# The Multi-Agent Flood Algorithm as an Autonomous System for Search and Rescue Applications

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades

DOKTOR DER NATURWISSENSCHAFTEN

Dr. rer. nat. -
genehmigte Dissertation
von

M.Sc. Florian Andreas Blatt

2017

Referentin: Prof. Dr.-Ing. Helena Szczerbicka
Korreferent: Prof. Dr.-Ing. Christian Müller-Schloer
Tag der Promotion: 26.10.2017

*Keywords*: multi-agent, search & rescue, communication, cooperation

*Schlagwörter*: Multi-Agenten System, Search & Rescue, Kommunikation, Kooperation

# Abstract

Technology should be used to help and aid mankind. This is especially true in the case of disasters. Potential victims have to be helped as soon as possible to ensure the survival of as many of them as it is achievable. For example after an earthquake the rescue teams have about 48 to 72 hours to find the victims, if it takes them any longer the probability to find victims alive decreases dramatically.

The aftermath of different disasters in the recent years have already seen the use of various robotic helpers. Currently the rescue teams use one or two robots guided by human teleoperators to search for victims or to enter spaces that are too small for human team members to reach or too dangerous for them to enter.

This thesis proposes a new algorithm to control multiple robots autonomously in urban search and rescue scenarios. The algorithm will be able to manage the movement of multiple autonomous robots in unknown terrain to find possible victims. Besides maintenance work the robots controlled by this method will not need any human input or help. The cooperation between the robots will be achieved by using two different ways of communication, direct and indirect, to allow data transfer between the robots to overcome certain hazards that can hinder direct communication or prevent it completely. Thus the robots will still be able to cooperate, even if only one part of the communication model is available. Another advantage of the proposed algorithm is the ability of the robots to return any time to the head quarter. A robot will return to the base as soon as it has found a victim. An important point made possible this way is that the human rescue team can start to rescue the found victims as soon as the robots share the information about the location of the victim. Thus, a parallel rescue process is made possible. This capability also allows the maintenance of the robots, repair of defective parts or simply allow the possible recharge of the battery.

The introduced algorithm is evaluated by simulating the agents on different test cases to show that the algorithm is a viable way to send autonomous agents into unknown terrain without further human interaction. The agents will also follow the specific constraints required by rescue robots, which in turn means that the algorithm can be used in “real world” scenarios.

# Abstrakt

Technologie soll zum Wohle der Menschheit eingesetzt werden. Dies ist im Besonderen der Fall wenn es um den Schutz und die Rettung von Menschenleben geht. Zum Beispiel nach einem Erdbeben ist es wichtig die verschütteten Opfer so schnell wie möglich zu finden und zu bergen. Hierbei sind die ersten 48 bis 72 Stunden entscheidend, da danach die Chance rapide sinkt noch Überlebende in den Trümmern zu finden.

In den letzten Jahren wurden bei diversen Einsätzen immer wieder Roboter eingesetzt um das Rettungsteam zu unterstützen. Dabei wurden ein oder auch zwei Roboter eingesetzt, die ferngesteuert nach Opfern gesucht oder die Räume erkundet haben, die für einen Menschen nicht erreichbar oder zu gefährlich waren.

Diese Dissertation stellt einen neuen Algorithmus vor, der dazu benutzt werden kann multiple Roboter autonom in urbanen Rettungsszenarien einzusetzen. Mit Hilfe dieser Methode können mehrere Roboter unbekanntes Gebiet erkunden und dort mögliche Opfer aufspüren. Der Vorteil ist, dass die Roboter keinerlei Eingreifen von menschlichen Teammitgliedern benötigen. Die Kooperieren der Roboter untereinander wird mit Hilfe von zwei Kommunikationswegen ermöglicht: Indirekt und Direkt. Dadurch werden mögliche Einschränkungen der Direkten Kommunikation umgangen, so dass die Roboter trotzdem in der Lage sind ihre Aufgabe zu erfüllen. Ein weiterer Vorteil dieses Algorithmus ist es, dass die Roboter jederzeit zur Basis zurückkehren können. Sollte ein Roboter ein Opfer gefunden haben, so wird diese Möglichkeit genutzt und ein Rettungsteam kann parallel zum laufenden Suchvorgang der Roboter starten und das Opfer bergen. Die Rückkehr zur Basis ermöglicht auch die Wartung und Aufladung der Roboter, wodurch diese längere Zeit im Betrieb sein können.

Der hier vorgestellte Algorithmus wird anhand einer Simulation evaluiert. Dafür werden die Agenten auf verschiedenen Karten simuliert. Dabei unterliegen die Agenten gewissen Einschränkungen, die für diese Art von Roboter nötig sind. Durch Einhaltung dieser Einschränkungen kann der Algorithmus auch leicht für reale Roboter adaptiert werden.

## Acknowledgements

I would like to thank Professor Helena Szczerbicka for her continuous support and supervision during my work on my thesis. Without her guidance, suggestions, and comments my whole research would not have been possible.

Further I would also like to thank all of my coworkers, who also helped me a lot with their advice and their ideas.

Last but not least, I would like to thank my parents and my brothers for their support and their encouragement.

# Contents

# List of Figures

# List of Algorithms

Chapter 1.

# Introduction

Time is of essence in the aftermath of disaster. Helpers have to arrive as quickly as possible to find victims after an earthquake, to help buried miners, or simply to avert any additional damage. Especially after an earthquake the rescue team has to find the victims as soon as possible, as the chance to find them alive decreases rapidly in the first 48 to 72 hours [Murphy et al. (2008); Murphy (2014)]. Current and new technologies offer various advantages that can be used to help, aid, and support search and rescue efforts in different ways.

Different rescue missions have shown that a robot guided by members of the human rescue team can be used to aid in the search for victims. Most of the time these mechanical helpers are used to scout ahead and to enter areas that are not safe for a human rescue team to enter. Various rescue robots exist already, either as science projects or in commercial applications. Some of these have already been used at different disaster sites around the globe. But all of them have in common that they need to be controlled by a human. In addition only one or maybe two robots are used at the same time, not really realizing the advantages of the synergies that a team of robots could bring.

This work will introduce a short overview of existing robots and in turn will take a look at existing algorithms which can be used to control multiple robots, thus enabling the robots to work as an autonomous team with the same goal. With the use of communication and cooperation the robots will be able to support the human rescue team, searching for victims after a disaster without any input from human teleoperators.

After taking a look at the various advantages and disadvantages offered by existing algorithms, a new algorithm is proposed. This algorithm will be able to send robots autonomously into unknown terrain, allow them to find victims, and to send the information about said victims back to the human rescue team in parallel to the ongoing robotic search process. This allows the human team to start the rescue process as quickly as possible, increasing the chance to find victims alive. The proposed algorithm will be evaluated by simulating the different agents to show that it is a viable approach. It will also adhere to the specific constraints required by rescue robots, thus creating an approach that can be used in “real world” scenarios.

## 1.1. A short Historical Overview about Rescue Robotics

The first robots used to help after a disaster were big and slow machines, sent into action just some months after the Three Mile Island nuclear incident in 1979 and a few years later again after the Chernobyl nuclear catastrophe in 1986 (Murphy, 2014, p. 3). These robots were heavily shielded to survive exposure to radiation and required multiple human operators.

Additional motivation for research on dedicated search and rescue robots was sparked by the big earthquake in Kobe and Osaka in 1995 and the bombing of the Alfred P. Murrah Federal Building in Oklahoma City in the same year [Davids (2002); Murphy et al. (2008); Murphy (2014)]

The first example of a search and rescue operation using dedicated robots was the morning after the attack on the twin towers in New York on the September 11th, 2001 [Davids (2002); Murphy et al. (2008); Murphy (2014)]. Five different robots were used to support the human search and rescue teams searching remains of the destroyed twin towers. These robots helped to find more than ten victims, although one got stuck and another one lost the wireless connection to the teleoperator (Murphy, 2014, p. 44).

The following years saw an increased application of rescue robots at various disaster and emergency sites. These included searching for victims after an earthquake, trying to find victims in the rubble of collapsed buildings, or trying to find a missing miner after a mine collapse, c.f. (Murphy, 2014, p. 25–28). Another well-known use of search and rescue robots was after another nuclear disaster at the Fukushima Daiichi Nuclear Power Station in 2011 [Nagatani et al. (2011); Murphy (2014)]. The robot that was used in this scenario was named *Quince* [Rohmer et al. (2010)].

Compared to the first robots that were used in ’79 and ’86 at the previous nuclear catastrophes, this robot is rather small and was especially created for urban search and rescue missions. Before it was used, the base variant of the robot was adapted to allow the collection of a sample of the contaminated water in the reactor room and to be able to install a water-gauge in the basement of the reactor building (Nagatani et al., 2011, p. 14).

Comparing these newer robots to their ancestors, two things always remain the same: first every one of these robots needs a human to control and steer it. Second, none of these robots acts in a team with other robots. Although each robot is well designed to handle their specific task, they are not equipped with enough “intelligence” to act as a team or to act autonomously. The ability to work as a team requires some specific algorithms to control the communication and cooperation between multiple robots, especially if the teamwork should happen without any human interference. Some ideas on how this could work were already proposed in Koenig et al. (2001) for example. In this example multiple robots are able to communicate by leaving chemical markings in the environment and thus all the robots act as a team. More recent examples, such as Ferranti et al. (2007), showed how communication between robots can be expanded to increase the search speed of the robot team.

Another example is the RoboCup, Kitano et al. (1998), which started out as a soccer competition between real or simulated robots to foster research in the field on intelligent multi-agent algorithms for the control of multiple robots. This concept was later expanded to the RoboCup Rescue, Kitano et al. (1999); Kitano and Tadokoro (2001), based on the experience gained after the earthquake in Kobe and Osaka in ’95. This way the insights gained from the RoboCup competition could be applied to a real world scenario and it also allowed the use of different features, for example reconnaissance of unknown terrain or manage relief units that were not applicable to the first scenario.

This thesis extends the current possibilities by investigating a concept for the use of multiple autonomous robots at the same time, which will act as a team. In turn their cooperation will speed up the search process and thus will cut down the time required to find as many victims as possible in an urban search and rescue scenario. The algorithm proposed in this work will adhere to the three design principles for rescue robotics stated by Murphy (2014). Robots must:

1. operate in extreme, even unknown, terrain and operating conditions
2. have the ability to function in GPS- and wireless-denied environments
3. provide appropriate human-robot interaction

Especially the second principle has to be kept in mind in case of a search and rescue scenario, because an algorithm that uses multiple robots needs a robust way to let them communicate and share data with each other. The method put forward in this work is based on indirect communication and only uses direct communication as an additional way to exchange data. If the surrounding terrain does not allow the use of radio signals for a direct communication, then the algorithm is still able to function, although not as fast. This approach leverages the advantages of a multi-agent system, even if only indirect transmission is possible through the use of graceful degradation of the communication channel.

## 1.2. Thesis outline

The reminder of this thesis is structured in the following way: Chapter 2 contains a survey of the current state of the art. We will take a short look at some existing rescue robots and survey different multi-agent algorithms for search and rescue scenarios. For each algorithm described in this chapter we will briefly discuss its advantages and disadvantages.

The next chapter, Chapter 3, will propose a multi-agent algorithm that tries to address the various disadvantages of the existing methods. This version of the algorithm will use only indirect communication. At the end of the chapter we will evaluate the algorithm and compare it with existing algorithms. The following chapters will contain improvements to this base algorithm, where each optimization is a step towards a realistic implementation.

Chapter 4 introduces an improvement of the algorithm proposed in Chapter 3. We will suggest concepts showing how to combine indirect and direct communication and how graceful degradation of the communication channel is possible. The results of these improvements will be evaluated and compared with the original version.

