6.9: Results of different methods for searching an unaware intruder. Left part shows the probabilities for three different methods to catch a slow moving intruder in one timestep. Right part shows the probabilities for a faster intruder.

be weak or unrealistic, this would not be reflected in the performance measurement used for the method. The correctness or quality of the underlying movement model can not be measured by the evaluation function presented here.

As the reader will have noticed, the greedy search algorithm is strongly correlated to the multirobot exploration algorithm presented in chapter 2. In fact, the algorithm can be seen as a form of multirobot exploration algorithm with an increasing information gain for places, which are not in the robot's sensor range. As we have proved, this approach leads to good results and allows a stable and adequate solution for the problem. However, there is

a fundamental difference between multi robot exploration and multi robot intruder searching: In case of the multi robot exploration, the planning has to be online, since the information collected by the sensors is an integral part of the next planning step. In our formulation of the multi robot intruder searching problem, the situation was different, we expect all measurement to be negative <sup>4</sup>, since a positive measurement would terminate the search. This was also reflected in the formulae presented in section 6.2. We argued the greedy one-step solution, since an optimal solution for the problem is almost intractable. An interesting idea might be to increase the planning horizon to more steps and to compare the performance of different levels, therefore trading calculation time for system performance. It would also be important to know a possible limit for the increase in calculation time, since the nondeterministic robot movement would only allow a limited time horizon to be adequately planned.

---

<sup>4</sup>Meaning that no intruder was found in the measurement



# Chapter 7

## Perspectives

### 7.1 Conclusions

In this manuscript, we have shown methods and techniques for searching a human intruder in a closed, 2-dimensional area with a group of robots.

The first step in our solution was to transfer the problem from searching an unknown environment to the task of searching a known environment. This can be done by carrying out a complete exploration prior to the search task itself. Since our work deals with groups of robots, we presented a coordination scheme to speedup the process.

To allow the construction of complex search methods, we developed human motion models with different levels of awareness and cooperation. Our main focus was set on two different models: The first model corresponded to a person unaware of pursuit and therefore without reaction to the searching robots. The effective implementation of this model required the development of advanced Markov Models, which allowed the estimation of a probability distribution of the intruder's location in the environment. The second model corresponded to a person with the intention of hiding from the pursuers and which had perfect intelligence of the pursuers' future motions.

Based upon these models, we proposed metrics to evaluate and compare different search techniques. As a basis for comparison, we shortly discussed some simple search techniques and their performances.

To catch a perfect intelligent intruder with the intention of hiding, we

presented a technique to project the problem on a graph structure, which was then searched by a highly optimized A\* Planner. The evaluation of the technique was done through simulation as with real world experiments and showed it to be practical as well as having an excellent performance. To complete the aspect of perfect intelligent intruders, we also discussed an alternative planning space for the A\* Planner, which permitted the effective planning in very large environments.

To catch an unaware intruder, we presented a greedy technique, which was based on the intruder models developed before. We suggested an approach based on the calculation of utility and costs and included a simple coordination scheme to generate an efficient search pattern. In experimental comparison the method has shown better results than simpler search mechanisms.

Summing up the above, we build a framework to construct a complete searching system, starting from exploration of the environment to the generation of search plans and the estimation of the intruder's whereabouts. The framework is suitable as a step by step manual how to build such a system and what results can be obtained in a specific environment, given an assumed type of intruder.

## 7.2 Future Work

The main focus of this work has been planning and coordination of a group of pursuing robots to locate an intruder. While we think that our solution is stable and delivers fast and usable results for the case of perfectly smart intruders, the question how smart real intruders (humans) behave is still open. In scenarios with a small number of searching robots compared to the complexity of the environment, the whole system depends on the quality of the human model. A smart intruder with knowledge about the used model could also use this knowledge to develop strategies to improve evasion. It is therefore necessary to come up with better human models and to include mechanism that incorporate and counter active evasion.