In Chapter 5, we will recommend a way to remove the grid abstraction made in the original version of the multi-agent algorithm proposed in Chapter 3.

This way the algorithm is easier to adapt to real world robots. Furthermore an improvement of the behavior will be proposed that adds more sophisticated capabilities to the agents, allowing them to explore the unknown terrain in a better way.

The last chapter of this thesis, Chapter 6, encloses a summary of the whole work and look at future work, based on the different proposals made in this thesis.

## 1.3. Publications

Parts of this work have been presented and published in the proceedings of the following conferences:

- Matthias Becker, Florian Blatt and Helena Szczerbicka: A Multi-agent Flooding Algorithm for Search and Rescue Operations in Unknown Terrain, *German Conference on Multiagent System Technologies (MATES)*, pages 19–28, 2013
- Matthias Becker, Florian Blatt and Helena Szczerbicka: A Concept of Layered Robust Communication between Robots in Multi-agent Search & Rescue Scenarios. *IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 175–180, 2014
- Florian Blatt, Matthias Becker and Helena Szczerbicka: Optimizing the exploration efficiency of autonomous search and rescue agents using a concept of layered robust communication. *IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*,pages 1–6, 2015
- Florian Blatt and Helena Szczerbicka: Realisation of Navigation Concepts for the Multi-agent Flood Algorithm for Search & Rescue Scenarios Using RFID Tags. *IEEE/ACM 20th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 112–115, 2016
- Florian Blatt and Helena Szczerbicka: Combining the Multi-Agent Flood Algorithm with Frontier-Based Exploration in Search & Rescue Applications. *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, 2017

CHAPTER 2.

# STATE OF THE ART

This Chapter will give a short introduction of currently used rescue robots and current concepts for multi-agent algorithms for search and rescue scenarios. All of the robots described in this chapter are research robots, only one of them has been used in a disaster at this time. The same caveat applies to the various multi-agent algorithms in the second part of this chapter, as not one of those was yet used for a real search and rescue scenario either.

## 2.1. Rescue Robots

This section will showcase different rescue robots that are currently in use. For each robot there will be a short description given and a short discussion about its advantages and disadvantages.

This work will discuss only unmanned ground vehicles (UGV). Due to specific design principles, for example the locomotion of the different robots, maritime disaster robots and unmanned air vehicles are not able to enter collapsed buildings or small cracks.

### 2.1.1. IUB Rugbot

The IUB Rugbot was introduced in 2006 by the International University Bremen and created by Birk et al. (2006b).

The first version of the Rugbot was used in the RoboCup competition in 2005 in Osaka. The design of the robot is based on the CubeSystem, a collection of hardware and software components which allows for fast robot prototyping, Birk (2004). The CubeSystem offers controller hardware, a special operating system and libraries for common robotic tasks, c.f. Birk et al. (2006b).

### Hardware

The Rugbot uses tracks and is a relatively small vehicle. It has a footprint of roughly $50 \times 50$ cm$^2$ and weighs about 35 kg. In addition to the tracks the Rugbot also has a “flipper mechanism” available, which is a movable arm, that allows the robot to navigate over rubble and even enables the robot to move up stairs.

The standard sensor suite of the robot includes a laser rangefinder, an ultrasound sensor, four cameras, one thermo camera and rate gyroscopes (a type of gyroscope that indicates the rate of change of angle over time). Additional sensors can be added if they are needed.

Each Rugbot also has its own onboard PC available which offers sufficient computational power to control the movement of the robot.

The onboard batteries allow for two to three hours of continuous operation, this also includes the movement through rough terrain.

### Software

The CubeSystem software used to operate the robot is designed to provide the realtime functionalities, required to compute the path and in turn steer the motor of the robot. Higher “intelligence”, from teleoperation to full autonomous behaviour, is supported through the onboard PC.

The onboard software is able to detect humans from the input of the various sensors and to create maps without the need for connections to additional servers [Birk et al. (2006b,a)].

### Advantages and Disadvantages

The advantages of the Rugbot are, that it is able to navigate through an obstacle course and find human victims without the help of a human operator. The addition of the flipper mechanism allows the tracked robot to traverse stairs, which are usually something that robots can not handle very well.

The main disadvantage of this robot is, that it is designed for the operation as a single entity without any other possible input from other team members. A radio network communication between multiple robots would be possible but the current software lacks the capabilities that are needed for a full cooperation between these robots.

### 2.1.2. Quince

Quince is another small robot that was conceived as a highly maneuverable and modular platform for robot rescue research and development [Rohmer et al. (2010)].

#### Hardware

The original concept of Quince is based on “modularity, interoperability and customization to simplify the integration of the different robotic technologies” (Rohmer et al., 2010, p. 1). This robot was built to especially handle uneven terrain, stairs, and rubble. It can achieve this kind of mobility by combining a main track with four smaller sub-tracks. The main track encloses the whole robot, removing more or less the possibility of getting the robot stuck on a part of the chassis without the opportunity to apply any friction. The sub-tracks, which are adjustable and are also called flippers, add further gripping power and allow Quince to traverse stairs. The complete chassis has a length of 71 cm and a width of 48 cm, with a weight of 27 kg.

In general the center of gravity of the robot is kept low, as everything is kept inside of its body.

The basic sensor layout of Quince includes two Position Sensitive Devices (PSD, which are able to measure the one- or two-dimensional position of a light point), an Inertial Measurement Unit (IMU), a current sensor and an encoder for each of the six motors, and three cameras. Additional sensors can be installed rather easily.

#### Software

Quince’s operating system is a highly customized Linux version. The robot only supports the minimal operations that are needed for movement and for teleoperability. Additional functionability would have to be added on a per use basis. Remote control is established using a standard wireless IP network.

#### Advantages and Disadvantages

The design goals for this robot were to create a small, maneuverable, and highly customizable machine. The only problem is, that the basic variant of the robot is dependent on a human controller, who can pilot it as long as a wireless connection is available.

#### The Redesign of Quince

The first big mission where Quince was used, was after the nuclear accident at the Fukushima Daiichi Nuclear Power Station on March 2011 [Nagatani et al. (2011); Murphy (2014)].

A high priority requirement for this mission was the hardware reliability and the communication reliability, due to the massive radiation in the target zone. The first part means, that the electronic components of the robot had to withstand the exposure to gamma rays. As Quince is a teleoperated robot it also needed a communication line to the teleoperator, but wireless communication may be blocked in the nuclear reactor. As a consequence non-wireless communication was a requirement for this mission, in this case a cable and tether were used.

### 2.1.3. ICARUS robots

The European ICARUS project tries to develop integrated components to assist search and rescue teams after an urban or maritime disaster, De Cubber et al. (2013a).

For example, the Belgian First Aid and Support Team (B-FAST) needed two different kind of robots to help in urban search and rescue scenarios, see De Cubber et al. (2013b):

- A larger robot, which can be used as a mobile base and sensor platform. It should be able to broadcast the data that it collects to the field operators. Another very important requirement is, that the robot is able to traverse rugged terrain.
- The second robot should be small and thus be able to enter collapsed buildings to search for buried victims

#### Hardware

The larger variant of the robots should be able to act autonomously, but still be able to act on human intervention, and collect data about the surrounding area by itself. It should also act as a carrier for the smaller UGV.

To support these three requirements, the large robot is powered by a combustion engine and moved by a chain-drive. The planned sensors for this platform should include a GPS receiver, a panning laser, bumper sensors, a stereo camera system, a time-of-flight camera for terrain traversability analysis, and victim detection sensors. The manipulator is designed to lift up a weight up to 250kg. An additional camera on the robot provides image data for teleoperation. As stated by De Cubber et al. (2013b) the whole concept is still in design.

The small UGV, that is carried by the larger one as a support unit, is also based on a tracked chassis and possesses a smaller manipulator. It is mainly designed to help the larger UGV, as it is able to enter smaller enclosed spaces and may find victims in places that can not be reached by the bigger robot. It also has a camera that forwards the pictures through the bigger robot to the teleoperator.

#### Software

Another aim of the ICARUS project is to develop robot-independent monitoring and control capabilities. This should allow the rescue team to tie a heterogeneous team of robots together and in turn optimize the flow of the data and information from the various robots to the team.

For this to work the whole robot team has to have access to a working communication network between themselves and the human team members. The ICARUS project proposes a multi-level network infrastructure, that should offer a reliable communication channel for all affiliated members. This infrastructure contains mobile and wire-less ad-hoc communications, mixed in with line-of-sight communication possibilities. Additionally the robots should be able to adapt the different radio signal settings to maximise the possible connectivity and data transfer, as proposed in De Cubber et al. (2013a).

#### Advantages and Disadvantages

The ICARUS project offers a whole range of tools for the use in search and rescue scenarios. These software and hardware tools should all be able to interact together to create a coherent search and rescue team, consisting of human controllers and robotic search units. Although the proposed larger unmanned ground vehicle may not be usable in all scenarios, depending on the designed size of the chassis.

### 2.1.4. Hector robots

The Heterogeneous Cooperating Team of Robots, or Hector in short, is a RoboCup Rescue league team, that participated since 2009 in the competition [Kohlbrecher et al. (2015)].

The team consists of three different types of unmanned ground vehicles:

- A tracked robot with flippers, similar in design to the aforementioned robot Quince
- A tracked robot without flippers
- A wheeled robot

#### Hardware

As mentioned above two of the robots are using tracks. One of those is also using additional flippers for more mobility in rugged terrain. The third robot is based on a four wheeled chassis.

Each robot is divided into two parts, the chassis and the autonomous box, which enables the robot to explore autonomously and victim detection. The sensors a also contained in this box and consist of a laser scanner, an ultrasound range finder, an inertial measurement unit, a RGB-D camera, and a thermal camera.

The different robots communicate between themselves using a standard 2.4 GHz 802.11g/n network.

#### Software

The software used in the different robots of Hector is based on the Robot Operating System (ROS). Usually the robots will act autonomously according to a specific mission statement. If this algorithm is not sufficient, each robot can be teleoperated by a human controller.

The autonomous box of each robot also allows for Simultaneous Localization And Mapping (SLAM), c.f. Durrant-Whyte and Bailey (2006), which uses the sensor input and produces a map for the use of navigation, while trying to keep track of the location of the robot in this map. Two team members can then exchange the map data and merge the different versions together to create a more complete map from the two partial maps.

Advantages and Disadvantages

The Hector robots are able to explore the disaster scenario autonomously and act according to a specific mission statement. The main point of interest of the inventors of team Hector was to create a cooperating team of robots that are able to work without human supervision.

### 2.1.5. Snake robots

The robots introduced above have either a tracked or a wheeled chassis. This allows for fast movement on flat terrain and enables them to traverse some rubble. The drawback of this type of locomotion is that a minimum size of the robot is needed to create sufficient traction. Otherwise the tracks or the wheels are too small to efficiently move over rugged terrain.

Another approach for a different type of locomotion is the snake robot. This UGV mimics the movement pattern of a snake by connecting small segments together. There are different types of locomotion for snake robots: tracked segments [Ito and Maruyama (2016)] or servo motors, that try to mimic snake like shifting [Crespi et al. (2005); Wright et al. (2007)].

Each segment has its own motor to control its movement. This design allows for a smaller robot that is able to crawl into cracks that maybe to small for normal tracked and/or wheeled robots. The snake like movement also enables the robot to enter terrain that would be unreachable for the other types of locomotion.

For example, a size comparison between the crawler introduced by Ito and Maruyama (2016) and Quince, Rohmer et al. (2010), shows that the snake robot has a length, width, and height of 103 cm $\times$ 20 cm $\times$ 13 cm, while Quince has a size of 111 cm $\times$ 42 cm $\times$ 21 cm. Comparing the weight shows that the crawler weights only 8 kg and Quince weights almost three and a half times as much with 27 kg.

Advantages and Disadvantages

The main advantage of this type of robot is the ability to get into smaller spaces and to reach areas, that may be to small for the bigger variants. The smaller size of the chassis also creates some drawbacks. A smaller body means, that the room for sensors is minimal, as each segment has to house its motor and its own battery. Adding additional computer power and sensors for example to the crawler proposed in Ito and Maruyama (2016) would be simple, but this in turn would change the size of the robot and thus may compromise the ability to enter cracks and small holes in the rubble.

### 2.1.6. Summary

This section gave a short overview about different existing rescue robots. Almost all of these robots are using tracks for locomotion, usually combined with some sort of arm to allow the robot to lever itself up. This enables the robot to climb over rubble or traverse stairs. Only one of the robots used wheels, the older variant of one of the Hector robots, and the chassis of the last one is based on another concept altogether, imitating the movement of a snake.

The sensor suite of each robot is almost identical and usually expandable. This suite includes one or more cameras (normal and/or heat vision), a gyroscope, a laser range finder, and the sensors needed to keep track of the locomotion. Only the snake robot deviates from this, as the chassis does not allow for a lot of sensors.

Usually each robot can communicate either with other robots or with the human team using a typical WiFi connection. Although, if it is required, the robots can also use a tether with a cable for the communication with the human team. Inter robot communication does only exist between the ICARUS and Hector team members. The other robots are usually sent out on their own.

All of the robots mentioned here are devised as teleoperated robots, needing the input of a human team member to control them. Only the Hector team offers the possibility for a completely autonomous movement.

## 2.2. Multi-Agent Search & Rescue Algorithms

The previous sections discussed a selection of existing rescue robots or concepts and ideas for the design of new ones. This section contains a short survey over different algorithms that will try to use multiple robots cooperating together to find human victims after an urban disaster in a search and rescue scenario.

The main idea behind this concept is very simple: multiple robots will try to explore the unknown terrain of the scenario and during the exploration will look out for victims (or points of interest in general). To speed the whole process up and to take advantage of the use of multiple robots, these entities will try to communicate between themselves and thus exchange collected data about the already explored terrain. Each robot should be able to act autonomously according to the main goal, while trying to share its collected data with others, helping them and removing the need for them to collect this data themselves. Each of these robots can also be called an agent and a multitude of these agents form a multi-agent system.

The conventional definition of a multi-agent system is offered by Michael Wooldridge, (Wooldridge, 2009, p. 5):

> “An agent is a computer system that is capable of *independent* action on behalf of its user or owner. In other words, an agent can figure out for itself what it needs to do in order to satisfy its design objectives, rather than having to be told explicitly what to do at any given moment. A multiagent system is one that consists of a number of agents, which *interact* with one another, typically by exchanging messages through some computer network infrastructure. In the most general case, the agents in a multiagent system will be representing or acting on behalf of users or owners with very different goals and motivations. In order to successfully interact, these agents will thus require the ability to *cooperate*, *coordinate*, and *negotiate* with other people in our everyday lives.”

Not everything of this definition is applicable for search and rescue scenarios. Usually the agents will cooperate, as the goal of each agent is the same. Also each agent will have the same owner, further diminishing the chance for any sort of conflict between two agents. As mentioned earlier in a search and rescue scenario a multi-agent system simply consists of multiple agents that will work together using communication and cooperation to search for as many (or all) human victims as possible. However some specific constraints apply for a search and rescue scenario that each multi-agent system has to respect and handle. The agents have little to no knowledge about the terrain, as already existing maps may be invalidated by an earthquake. Additionally the agents can not rely on a global positioning system, thus their exact location is more or less unknown. But the most crucial restraint, which affects a multi-agent system the most is the constraint on the use of direct communication. As already mentioned previously, the robots used at the World Trade Center were not able to communicate via radio signals with the human team members. Also Quince could not rely on radio signals as it entered the reactor core of the Fukushima Daiichi Power Station. As a consequence each algorithm has to find a way to still be able to communicate despite these restraints.

The main research effort in this direction in the last years was concentrated on the RoboCup Rescue Simulation [Kitano et al. (1999); Kitano and Tadokoro (2001)], which is a comprehensive simulation environment for research in disaster response management. This project is a competition operated by the RoboCup, best known for the Robot Soccer World Cup competition, see Kitano et al. (1998), and the RoboCup Rescue competition. The difference between the Rescue and the Rescue Simulation projects is that the first one is for real robots, while the latter one is intended for multi-agent simulations[1]. The main goal of the rescue simulation is to evaluate the effectiveness of rescue team agents in a city scenario, for example fire brigades extinguishing fires or ambulances rescuing civilians after an earthquake happened. The simulator offered by the project and the algorithms used in the project aim at a greater scope. For this reason other multi-agent algorithms were considered and the following subsections will showcase a selection of these multi-agent algorithms for search and rescue scenarios.

[1]see also http://roborescue.sourceforge.net

### 2.2.1. Ants

The first ideas to borrow from biology to create a multi-agent system was described by Marco Dorigo and others [Dorigo et al. (1996); Dorigo and Gambardella (1997)]. They proposed to use agents similar to ants as a new optimization method and applied it to find solutions for the Travelling Salesman Problem [Jünger et al. (1995)].

Sven Koenig and Yaxin Liu proposed to use the idea behind this concept to apply it to robots [Koenig and Liu (2001)]. Instead of creating intelligent agents, they created multiple ant agents that explore the terrain and leave markings in it to communicate with other robots. This kind of indirect communication via markings in the environment is called *stigmergy*.

The term was devised by the zoologist Grassé in 1959 and he defined it as a means of communication through modification of the environment [Grassé (1959); Marsh and Onof (2008)]. Ants use a trail of pheromones to navigate towards food or back towards the hive.

The first advantage of the approach by Koenig et al. is that the multiple ant robots offer fault tolerance, as one failing robot will not hinder the exploration of the other robots. The second advantage is, that multiple agents offer parallelism, meaning a group of agents will usually explore faster than a single agent. Another asset would be, that one single robot does not have to possess a lot of computational power or a lot of sophisticated sensors.

#### The algorithm

Koenig et al. modelled the terrain as a directed graph, Koenig and Liu (2001). This can be done by simply overlaying the terrain with a regular grid. Figure 2.1 shows how a grid can be seen as a graph, where each cell in the grid is represented by a vertex and the edges are the connections to the neighboring cells.

As the algorithm is based on stigmergy each ant robot needs to be able to mark the environment, representing a pheromone. These markings are stored in a value called $u$ in each vertex $s \in S$: $u(s)$. This value represents the strength of the pheromone marking and starts at 0, stating in this case that no marking is present. The robots can read the markings stored in the neighboring vertices surrounding their current position and will only decide

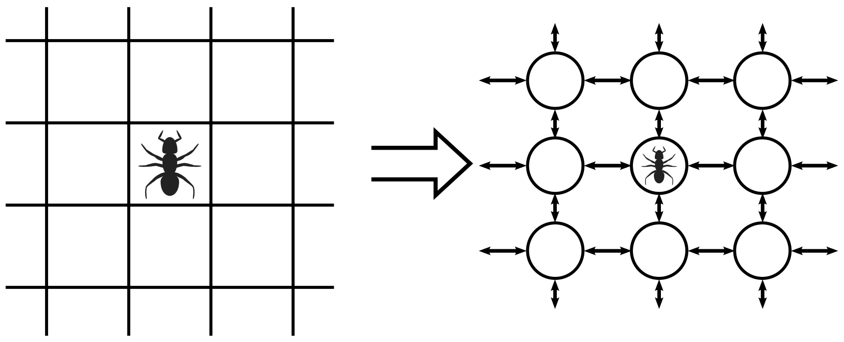

Figure 2.1.: Modelling a grid as a graph

their next movement based on this value alone, by choosing the one with the smallest $u$-value. Before leaving a vertex an agent is able to change the corresponding $u$-value, depending on the used update rule (the base rule is to simply count the nodes or vertices: $u(s) := 1 + u(s)$).

Feasibility

Jonas Svennebring and Sven Koenig did a feasibility study of this algorithm, Svennebring and Koenig (2004), showing that the algorithm is usable with real robots instead of a simulation. The markings are realised using drops of chemical substances. This experiment illustrated the robustness of this approach despite the limited hardware and software used, even if the technique used for the markings does not allow for erasing them again.

### 2.2.2. Brick&Mortar

The second multi-agent algorithm is called Brick&Mortar (or simply B&M) and was created by Ettore Ferranti and others in 2007, Ferranti et al. (2007). This algorithm is an optimization of the Ant algorithm from the previous subsection. The drawbacks of this algorithm are that agents do not know, if they have finished the exploration and that a lot of agents will stay in already explored terrain. Thus, the advantage of using multiple agents is degraded.

Model of the Algorithm

The base model for this algorithm is the same as in the Ants algorithm. The terrain gets divided into square cells. Each cell is either traversable or it is not. A non-traversable cell is considered a wall. A traversable cell can either be empty, contain a victim or another agent. The markings of each cell can be either one of the following, taken from (Ferranti et al., 2007, p. 2):

**Wall:** This is a non-traversable cell

**Unexplored:** This cell has not been visited yet

**Explored:** This cell was already visited once, but the agents may need to go through it again to reach other *unexplored* cells

**Visited:** This is also an explored cell, but the agents know that they do not need to enter it again. The agents treat these cells like a *wall*

Each agent is able to move from its current cell into one of the four adjacent cells in the North, South, West, or East. This pattern of surrounding cells is also called the Von Neumann neighborhood [Von Neumann and Burks (1966)], as seen in Figure 2.2.

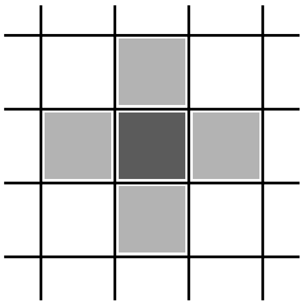

Figure 2.2.: Von Neumann neighborhood, as seen from the dark gray center

The only possible way for the different agents to communicate between each other, and in the end to create a cooperation between themselves is to use the markings of the cells. The authors assume that the building already possesses some uniformly distributed miniature devices (for example RFID chips), which are able to store a small amount of information to store the markings described above, and that the robots can deposit additional devices

**Algorithm 1** Brick&Mortar Without Loop Handling

```
1: Marking Step
2: if the current cell is not blocking the path between any two explored or
   unexplored cells around then
3:     mark the cell as visited
4: else
5:     mark the cell as explored
6: end if
7: Navigation Step
8: if at least one of the four cells around is unexplored then
9:     for each of the unexplored cells see how many wall or visited cells
       are around it, then go to the cell with highest number of these cells
       surrounding it, which is most likely to be marked as visited in the
       marking step
10: else if at least one of the four cells around is explored then
11:     go to one of them. Avoid selecting the cell where you came from unless it
        is the only candidate. Instead select the first explored cell in an ordered
        list of adjacent cells, e.g. [North, East, South, West] {The order of cells
        in the list depends on the agentID, so that different agents disperse in
        different directions}.
12: else
13:     terminate {All adjacent cells are inaccessible, i.e. visited or wall cells}
14: end if
```

in cells, which are not covered, as they explore the area for the first time. For the model Ferranti et al. presume the abstraction that the terrain is divided into square cells and at least one device exists per cell (Ferranti et al., 2007, p. 2).