While being practical in almost all cases tested, the A\* planner discussed in chapter 5 required two optimizations, which sacrificed the optimality of



the generated plan. An important step to improve the system should be the development of an admissible heuristic which is still efficient enough to let the search terminate in reasonable time. For the second optimization, we have shown that the exclusion of recontamination does not always hold for our formulation of the problem, as long as we forbid the jumping of a robot from one node to non adjacent node. An interesting question would be, how much recontamination is needed for our case to regain its optimality and how this can be included in the planner.

Another important step for the development for a real (and perhaps commercial) system is the availability of cheap sensors and robots. In our experiments we used commercial off-the-shelf robots with a price around 3000€ per robot. The sensor used was mainly the commonly found SICK-LMS-200 Laser Scanner, which price is usually around 4000 € per piece. Since every robot needs two of them, a moderate sized team would alone cost almost 100.000 €. At the beginning of the manuscript, we motivated the use of autonomous security robots among other things with the argument of a cheap price compared to human labor. This is clearly not the case at this time. While we think that commercial laser scanners and robot platforms will become much cheaper in the future, it is difficult to tell when this will happen. However, cheaper sensors with different characteristics are available, for example cameras. While we did our experiments with laser range finders, the methods and algorithm are not based on specific characteristics of a laser, our only assumption was a sensor with omnidirectional view. With the argument of price in mind, an omnidirectional camera or a cluster of cheap cameras with an omnidirectional field of view, could provide a cheap alternative to the laser.

Another interesting direction of research could be the use of flying robots, which offer the advantage of high speed and maneuverability. This is especially interesting, since flying robots, for example quadcopters, can move significantly faster than a human intruder. Naturally, this calls for an outdoor scenario and flying robots with fast collision free navigation are still at the start of their development. While the planning problem can still be treated as a 2-dimensional problem, the sensor has to be 3-dimensional, therefore requiring an alternative to the 2-dimensional lasers.



# Chapter 8

## Appendix

**Theorem 1.** Consider a set of  $n$  positive numbers  $(p_1, \dots, p_n)$ , which add up to a given number  $X$  ( $X = \sum_{j=0}^n p_j$ ). The product of all  $p_j$  gets maximum if all numbers are set to  $\frac{X}{n}$ .

*Proof.* We prove by contradiction. Let us assume we have found a set, which holds the above restrictions but has at least one  $p_j \neq \frac{X}{n}$ . Following from that, that there is at least one pair  $p_h > p_l$ .

Then we set  $\epsilon = \frac{p_h - p_l}{2}$ . Since we knew  $p_h > p_l$ , we can also assume that  $\epsilon > 0$ . We set the product

$$Q_{max} = \left( \prod_{j \in \{h,l\}}^n p_j \right) \cdot p_h \cdot p_l$$

Now assume a second set where  $p_h$  is substituted by  $p_h - \epsilon$  and  $p_l$  by  $p_l + \epsilon$ . The sum of the set is still  $X$ . But the product is:

$$\begin{aligned}
Q_{new} &= \prod_{j \setminus \{h,l\}}^n p_j \cdot (p_h - \epsilon) \cdot (p_l + \epsilon) \\
&= \prod_{j \setminus \{h,l\}}^n p_j \cdot (p_h \cdot p_l - \epsilon \cdot p_l + \epsilon \cdot p_h - \epsilon^2) \\
&= \prod_{j \setminus \{h,l\}}^n p_j \cdot (p_h \cdot p_l + \epsilon \cdot (-p_l + p_h - \epsilon)) \\
&= \prod_{j \setminus \{h,l\}}^n p_j \cdot (p_h \cdot p_l + \epsilon \cdot (p_h - p_l - \frac{p_h - p_l}{2})) \\
&= \prod_{j \setminus \{h,l\}}^n p_j \cdot (p_h \cdot p_l + \epsilon \cdot (\frac{p_h - p_l}{2})) \\
&= \prod_{j \setminus \{h,l\}}^n p_j \cdot (p_h \cdot p_l + \epsilon \cdot (\epsilon)) \\
&= \prod_{j \setminus \{h,l\}}^n p_j \cdot (p_h \cdot p_l + \epsilon^2)
\end{aligned}$$