How it works

The Brick&Mortar algorithm tries to address the shortcoming of the Ants algorithm, which in this case would be the lack of knowledge when the agents can stop exploring the area and the algorithm ends. Additionally, the algorithm is constructed in the way that one cell will only be traversed at a maximum of two times, further minimizing the amount of time that the agents will spend on already explored terrain.

The main idea behind this algorithm, shown in Algorithm 1 and taken from (Ferranti et al., 2007, p. 5), is to mark as many cells as possible and as fast as possible as *visited*. As a *visited* cell is non-traversable for the agents the

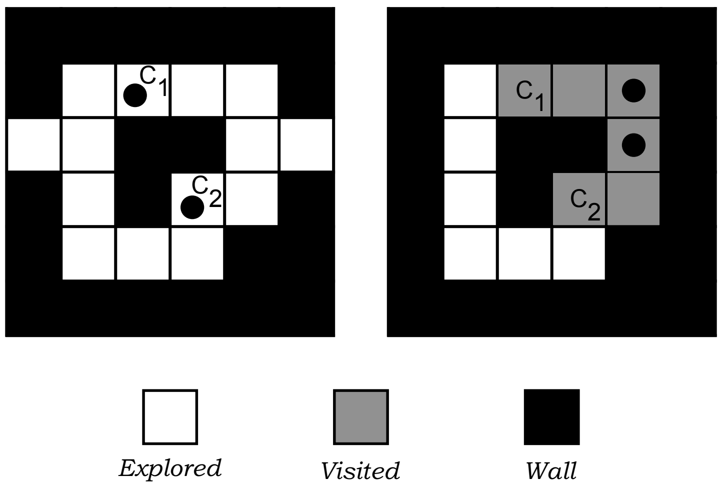

Figure 2.3.: Brick&Mortar loop problem: two agents traverse the same loop, but marking the cells independently. This way an agent can not discern the closure of the loop anymore without help.

amount of traversable cells will be reduced with each former reachable cell that gets marked as *visited*. The agents will steadily increase the number of *visited* cells, while always keeping the cells connected that are still needed for movement, see line 2 in Algorithm 1. This means, that the agents will create corridors of cells that are only marked *explored* (still allowing agents to traverse these cells, which is not possible as soon as a cell is marked as *visited*), which in turn combine all *unexplored* elements of the grid. The primary rule, which all robots must follow is to never set a cell to *visited*, if this will block the way between two traversable cells.

The algorithm described above will work without a problem, as long as the agents will not traverse the same path of *explored* cells multiple times, while being unable to change the status of at least one cell to *visited*. This behavior is called a loop. These loops will occur if there are unconnected walls in the middle of the map. If at least one of these obstacles is present in the scenario it will trap the agents and the algorithm will not terminate. A single agent traversing a loop is able to mark the *explored* cells in a loop without a problem and in this way the loop can be closed. If multiple agents traverse on the same loop agents are not able to close the loop without additional help.

Figure 2.3 depicts an example of the loop problem, (Ferranti et al., 2007, p. 5). The two dots represent the agents, while the white cells in the grid show the already *explored* cells, the black cells represent *walls*. The grey cells in the right picture represent the cells that are marked as *visited*. The two agents $A_1$ and $A_2$ can move around the obstacle in the loop depicted on the left hand side in Figure 2.3, starting at point $C_1$ and $C_2$ respectively. As they move they will mark each visited cell as *explored*. Once they have surrounded the obstacle and are back at their starting points, $C_1$ and $C_2$, they will detect the loop. Using the procedure described in the paragraphs above the agents now would mark each cell as *visited*. The problem will arise as soon as the two agents meet, as shown in the left side of the picture. The surrounding cells are all marked as *visited* and both agents are stuck and can not move anymore.

This is a known problem and other algorithms either dismiss it, as Icking et al. (2005) have done or need a human operator to solve it manually, as in Howard et al. (2006). As human intervention may not be possible at a disaster area, Ettore Ferranti and his coauthors added a loop handling routine to the B&M algorithm, so that this method is able to solve the problem autonomously, c.f. Ferranti et al. (2007).

### Loop Handling

To add the ability to detect and handle loops to the algorithm, the authors made an additional assumption: an agent can mark a cell with its own unique ID (in this case a simple number) and the directions (North, South, West, or East) which the agent enters or leaves the cell. With the help of this new information an agent can now detect a loop as soon as it enters the same *explored* cell a second time from the same direction.

This works fine, if only a single agent is used. But as this method is a multi-agent algorithm using only one agent is not the goal. As long as only one agent is used, it can simply mark one of the cells as *visited* after the loop was detected. But doing this in a scenario with multiple agents may block the way of other agents and in turn trap these robots, so that they are unable to leave their area and can not help any further with the rest of the exploration.

The procedure to detect and “close” a loop for multiple agents is divided into four steps: Loop detection, Loop control, Loop closing, and Loop cleaning. The Loop detection phase is the same as described above, but the agent will switch into the Loop control phase as soon as a loop is detected.

In the Loop control phase the agent starts following the loop a second time in the same direction. This time each cell will be annotated with its ID, as long as it is not already marked this way by another agent (agent A can only mark "empty" cells and will not overwrite agent B's notations). The agent will realize, that it has taken control of the loop as soon as it enters a cell with its own marking again. If an agent has taken control of a loop in this matter it will switch to the Loop closing phase.

Having taken control over a loop in the previous phase allows an agent in the Loop closing phase to break said loop by changing the status of the current cell from *explored* to *visited*. The agent will continue to do so with each subsequent cell belonging to the loop until it reaches the first cell that has an *explored* neighbor cell which does not belong to the loop, a so called intersection. This enables the agent to enter the last phase of the loop handling algorithm, the Loop cleaning.

In the last phase the agent will remove any of its own footprints from the Loop control phase. It will move backwards through the loop and erase its own markings in each cell it visits.

If an agent is not able to take control of a loop in the Loop control phase, for example because agent B has also started marking the same loop, it will either switch directly to the cleaning phase and abandon the loop handling for this specific loop permanently or it will wait in a cell, going into a *standby* mode, until the state of the cell is changed.

To define the behavior of the agent in this specific case, the authors provided a set of rules that the agent will use to decide whether to continue the loop control, quit it permanently, or pause the controlling phase by going into standby. These rules are as follows for an agent as it decides to move into the next cell:

- control cell: if cell is *explored* without control
- start loop cleaning: if cell is *visited*
- start loop cleaning: if agent B controls cell & ($\text{ID}_B > \text{ID}_A$)
- start loop cleaning: if agent C is in standby & ($\text{ID}_C > \text{ID}_A$)
- switch to standby: otherwise

Finally a set of rules is needed for an agent to decide when to leave the standby state. The agent will switch directly to the Loop cleaning phase, if its current cell is replaced by another agent or if its cell becomes *visited*. It will continue with the Loop control phase, if its cell is cleaned and it will remain in standby in all other cases.

The whole loop detection and handling algorithm works like a semaphore system, which is used to manage access to a common resource to allow concurrent operations of the agents, while keeping the whole system in a logical and consistent state. This system was first proposed in Dijkstra (1968). Which in case of the Brick&Mortar algorithm means, that all *explored* and *unexplored* cells will remain connected and no agent will be trapped behind *visited* cells. Ferranti et al. also state, that they can prove, that this algorithm will never cause a deadlock and that the agents will never get trapped, see (Ferranti et al., 2007, p. 6).

#### Advantages and Disadvantages

This algorithm is an improvement upon the Ants algorithm. Through the use of the markings in the environment the agents are now able to discern which cells were already visited and if a cell is not needed for the remaining exploration. This lessens the need of the agents to traverse the same cell multiple times.

The drawback of this method is, that it requires devices for the storage of the markings in the environment. Also the authors do not supply a feasible way to distribute these markings. Another point, which is not further explained by the authors is, how the collected information at the end of the algorithm gets communicated to the rescue team. The robots will all stop moving, one after the other until no agent is able to move anymore, as all cells are marked as *visited*. If this is the case the algorithm will terminate successfully. But as the agents do not have any way to communicate with each other or the base there is no way to send the information about the found victims onward, such that that the rescue team could do its job.

### 2.2.3. HybridExploration

The HybridExploration algorithm is another method introduced by Ferranti et al. (2009). It is an extended version of the Brick&Mortar algorithm presented in the last subsection.

The assumptions for this version are the same as for the Brick&Mortar algorithm: the agents do not know the terrain, no access to a global positioning system, and no centralized control of the agents. The HybridExploration algorithm also makes use of distributed devices (in this case RFID tags) to store information in the environment. Although the authors make use of active RFID tags, which are able to communicate between themselves. This technique is used to optimize the loop handling part of the Brick&Mortar algorithm.

As described in the last subsection the B&M algorithm has to have a way to solve the problem of agents getting stuck trying to get around a single obstacle, as an agent usually can not determine, if he is stuck in a loop. The Brick&Mortar algorithm offers a solution for this problem, but this in turn requires the agent to travel at least two to three times around the obstacle: the first time to detect, that there is a loop, the second time to mark all cells belonging to the loop, and the third time to change the state of all the cells included in the loop. By using the newer available technology, in this case the ability of the active RFID tags to initiate communication between the distributed tags, the HybridExploration algorithm removes the need for the agent to circle the obstacle multiple times.

For the whole idea to work the authors had to make some specific additional assumptions, c.f. (Ferranti et al., 2009, p. 214):

1. Every communication in this model is free from any errors:
   Perfect tag-to-tag and agent-to-tag communications

2. As the whole model is still based on a grid the agent is able to move from one center of a cell to the next. The agent is also able to identify without any error to which cell a tag belongs:
   Perfect agent movement and localisation of the tags

3. The environment does not change dynamically during the simulation:
   Static environment

4. The batteries used in the tags will last for the whole runtime of the simulation:
   Abundant network lifetime

5. Each tag is able to perform one operation at each time, if multiple agents are communicating with the same tag, the requests will be handled on a first come, first served base and additional requests are stored in a queue:
   Atomic operations

How it works

Ferranti et al. mention, that the drawbacks of the physical agents are twofold. Firstly physical agents are often inefficient in exploring the area, as they will cover already explored terrain more often instead of focusing on unexplored terrain. Secondly the agents will sooner or later visit every cell at least once, but they do not realise when the exploration is finished.

The HybridExploration algorithm combines two parts: agents and dispersed tags. The distributed active RFID chips are combined into a network. The most important change to the Brick&Mortar algorithm is that the agents do not need to circle an obstacle multiple times to detect a loop. The HybridExploration algorithm uses the ability of the tags to communicate with each other to exchange messages and with the help of these messages to detect a loop. These messages represent the virtual agent component of the algorithm.

The virtual agents are messages propagated from one cell, respectively from one active sensor node, to another. This in turn works only on already visited cells, as the robots need to distribute the tags in the terrain, if there are no tags available. Like their physical counterparts, the virtual agents also can only "move" through *explored* cells, they will ignore *visited* cells.

The physical agent algorithm controls the robots and decides how the robots will explore the unknown terrain. This part is covered by the Brick&Mortar algorithm. As stated in the previous subsection, this algorithm is able to explore the terrain, but as soon as there are obstacles in the area the runtime will increase, as the loop handling part of the algorithm will need additional steps to solve the loops around the various obstacles.

Ferranti and his coauthors differentiated between two objectives to assess the performance of the algorithm (Ferranti et al., 2009, p. 214):

1. *Exploration Objective*: all traversable cells in the scenario are at least

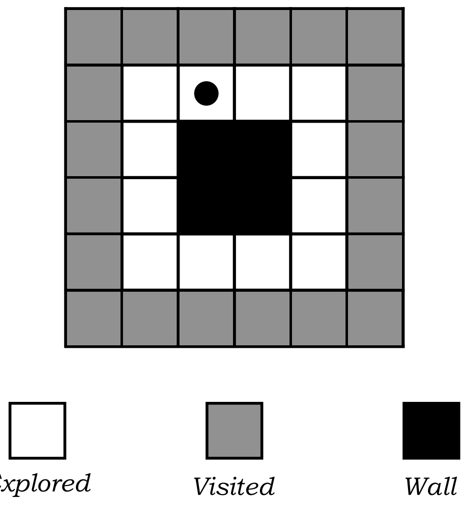

Figure 2.4.: HybridExploration loop problem

visited once, resulting in the state, that no cell will still be *unexplored*

2. *Termination Objective*: the state of all cells will be either *visited* or *wall*

The authors claim, that the physical agents are able to successfully fulfill the *Exploration* and the *Termination Objective* as long as there are no obstacles in the scenario present: in other terms, as long as there is no loop to resolve. If the agents have to resolve one or more loops in the map, the Brick&Mortar algorithm is not able to achieve the *Termination Objective* anymore.

Figure 2.4 shows, how a physical agent alone cannot handle the loop correctly. Usually the agent only has to mark one of the *explored* cells as *visited* to close the loop, but as each cell has two *explored* cells as neighboring cells, the agent is not allowed to change the state of the cell (see also line 2 in Algorithm 1). Thus the agent will continue to try to close the loop as it surrounds the obstacle ad infinitum.

With the help of the virtual agent protocol, which works on the active tags, that are dispersed in the area or are distributed by the physical agents as they move through the terrain, the combined algorithm is able to still terminate successfully and achieve both objectives. The virtual agent protocol builds a tree data structure out of the *explored* cells and initiates a recursive search through this tree. This is a Depth-First-Search tree (DFS), Tarjan (1972). The goal of this data structure is to remove loops or better cyclic paths from the *explored* cells.

The virtual agent protocol sends messages around to the cells and extends the DFS tree with each new cell, that a physical agent has found and marked as *explored*. The algorithm will mark these cells as *visited* when it iterates through the nodes of the tree in the upward direction. While doing this it has to follow two rules:

1. The algorithm cannot mark a cell as *visited* as long as a physical agent is in the same cell.
2. A cell cannot be marked as *visited*, if at least one *unexplored* adjacent cell exists.

By following these two rules the algorithm will follow the physical agents and will remove any loops that are found by the Brick&Mortar algorithm. The messages sent through the RFID tags, each representing a virtual agent, do not delay or trap the physical agents and in turn the combined algorithm is able to fulfill the *Exploration* and the *Termination Objective* again.

The implementation of these two rules are shown in lines 9-12 and 22-25 in Algorithm 2, which shows how the behavior of the virtual agent is realised, (Ferranti et al., 2009, p. 225). The *current cell* in the code describes the cell in which the agent is currently located. The *previous cell* is the cell, from which the agent is coming. In the DFS tree the relationship between the cells is defined as *parent* and *child*. Each parent cell has a counter for each of its children cells. This counter expresses the number of virtual agents, that moved through the parent cell into this specific child.

### Advantages and Disadvantages

Ferranti and coauthors showed in Ferranti et al. (2009) that the use of the virtual agent decreases the runtime as compared to Brick&Mortar algorithm, c.f. Ferranti et al. (2007).

An additional assumption that was made for the algorithm to work, was the use of active RFID tags that are able to communicate among each other to facilitate the virtual agents. Without this sort of data exchange between the tags the algorithm will fall back to the behavior of the B&M algorithm.

Another disadvantage is the applicability of the algorithm. The virtual agents have to follow two rules, as described above. The second rule states, “that a

**Algorithm 2** HybridExploration, Behavior of the virtual agent

```
1: if you are coming from a cell which is child of the current one then
2:     decrease the counter of the current cell associated with the child
3: end if
4: if the current cell is not part of the DFS tree then
5:     mark the current cell as part of the DFS tree;
6:     define the previous cell as parent of the current one
7: end if
8: if the parent of the current cell is visited and the current cell has one
   child cell only and all the adjacent explored cells are part of the DFS tree
   then
9:     if there are any adjacent unexplored cells or a physical agent is in the
       curren cell then
10:        stay in the current cell;
11:        return
12:    end if
13:    mark the current cell as visited
14: end if
15: if there are explored adjacent cells which are not part of the DFS tree
    then
16:    go to one of them [each agent chooses a different cell according to its ID]
17: else if there is at least a one child which is not visited then
18:    go toward the child cell with the minimum associated counter;
19: else
20:    if there are any adjacent unexplored cells or a physical agent is in the
       current cell then
21:        stay in the current cell;
22:        return
23:    end if
24:    mark the current cell as visited;
25:    go to the parent cell
26: end if
27: if you are going to a cell which is child of the current cell then
28:    increase the counter associated to that cell
29: end if
```

virtual agent cannot mark a cell as *visited* if at least one of the adjacent cells is *unexplored*. . . . the virtual agent stops any activity until the adjacent cell is *explored* by a physical agent." (Ferranti et al., 2009, p. 224). Currently the only data stored in the tags is the state of the cell, which is one of the three possible states: *unexplored*, *explored*, and *visited*. It is especially unclear, if this rule can be implemented by the virtual agents with the data that is currently available to the agents, both virtual and physical. How does a virtual agent distinguish between a wall and an unexplored cell? Both of those are for the virtual agent the same, a blank spot.

Also the problem of relaying the information about the victims forward to the rescue team was not really addressed in this paper. One possibility would be the use of the dispersed RFID tags as a communication network to forward any information about a found victim towards the rescue team, but this was not clearly described by the authors.

### 2.2.4. Rendezvous

The usual problem in a search and rescue scenario is, that there is no way to guarantee a reliable form of wireless communication between the agents or between the agents and the rescue base. Julian de Hoog, Stephen Cameron, and Arnoud Visser proposed another solution for a multi-agent algorithm to help in a search and rescue scenario. They introduced the idea to use optimized meeting points for robots to share and exchange data, De Hoog et al. (2010).

The central idea behind the algorithm is to use a heterogeneous agent population: one part of the population tries to explore the unknown terrain and to find the victims. The other agents will act as relay to forward any collected data between the exploring agents or directly towards the headquarter. This results in a role-based exploration scheme. As wireless communication may not be possible in a search and rescue scenario the agents have to meet physically to exchange data. For the most efficient use of the relay agents de Hoog et al. created an algorithm that calculates optimal meeting or rendezvous points for the data exchange in the scenario.

Each agent is assigned one of the following roles:

- *Explorer*: The exploring agent will move through the unknown terrain and will communicate its collected data to the nearest relay agent, which will wait at a specific calculated rendezvous point.

- *Relay*: The relay agent will exchange data with one or more exploring agents and in turn it will periodically return to the headquarter. Additional information that is gathered as the relay agent is on its way will be added to the data store and in turn shared.

The team hierarchy is determined at the start of the algorithm. As stated above, one relay agent may collect the data from more than one exploring agent. There may be also multiple relay agents between one relay agent in the field and the headquarter. Also if an exploring agent meets a relay agent before the specific rendezvous point is reached, the data exchange will be done in this instant instead of waiting for them both to move to the meeting point.

An explorer will exchange its complete knowledge with a relay agent, resulting in the state that both agents will know the exact same things (all gathered sensor input is collected in a map by the exploring agent). Also both types of agents share the same movement model, this in turn allows the exploring agent to calculate how much time the relay agent needs to move back to the next relay agent or to the headquarter. Thus enabling the exploring agent to predict when it needs to return to the rendezvous point at exactly the right time to meet the relay again.

If the explorer saves the exchanged map from the relay agent separately, it is able to predict the position of the relay robot, even if they are not in range to communicate (the map of the relay agent will not change that much). Additionally a pair of exploring and relay agents can specify a fallback point, in case that the "normal" rendezvous point may not be reachable (which is only the case, if the scenario allows a dynamic change of the environment).

#### How it works

The exploration is based on the frontier exploration algorithm introduced by Yamauchi (1998). This concept is based on the central question in exploration: *"Given what you know about the world, where should you move to gain as much new information as possible?"*, (Yamauchi, 1998, p. 47). The frontier based algorithm will answer this question by moving its agents to the boundary between known and unknown terrain.

Brian Yamauchi defined a *frontier* as a region on the boundary between open space and unexplored territory. As a robot reaches a specific frontier it can gather new information about the formerly unknown terrain and in turn create new frontiers. By successively moving from frontier to frontier an agent can continually increase its knowledge of the world.

In a multi-robot scenario each robot will use its own sensor data to create its own list of frontiers, which are detected by this robot. In turn this data is also broadcasted to all other robots. This allows for a decentralized approach and the failure of a single robot will not restrict or hinder the other robots.

The main idea behind the rendezvous algorithm is not the specific type of exploration method, but the precise calculation of meeting points for the data exchange between the different agents. Previous versions of the Rendezvous, see De Hoog et al. (2010), algorithm simply used the position of the exploration agent, as it has to turn back for the data exchange as the future meeting point. This led to a deeper exploration of the environment, as the relay agent had to follow the exploration agent into the explored terrain. But this approach also had its drawbacks, creating inefficiencies and unnecessary backtracks, c.f. (De Hoog et al., 2010, p. 4).

De Hoog et al. introduced another idea to calculate meeting points: the use of thinning algorithms from digital image processing to "…reduce a shape to its skeleton by making the shape as thin as possible while keeping it connected and centered." (De Hoog et al., 2010, p. 4). The agents gather information about the terrain as they explore the unknown area and build a map based on this data. A two dimensional representation of this map can now be used with the help of a thinning algorithm to calculate the rendezvous points.

After the list of possible rendezvous points is available the algorithm has to choose the optimal point, so that the meeting point is still deep in the explored terrain but not far enough away to provide a good communication range for the agents. Points with the highest neighbor traversal value will be preferred, as these points mark important junctions. If two or more points share the neighbor traversal values, the rendezvous point with the best communication range is chosen.

### Improving the Rendezvous algorithm

In 2014 Victor Spirin and Stephen Cameron introduced an improvement to the Rendezvous algorithm, as described by Spirin and Cameron (2014). They argued that a single rendezvous point for a pair of agents is easy to implement and robust to communication failure, but it may not make use of the whole communication range available to the robots. While the robots may be able to exchange data through some obstacles or over rugged terrain, which would either hinder the robots or add a lot of travel time, the version from De Hoog et al. would ignore this advantage and force the robots to still meet at the meeting point although this would increase the movement time.

Spirin and Cameron proposed to use two rendezvous points or a point pair. A pair, $a$ and $b$, of these points would have to comply with two conditions:

1. $a$ and $b$ are in a range, that would allow a communication to take place between two agents
2. The distance from $b$ to the base has to be closer or at least equal to $a$. This would lead an explorer to meet at $a$ and relay to $b$.

If $a = b$, then the usual rendezvous algorithm applies.

In the end the improvement does not speed up the exploration time significantly, but it does enable a faster message propagation, which in turn allows the rescue team to act faster, c.f. (Spirin and Cameron, 2014, p. 5).

### Implementation

The authors of the Rendezvous algorithm implemented their method in their own simulator called Multi-Robot Exploration Simulator (MRESim), see also Spirin et al. (2014). It supports multiple algorithms that were used in the research of Julian de Hoog and Victor Spirin respectively, c.f. De Hoog et al. (2010); Spirin et al. (2013); Spirin and Cameron (2014). The software is available at GitHub[2] released under the GNU GPL 3.0.