We assumed that  $Q_{max}$  should be maximal:

$$\begin{aligned}
Q_{max} - Q_{new} &\geq 0 \\
\prod_{j \setminus \{h,l\}}^n p_j \cdot p_h \cdot p_l - \prod_{j \setminus \{h,l\}}^n p_j \cdot (p_h \cdot p_l + \epsilon^2) &\geq 0 \\
\prod_{j \setminus \{h,l\}}^n p_j \cdot p_h \cdot p_l - \prod_{j \setminus \{h,l\}}^n p_j \cdot p_h \cdot p_l - \prod_{j \setminus \{h,l\}}^n p_j \epsilon^2 &\geq 0 \\
- \prod_{j \setminus \{h,l\}}^n p_j \epsilon^2 &\geq 0 \\
\prod_{j \setminus \{h,l\}}^n p_j \epsilon^2 &\leq 0
\end{aligned}$$

Since we knew all  $p_j > 0$  and  $\epsilon > 0$ :

$$\begin{aligned}
0 &< \prod_{j \setminus \{h,l\}}^n p_j \epsilon^2 \leq 0 \\
0 &< 0
\end{aligned}$$

Which is clearly wrong. Therefore the theorem is true.  $\square$

# Bibliography

- [AG] SICK AG. LMS200 Laser Meßsystem, Indoor Version, Technische Beschreibung. Technical Manual. [www.sick.de](http://www.sick.de).
- [BBC<sup>+</sup>95] Joachim Buhmann, Wolfram Burgard, Armin B. Cremers, Dieter Fox, Thomas Hofmann, Frank Schneider, Jiannis Strikos, and Sebastian Thrun. The mobile robot RHINO. *AI Magazine*, 16(2):31–38, Summer 1995.
- [BCF<sup>+</sup>98] W. Burgard, Armin B. Cremers, D. Fox, D. Hähnel, G. Lake-meyer, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 1998.
- [BFDW03] F. Bourgault, T. Furukawa, and H.F. Durrant-Whyte. Coordinated decentralized search for a lost target in a bayesian world. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 48–53, 2003.
- [BFDW04] F. Bourgault, T. Furukawa, and H.F. Durrant-Whyte. Process model, constraints, and the coordinated search strategy. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2004.
- [BFGK98] Wolfram Burgard, Dieter Fox, Jens-Steffen Gutmann, and Kurt Konolige. An experimental comparison of localization methods. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1998.

- [BMF<sup>+</sup>00] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2000.
- [BMS02] W. Burgard, M. Moors, and F. Schneider. Collaborative exploration of unknown environments with teams of mobile robots. In M. Beetz, J. Hertzberg, M. Ghallab, and M.E. Pollack, editors, *Plan-Based Control of Robotic Agents*, volume 2466 of *Lecture Notes in Computer Science*. Springer Verlag, 2002.
- [BMSS05] W. Burgard, M. Moors, C. Stachniss, and F. Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376–378, 2005.
- [BO99] T. Başar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. SIAM, 1999.
- [Bre67] R. Breisch. An intuitive approach to speleotopology. In *Southwestern Cavers*, volume VI.5, pages 72–78. Southwestern Region of the National Speleological society, 1967.
- [Bun00] Statistisches Bundesamt. *Statistisches Jahrbuch 2000 für die Bundesrepublik Deutschland*. Metzler-Poeschel Verlag, 2000.
- [CN86] W Chin and S Ntafos. Optimum watchman routes. In *SCG '86: Proceedings of the second annual symposium on Computational geometry*, pages 24–33, New York, NY, USA, 1986. ACM.
- [DJMW91] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859–865, 1991.
- [EvP94] T. Edlinger and E. von Puttkamer. Exploration of an indoor-environment by an autonomous mobile robot. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1994.