### Advantages and Disadvantages

The Rendezvous algorithm allows the use of two different populations of robots to act as a team for the exploration of an unknown terrain. Especially

[2]https://github.com/v-spirin/MRESim

the communication structure offered by the relay agents allows for a faster and more robust message propagation between the agents and especially towards the headquarter. While the exploration of the terrain is not faster than a traditional frontier based algorithm, the information available at the headquarter is about 9 to 10% higher, see Fig. 6 in (De Hoog et al., 2010, p. 9).

### 2.2.5. Requirements for a search & rescue algorithm

This is a short summary of the points and restrictions that a viable multi-agent search & rescue algorithm has to adhere to:

1. The agents should be able to operate in extreme terrain and operating conditions
2. The algorithm has to function in GPS- and wireless-denied environments
3. The agents have to act autonomously without any help of a human teleoperator
4. The algorithm should find as many victims as possible in the shortest time frame

The first two points are taken from (Murphy, 2014, p. 5). The ability to operate in extreme terrain and conditions depends mostly on the hardware used for the actual robots. The agents need sufficient sensors to be able to sense their surroundings and gather enough data to act accordingly. The second point is, especially pertaining to this thesis, the more important point. The agents of a multi-agent algorithm need to communicate between each other to achieve the synergy between the team members. This means that the algorithm has to provide a robust mechanism to communicate between the agents. Additionally, the agents can not rely on GPS signals to determine their exact position in the terrain. Thus the algorithm has to work without any GPS signals. The third requirement is the capability of the algorithm and the agents to work without any additional input from a human team member as soon as the algorithm is started. The last objective for every viable search & rescue algorithm is to find as many victims as possible in the shortest time frame attainable.

### 2.2.6. Summary

Four different multi-agent algorithm were presented in this section. Each of these methods is able to explore unknown terrain and still allows communication between the agents, despite the constraints imposed due to the search and rescue scenario.

The common way to still offer a possibility to communicate data from one agent to another is the use of indirect communication. Be it either through chemical markings used in the Ants algorithm or by leaving sensor nodes in the terrain as the HybridExploration algorithm does. Each variant has its own advantages and disadvantages. A chemical marking is easier to read, but harder to overwrite. A sensor node needs to be distributed in the terrain and, in the case of the HybridExploration algorithm, it still has to be able to send signals from one sensor node to another. This in turn could be prone to errors as the scenario specified that radio signals may not be reliable. The Rendezvous algorithm also relies on radio signals for communication, but imposes additional restraints, which in turn decrease the error rate drastically. It specifies that the agents can only use a line of sight communication at specific points in the terrain, thus decreasing the range of the wireless data transmission and in turn increasing the reliability.

Of all described algorithms only the Rendezvous algorithm offers a way to get information about the found victims back to the human team members. The other algorithms do not specify how the information at the end of the algorithm is passed on, thus allowing the rescue team to start its work.

The HybridExploration algorithm looks the most promising, but has the inherent drawback of the direct communication between the sensor nodes.

Each of the listed algorithms follows the four requirements for multi-agent search & rescue algorithm in their own way. None of the algorithms presented here offers a specific hardware layout for their agents. Each of the algorithms works without GPS. The main difference between the algorithms introduced in the second part of this chapter is the way how they allow their agents to communicate between each other. The Brick&Mortar algorithm relies on indirect communication between the agents, using RFID chips that are not further specified. This applies also to the HybridExploration algorithm. The authors only specify that the RFID chips should be some kind of an active sensor. The Rendezvous algorithm uses direct communication via radio

| Name | Goal | Communi-cation | Movement | Information back to HQ | Maintain-ability | Size |
|---|---|---|---|---|---|---|
| Ants | PoIs | Indirect | Grid-based | ✗ | ✗ | ✗ |
| Brick&Mortar | Explore | Indirect | Grid-based | ✗ | ✗ | ✗ |
| HybridExploration | Explore | Indirect | Grid-based | ✗ | ✗ | ✗ |
| Rendezvous | Explore | Direct | Grid-less | ✓ | ✗ | ✗ |

Table 2.1.: Comparison of the different existing algorithms

signals, shortening the range between the agents until a data transfer between two agents should be possible. The only similarity between the algorithms, is the use of a single type of communication: either indirect or direct.

A comparison of the different features of the presented algorithms is listed in Table 2.1. The *goal* row describes if the algorithm tries to explore the terrain or find points of interest. The *communication* column shows what kind of communication model the algorithm uses. The *movement* column describes if the agent of the algorithm uses a grid or is able to function without a grid. The column *information back to HQ* shows whether the algorithm has a way to communicate any gathered information back to the base during the runtime. The second to last column, *maintainability*, represents the ability of the agents to return to the base for maintenance during the runtime. The last column, *size*, illustrates the ability of the algorithm to stipulate a specific size for the agents.

Chapter 3.

# The Multi-Agent Flood Algorithm

The previous chapter introduced different rescue robots that are currently used in real rescue scenarios and/or are used in research projects for future use in these scenarios. Each of these robots offered different kinds of advantages and disadvantages, but almost all of them have in common, that they currently need a human operator to do their work. This means that a human needs to control each robot used in a rescue mission, which in turn also implies, that each robot has to have some sort of connection back to the headquarters. Other examples show, as the attack on the twin towers in New York in 2001 or the nuclear disaster in Fukushima in 2011 [Davids (2002); Murphy et al. (2008); Murphy (2014)], that one cannot use wireless communication for the teleoperation of rescue robots. As a consequence each robot needs a wire that connects it with the base. This cable does have some advantages, as it can be used as a tether to pull the robot back if it gets stuck or to belay it. The main drawback remains, be it by wire or by tether, that the robot is connected to the base and the movement of the robot has to adapt to this trailing cable.

Another handicap of most of the introduced robots is, that they act alone and that teamwork is only the job of the human controllers. The ICARUS project tries to implement some teamwork between the robots, as the larger unmanned ground vehicle will carry the smaller robot to where it may be needed. The robots of the Hector team try to solve the problems of the RoboCup competition using teamwork, which is one of the few instances where the robots try to act autonomously and use the team as an advantage to reach the desired goal faster through communication and cooperation.

Each of the machines mentioned in the previous chapter show what current technology is able to do for search and rescue scenarios. Robots are able to find buried victims in the case of an urban search and rescue mission. They are able to move autonomously and they can work as a team, if they are programmed accordingly.

Chapter 2 also put forward different types of multi-agent algorithms. Methods that are based on agents, which use teamwork, and the ensuing communication and cooperation to reach the goal of the algorithm. Applying these algorithms to a team of robots will allow them to act autonomously, so that no human teleoperator will be needed for each robot, and it will also grant the rescue team the ability to insert multiple robots into the disaster area. Whereas “multiple” can in the simplest case still mean only one or two robots, or if the scenario warrants it the numbers can easily rise to fifty or sixty robots for example.

As the methods mentioned previously have illustrated, the different multi-agent algorithms are able to explore the unknown terrain with out any additional input by human controllers and they are able to find points of interest, or in this case victims. Similar to the different rescue robots, each algorithm has its own advantages and disadvantages. The main point that links most of the introduced algorithms is that each method needs to create some kind of markings in the environment, so that the agents can communicate with each other. The Ants algorithm proposed to use chemical markings, whereas the Brick&Mortar and the HybridExploration algorithm rely on electronic tags or Radio-frequency identification (RFID) chips. However the authors do not offer any more information on how RFID tags are distributed in the environment and which kind of chips are used for the algorithms.

Nevertheless each algorithm follows the same premise: Find as many victims as possible in an unknown terrain and do this as fast as possible. Additionally information about any victim should be forwarded to the search and rescue headquarter, so that the rescue teams can start to work. Time is very valuable in this case, as the chance to find surviving victims after 48 to 72 hours after the disaster declines rapidly [Murphy et al. (2008); Murphy (2014)].

Only the Rendezvous algorithm, from all of the different shown methods, really addresses this aspect. It may not be really faster to search the terrain as a simple frontier based algorithm [De Hoog et al. (2010)], but it tries to get the information about any new found victim to the headquarter without any delay. Whereas the authors of the Brick&Mortar algorithm and the HybridExploration algorithm do not even mention it, although one could argue that the active tags used by the HybridExploration method could be used to send messages to communicate this information to the headquarter.

The method proposed in this chapter will provide a solution to the two main points of a multi-agent search and rescue algorithm: be able to explore unknown terrain autonomously, without relying on GPS signals and radio communication, and find possible victims in the first 48 to 72 hours. Additionally it will forward the information about any found victim to the headquarter as fast as possible by returning to the starting point. This way the human rescue team members can start the rescue process in parallel with the still ongoing search process. This return to the base also allows for a maintenance of the robots, which is completely ignored by the existing algorithms.

The next chapters of this work introduce further changes to the algorithm presented here, so that the method is more applicable to real hardware. Chapter 4 demonstrates how the communication model can be extended to create a robust communication channel for the agents of the algorithm. Further optimizations are shown in Chapter 5, as more of the initial abstractions are removed. Additionally concepts about the use of RFID chips are discussed, which offer additional enhancements and an increased application for the communication between the agents.

## 3.1. The Multi-Agent Flood Algorithm

This algorithm represents the first version by the author, proposed in Becker et al. (2013). The difference to already existing algorithms is the possibility to send information about any found victims back to the base during the runtime. Another advantage of this approach is to enable a maintenance of the different agents and an accumulation of the already collected data at a single point. Furthermore the agents using this method are able to monitor already explored terrain and thus enables the algorithm to adapt to any possible changes in the environment.

## 3.1.1. Assumptions and Definition of the Scenario

To ensure the communication and the cooperation between the agents the algorithm uses indirect communication by using markings in the environment, this way each agent can read and then act based on this data. The algorithm uses RFID chips distributed in the environment and to store the data, that is intended for the other agents. The map of the disaster area is divided into a grid, akin to the model used by the Brick&Mortar algorithm. The cell-size of the grid depends on the used RFID tag, as an agent should be able to read the chips surrounding its current position. Each cell represents either traversable space or a non-traversable wall. Additionally, a traversable cell can contain either the starting point or the headquarters, or a point of interest, a victim in this case. As a consequence the agent has to be able to "see" its surrounding environment. To be able to sense adjacent cells the agent has to be equipped with different sensors. For example, to calculate the distance to the nearest walls a laser-range finder can be used. Otherwise the agent can also employ a camera to discern different characteristics of the environment. The combination of various sensors, the laser-range finder, one or more cameras, a bumper sensor, et cetera, allows the agent to detect the diverse objects in the neighborhood. This algorithm is called the Multi-Agent Flood algorithm or MAF.

As a consequence a robot controlled by this algorithm needs at least a laser-range finder and a bumper sensor to gather all data relevant for movement. Further sensors like at least one or two infrared cameras are needed to identify a victim in the environment.

## 3.1.2. Design of the Agent

The base agent of the MAF algorithm can only react to the environment that it can currently sense and to the markings, which it can currently read. Each cell, with exception of the cells at the border of the grid, has eight neighboring cells. This is called the Moore neighborhood, after Edward F. Moore, as depicted in Figure 3.1.

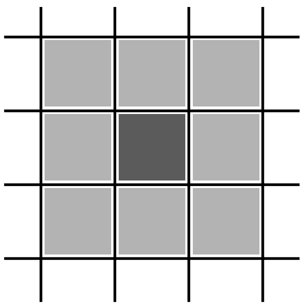
Figure 3.1.: Moore neighborhood, as seen from the dark gray center

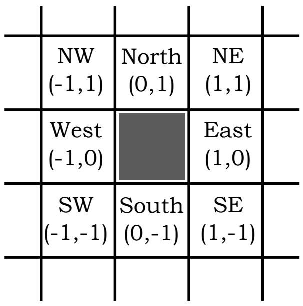

Figure 3.2.: Directions in a grid

Search Mode

Each agent has an orientation in which it will move. With a high probability (95%) an agent will move in this direction, in the case of the other five percent it will either move to the cell to the left of the current direction or in the cell to the right.

Let $directionList[]$ be an array of direction modifiers (or a an array of two dimensional vectors):

$$directionList[] = [(0, 1), (1, 1), (1, 0), (1, -1), (0, -1), (-1, -1), (-1, 0), (-1, 1)]$$

These direction modifiers in $directionList[]$ represent the following directions in the exact same order: North, North-East, East, South-East, South, South-West, West, North-East. In other words these are the directions on the compass rose starting from the North and going clockwise around the rose, ending at North-West. This is also shown in Figure 3.2. For example agent $A$ has a direction of East or 90°. In 95 times out of one hundred it will follow this direction, with a chance of two and a half percent it will change its heading to North-East or 45°, and in the case of the other two and a half percent it will switch to the South-East direction or 135°.

Each agent will store its current heading in a variable called $currentDirection$, represented by an integer between zero and seven. The next direction for an agent is then simply calculated by generating a random number and in case of a change of heading by either adding one or substracting one from the $currentDirection$ using modulo eight:

$$newDirection = \begin{cases} currentDirection & & 95\% \\ currentDirection - 1 & \mod 8 & 2.5\% \\ currentDirection + 1 & \mod 8 & 2.5\% \end{cases}$$

Let $NewPos(pos) : (x, y) \rightarrow (x', y')$ be a function to calculate the new position of an agent. It is defined by adding the specific direction modifier to the current position of the agent.

As this does not yet include any communication or even cooperation between the agents, an agent will need to mark its current cell to store information for the next agent, that visits this cell.

Each agent will count its steps from one cell to another, starting with zero at the starting point, or the headquarter. These steps are counted in a variable called $stepCounter$ and it is incremented by one as soon as the position of an agent is updated. As an agent enters a new cell it will store its current value of the $stepCounter$ variable in this specific cell. If the agent has entered a cell which already has a marking, then the stored value from this cell is read out first. The agent will then compare the newly acquired step counter with its internal $stepCounter$. If the internal value is smaller than the stored value in the cell, then the agent will overwrite the stored value with the number from its own $stepCounter$, otherwise it will leave the stored number alone. In no case will the internal $stepCounter$ be changed or overwritten, as without it a continuous marking is not possible.

This describes the communication between the agents, and the movement algorithm has to be adapted to include these markings to achieve a cooperation between the agents. The algorithm introduced above does not take any markings into account. To change this, additional cases have to be taken into consideration:

- If any neighboring cell contains a point of interest, then the agent will move directly into this cell. If multiple points of interest are found in the neighboring cells, then go to the first one found in the list of cells (the list will start at the upper left and end at the lower right).
- Already marked cells will be ignored as long as unmarked cells are available.
- If only marked cells are available, the agent will move to the cell with the highest marking with a chance of 95% and to any other cell otherwise

**Algorithm 3** Multi-Agent Flood algorithm movement

```
if is a point of interest in a neighboring cell then
  point newDirection to this cell
else if are there any unmarked cells left in the neighborhood then
  generate a random number i between 0 and 1;
  if $i \leq 95$ then
    newDirection = currentDirection
  else if $95 < i \leq 97.5$ then
    newDirection = currentDirection - 1 mod 8
  else
    newDirection = currentDirection + 1 mod 8
  end if
else
  generate a random number i between 0 and 1;
  if $i \leq 95$ then
    point newDirection to the cell with the highest marking
  else
    repeat
      r = random number between 0 and 7
      newDirection = r
    until $r \neq$ currentDirection
  end if
end if
return NewPos(oldPosition)
```

The code listed in Algorithm 3 describes the searching part of the Multi-Agent Flood algorithm. Agents are now able to start from the starting point, explore the unknown terrain, and find victims in the scenario. This is the one part of the algorithm, the second part handles the conduct of an agent after it has found a point of interest. To simplify this the "intelligence" of an agent is split into two parts: The first part, that describes the search for possible victims is called the *Search Mode*, the second part, which controls how an agent should act after it has found a point of interest is called the *Return Mode*.

## Return Mode

After an agent has found a point of interest in the recently explored terrain it will move to this point, accordingly to the movement algorithm described above. As soon as it reaches the cell, where this victim is located, the mode of the agent will switch and it tries to get back to the starting point on the shortest currently known path. The agent is able to detect the currently shortest known path through the markings in the environment, that were created during the *Search Mode* by itself and the other agents. This is also were the "flood" part of the name of the algorithm comes into play.

Consider a graph $G = (V, E)$ containing $V$ vertices and $E$ edges, whereas an edge is an unordered pair of two vertices $e_n = \{v_x, v_y\}$. Let node $v_0$ be a randomly chosen starting point in the graph. To apply a flooding algorithm one starts at $v_0$ and marks it with one. Now create a subset $s_1$ of all edges that are connected to $v_0$ and mark all corresponding vertices, that are part of these edges with the number two. In the next step create a new subset $s_2$ and chose all edges, where only one of the nodes is already marked and select the unmarked vertices from these edges. Mark these unmarked vertices with the number three.

The following is a formal description on how to choose the edges for a subset $s_n$ with $n > 0$:

$$s_n := \{\forall e \in E : e = \{v_{n-1}, x\}, \text{for } x \in V \text{and } x \neq v_{n-1}\}$$

After all nodes are marked in this manner, the algorithm will terminate successfully. This results in a marked graph and by using the markings the shortest way back from any node to the starting node $v_0$ can be found. To do this simply pick any node, for example $v_x$. Now check all edges that contain $v_x$ and read the markings in the nodes that are connected through these edges

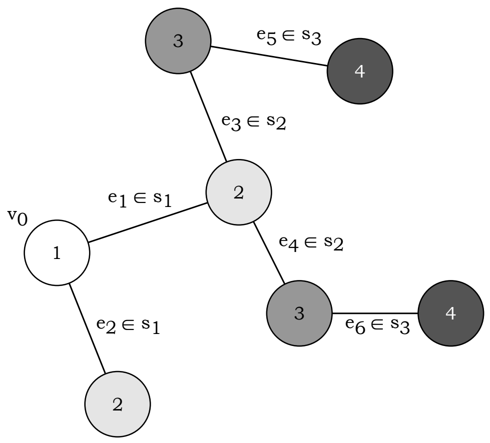

Figure 3.3.: Example of the flooding algorithm for graphs

to $v_x$. Pick the node with the smallest marking in it and repeat this procedure from this node onwards. By always picking the node with the smallest mark, the algorithm will move through the graph by using the shortest path which connects the chosen end point $v_x$ with the starting point $v_0$. The graph in Figure 3.3 shows an example how the algorithm works. $v_0$ is colored in white and marked with the start counter 1. The edges are named from $e_1$ to $e_6$ and each edge also shows to which subset of $s_n$ it belongs. The rest of the vertices is already marked with the step counter and colored in a color that is getting darker as the step counter gets higher. This is called a flooding algorithm for graphs.

Richard Bellman proposed a cellular automaton in Bellman (1972), which is able to compute the shortest path between the starting point and one or more end points. His version is based on the flooding algorithm for graphs and works in the same way. By choosing a starting point in a cellular automaton $p_0 = (x, y)$ with an initial marking of one and the following rule to update any cell: If a cell has an already marked neighbor cell, using the Moore neighborhood, then also mark said cell with the same number incremented by one as the already marked cell.

When all cells in the automaton are marked stop the automaton and a shortest path will be available from any arbitrarily chosen cell back to the starting point $p_0$. This works just as in the flooding algorithm for graphs: Take a look at the markings of the neighboring cells, pick the one with the smallest marking and repeat the steps in this cell until the starting point is reached.

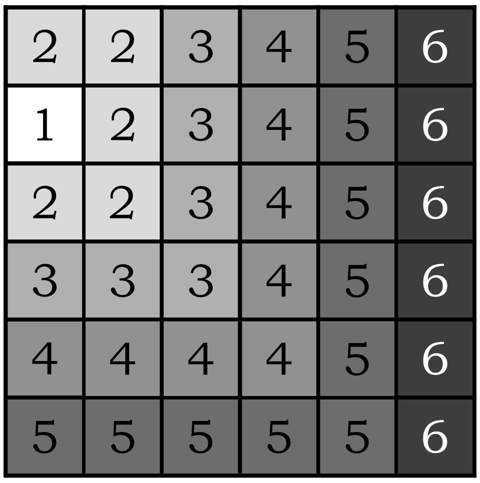

Figure 3.4.: Example of the Bellman flooding algorithm as a cellular automaton

Figure 3.4 shows an example of the algorithm in a 6×6 grid with the starting point shown in white and the mark set to one. Successive steps are colored a little bit darker and the mark is always incremented by one until the whole grid is covered and each cell is marked with a step counter.

Bellman's approach is easily adaptable to be used in a multi-agent algorithm. Each agent possess a step counter, which is incremented by one as the agent moves from one cell to another, and as soon as an agent enters a cell, it will mark said cell with the current counter, as long as the specific cell is yet unmarked.

As depicted in Figure 3.5 one agent is able to mark all cells in this fashion after enough time has elapsed. Although the markings will not really represent the same pattern as shown in Figure 3.4. The markings will not be concentric and there may be huge differences in the values of the markings of two neighboring cells. It is easily seen, that the current, while flawed approach, is usable by multiple agents. Each agent uses his own step counter and is able to mark the cells independently as shown in Figure 3.6.

To achieve the functionality of the flood algorithm proposed by Richard Bellman the agents have to be able to overwrite already existing marks in the cells. An agent is allowed to overwrite a mark, if the value of said marking is higher than the current step counter stored in this agent. This is illustrated in Figure 3.7, where the blue agent marks the cells first and the second agent, in red, overwrites the cell with the mark containing the nine, changing it to a

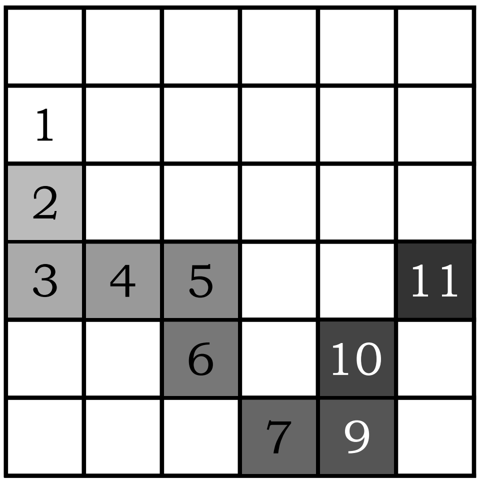

Figure 3.5.: Single MAF agent marking the grid

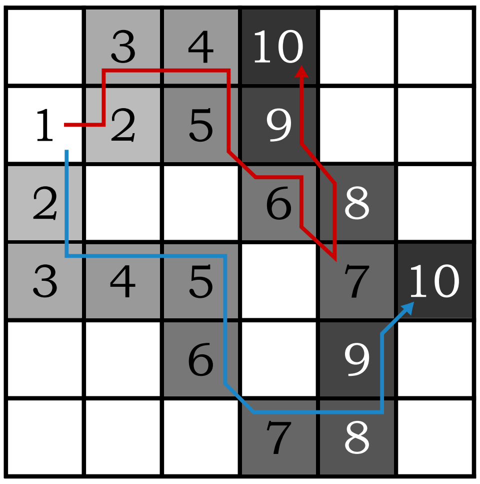

Figure 3.6.: Two MAF agents marking the grid simultaneously

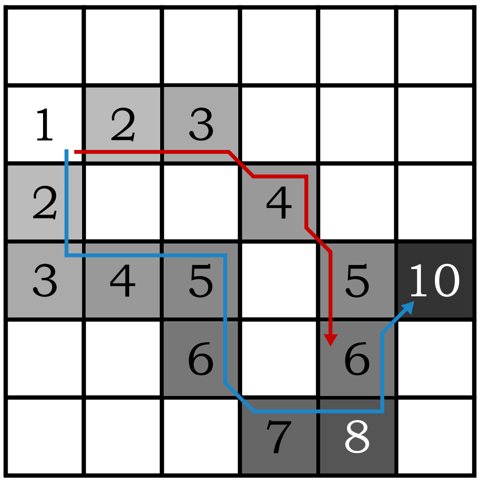

Figure 3.7.: Two MAF agent marking the grid, the second overwriting the mark of the first

six.