- [FBKT00] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 2000.
- [GLL<sup>+</sup>99] L.J. Guibas, J.C. Latombe, S.M. LaValle, D. Lin, and R. Motwani. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry and Applications*, 9(5):471–493, 1999.
- [Gmb06] Messe Essen GmbH. Security journal 2006. Fair-Publication, August 2006.
- [GSS93] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. A novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F*, 140(2), 1993.
- [GTG04] B. Gerkey, S. Thrun, and G. Gordon. Clear the building: Pursuit-evasion with teams of robots. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, San Jose, CA, 2004. AAAI.
- [Haj75] O. Hajek. *Pursuit Games: An Introduction to the Theory and Applications of Differential Games of Pursuit and Evasion*, volume 120 of *Mathematics in Science and Engineering*. Academic Press, 1975.
- [HEL00] K. HELSGAUN. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European J. Oper. Res.*, 126(1):106–130, 2000.
- [HKS99] J. Hespanha, H. Kim, and S. Sastry. Multiple-agent probabilistic pursuit-evasion games, 1999.
- [HPS00] J. Hespanha, M. Prandini, and S. Sastry. Probabilistic pursuit-evasion games: A onestep nash approach, 2000.
- [IB96] M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. In *Proc. of European Conference on Computer Vision (ECCV)*, pages 343–356, 1996.

- [Isa65] R. Isaacs. *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley & Sons, New York, 1965.
- [LaP93] Andrea S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, 1993.
- [MHG<sup>+</sup>88] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *J. ACM*, 35(1):18–44, 1988.
- [Mor88] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, Summer 1988.
- [MRS05] M. Moors, T. Röhling, and D. Schulz. A probabilistic approach to coordinated multi-robot indoor surveillance. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.
- [MS06] M. Moors and D. Schulz. Improved markov models for indoor surveillance. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4072–4077, 2006.
- [Nas50] John Nash. *Non-cooperative Games*. PhD thesis, Princeton University, 1950.
- [Nta92] Simeon Ntafos. Watchman routes under limited visibility. *Comput. Geom. Theory Appl.*, 1(3):149–170, 1992.
- [O’R87] Joseph O’Rourke. *Art gallery theorems and algorithms*. Oxford University Press, Inc., 1987.
- [Par76] T.D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D.Lick, editors, *Theory and Applications of Graphs*. Springer Verlag, 1976.



- [Par78] T.D. Parsons. The search number of a connected graph. In *Proceedings of the 9th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 549–554. Utilitas Mathematica Publishing, 1978.
- [RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1995.
- [SAB<sup>+</sup>00] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2000.
- [SBFC01] D. Schulz, W. Burgard, D. Fox, and A.B. Cremers. Tracking multiple moving objects with a mobile robot. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001.
- [SBFC03] D. Schulz, W. Burgard, D. Fox, and A. Cremers. People tracking with mobile robots using sample-based joint probabilistic data association filters, 2003.
- [SY92] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, 1992.
- [TBB<sup>+</sup>98] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schimdt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R.P. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA, 1998.
- [TBB<sup>+</sup>99] S. Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: A second generation mobile tour-guide robot. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 1999.

- [TFBF01] S. Thrun, D. Fox, W. Burgard, and Dellaert. F. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2), 2001.
- [Thr93] Sebastian Thrun. Exploration and model building in mobile robot domains. In *Proceedings of the ICNN-93*, pages 175–180, San Francisco, CA, March 1993. IEEE Neural Network Council.
- [VSK<sup>+</sup>02] R. Vidal, O. Shakernia, J. Kim, D. Shim, and S. Sastry. Probabilistic pursuit-evasion games: Theory, implementation and experimental evaluation. *IEEE Transactions on Robotics and Automation, Special Issue on Multi-Robot Systems*, 2002.
- [Yam98] Brian Yamauchi. Frontier-based exploration using multiple robots. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 47–53, Navy Research Laboratory, Washington, DC 20375-5337, May 1998. Navy Center for Applied Research in Artificial Intelligence.
- [YSA99] B. Yamauchi, A. Schultz, and W. Adams. Integrating exploration and localization for mobile robots. *Adaptive Systems*, 7(2), 1999.