As the step counter is only incremented by the movement of the agent and never decremented the algorithm also needs a method to reset the counter, so that an update is even possible. The step counter of an agent will be reset to one as soon as it reaches the starting point again. This allows the agents to start the marking process again with a fresh step counter and enables them to update already existing marks in cells. Now it is only a matter of time until the same concentric pattern as presented in Figure 3.4 will emerge.

**Algorithm 4** Multi-Agent Flood algorithm Return Mode

```
1: List neighbors[] = getNeighboringCells(currentPosition);
2: int min = MAX_INTEGER_VALUE;
3: for all Cell c ∈ neighbors[] do
4:    if c.getMark < min then
5:       min = c.getMark
6:       newDirection = getDirection(currentDirection, c)
7:    end if
8: end for
9: return NewPos(oldPosition)
```

The code shown in Algorithm 4 describes how the movement will work in the *Return Mode* of the Multi-Agent Flood algorithm. An agent will switch from the *Search Mode* to the *Return Mode* as soon as it enters a cell with a point of interest in it. In turn it will switch back from the *Return Mode* to the

*Search Mode* as soon as it arrives back at the starting point. This will also reset the step counter, as mentioned above, and will enable the agent to start the search again.

The combination of these two modes allows an agent to explore the unknown terrain, search for victims, and return as soon as a victim is found to the headquarter with the information about the victim and the currently shortest known path to reach said victim.

### 3.1.3. Comparison with existing algorithms

Chapter 2 lists four different variants of multi-agent algorithms for the use in search and rescue scenarios. Two of them, Brick&Mortar and HybridExploration, are more or less the same algorithm, whereas the HybridExploration variant is an improvement to the base Brick&Mortar algorithm.

The Ants algorithm was one of the first proposed multi-agent algorithms for search and rescue scenarios based on the biological ant. It showed that the approach using simple robots to do a complex task like a search process after a disaster was a feasible thing, although the way to use the indirect communication was yet not very practicable. Using chemicals to mark the surrounding terrain does not allow for a lot of information to be stored in the markings and depending on the chemicals may not be very reliable, if there is water from rain or a destroyed pipe around. Even an open fire, which is a real possibility after a disaster, may hinder the use of these chemicals for storing information the environment.

The Brick&Mortar algorithm and its improved version, the HybridExploration algorithm, proposed to use electronic markings or tags to store information in the environment. This is usually realised by either using already existing tags or by distributing these tags while exploring the unknown terrain. Although the authors of the algorithms proved that their method will explore the whole terrain and, under these circumstances, will find all the points of interest spread through the scenario, did not really provide a way for the agents to rely the information about any found victims back to the base. Instead the algorithm was based on the behavior, that the robots will stop moving as they have finished exploring and thus are unable to transfer the vital information to the rescue team. Additionally, as the agents controlled by these algorithms would ignore already explored terrain and would never visit it again, these algorithms would not be very applicable in a dynamic scenario, where the terrain may change or where the victims may be mobile and could move from unexplored terrain into already explored terrain. The latter case would result in stuck robots who would ignore said victims completely.

The fourth method introduced in the second chapter, the Rendezvous algorithm, was the first algorithm that was designed from the ground up to rely information as fast as possible to the headquarter. By introducing a second population of agents whose job it is to solely handle the communication between the population of the searching agents it is able to transfer vital information back the base. But as the authors stated it was not optimized to improve the exploration and search by itself, c.f. De Hoog et al. (2010).

The Multi-Agent Flood algorithm tries to combine the good parts of each of those algorithms into one. The use of indirect communication offers a robust framework so that the algorithm can still function, if no direct transmission via radio signals is possible. The split into two modes offers further advantages. By returning to the base after a victim was found on the currently shortest known path the agent will provide vital information to the rescue team. The team is able to act on this information as soon as the agent arrives at the base, starting the rescue process in parallel with the search process. Gaining an important time advantage and raising the chance to find victims still alive.

This behavior offers also the option to do maintenance on the robots as they return to the starting points. Usually rescue robots are powered by batteries. These batteries allow a robot to work for an average of three hours in the field [Birk et al. (2006b); Rohmer et al. (2010)]. By always having an option available to return to the headquarter a robot is now able to automatically initiate a switch to the *Return Mode* in time and find its way back to the starting point if the power is low. As the MAF agents do not completely ignore already visited terrain, they are also able to act in a dynamic scenario, as they will simply check their surroundings for any victims and, if a victim is found in already explored terrain, will trigger the mode switch and return homeward.

## 3.2. Experimental Simulation Setup

To show that the Multi-Agent Flood algorithm is applicable and a working solution to find victims in unknown terrain in a search and rescue scenario it was simulated multiple times using different numbers of agents and different maps.

### 3.2.1. The Simulator

The simulator used for the simulations is based on the Multi-Agent Simulator Of Neighborhoods framework, abbreviated as MASON, which was created by Sean Luke and others.

> "... MASON is a general-purpose, single-process, discrete-event simulation library intended to support diverse multiagent models across the social and other sciences, artificial intelligence, and robots, ranging from 3D continuous models, to social complexity networks, to discretized foraging algorithms based on evolutionary computation (EC). MASON is of special interest to the social sciences and social insect algorithm community because one of its primary design goals is to support very large numbers of agents efficiently. As such, MASON is faster than scripted systems such as StarLogo or breve, while still remaining portable and producing guaranteed replicable results."

The quote from (Luke et al., 2003, p. 2) describes the design goals of the library written in Java, which was improved and extended into a whole framework in Luke et al. (2004, 2005).

The MASON framework provides a suitable environment for the testing of the Multi-Agent Flood algorithm, as the underlying tools of the simulator were already available, namely the scheduler and the interfaces to the graphical visualization. This allowed for a rapid programming and testing of the algorithm, without having to worry about the correctness of the core of the simulator.

### 3.2.2. Definition of the Scenario

The model used for the Multi-Agent Flood algorithm was in part described above. The map is modelled as a grid or two dimensional matrix $M^{m \times n}$. Each cell of the matrix can have four different states:

1. *Wall*: This state represents an obstacle and is not traversable for the agents
2. *Empty*: This is the "usual" state of the cells, agents can freely traverse this cell
3. *Point of Interest*: This designates a victim or PoI and agents can move onto this cell, initiating a mode switch in the specific agent
4. *Starting Point*: This cell is the starting point for all agents and the agents will return to this cell if they are in the *Return Mode*

To store the step counter markings from the various agents, each cell also has an assigned integer value, that will be used to represent these markings.

As the algorithm does not include a way to handle the case of agents being positioned on the margin of the map, each map will have a wall around the border. The *starting point* will be placed randomly in a cell adjacent to this border. Only one starting point may be designated in the scenario and all agents will start from this point. The points of interest or victims will be uniformly distributed in *empty* cells through the matrix.

The time in the simulation was measured in steps. Each agent is able to move from one cell to another in each step. Changing the heading does not take any time and will be done instantly before the agent moves. Marking a cell or updating the already existing mark will also not consume any further steps.

The steps will also be the main measurement of the algorithm, as they will represent the time, that the MAF agents need to explore the terrain, find the victims, and transport the information about the victims back to the starting point. As the Multi-Agent Flood algorithm has no internal termination a set of rules was defined to terminate the algorithm. To finish the run successfully the agents have to find at least 95% of all points of interest and they have to explore at least 95% of the explorable and traversable cells. If they could do that before the time limit was reached, the run was considered a success. If they did not reach these goals in the allotted time frame, the run resulted in a failure. The time limit was dynamically set to depend on the size of the map. The maximum number of steps, which were allowed to be used before the termination of the simulation was the product of the length and the height of the grid, meaning $t_{max} = m \cdot n$, for $m$, $n$ from $M^{m \times n}$. This time limit was chosen so that a single agent should be able to traverse a map with the size of $m \times n$ cells at a maximum of $t_{max}$ steps, as long as no obstacles are present. As a consequence an algorithm using multiple agents should be faster than $t_{max}$, even if obstacles are present. Otherwise using multiple agents would not be beneficial.

To compare the results of the MAF algorithm, the Brick&Mortar algorithm was implemented and also simulated. Furthermore to see how the number of agents contribute to the result of the algorithm, the simulations started using only one agent for each of the algorithms. Additional runs then incremented the number of agents by one until the count of agents reached 20. Each simulation configuration, composed out of the map, the number of agents, and the type of algorithm, was simulated 30 times to create an average of all results.

### 3.2.3. Selection of the simulated Maps

As the primary goal of the Multi-Agent Flood algorithm is to support search and rescue teams in urban search and rescue scenarios, the maps for the simulation were chosen to represent typical urban terrains. Additionally different maps were chosen to showcase simple scenarios and maybe structural difficulties, that may hinder agents depending on the obstacles provided in the map.

For this reason the MAF algorithm was simulated on five different maps, starting with two plain maps, without any obstacles and only the border to confine the agents in the scenario. The only differences between these two maps were the size of the maps.

The third map, shown in Figure 3.8, was chosen to represent very small paths were the agents could only move forward or backward, trying to trip them up, as the markings of different agents could block the movement

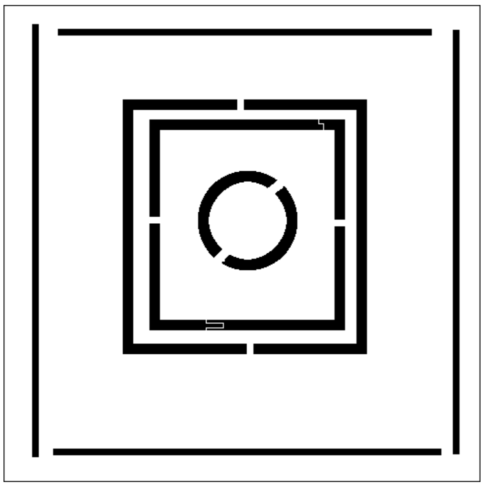
Figure 3.8.: Scenario Plain with obstacles, 500×500 cells

through these narrow gaps.

The fourth and fifth map on the other hand were based on the layout of a house or a street of houses, respectively depicted in Figure 3.9 and 3.10.

The size of the maps was deliberately chosen to be on the bigger side. The algorithms described in the Chapter 2 of this work were usually simulated on rather small maps. The Ant algorithm provides results for tests done on a map containing 40×30 cells (Koenig and Liu, 2001, p. 4). Ferranti et al. tested the Brick&Mortar algorithm by using a map with 2500 cells or 50×50 cells (Ferranti et al., 2007, p. 6). Also the improved version of their work, the HybridExploration algorithm, was only run on maps using the same size (Ferranti et al., 2009, p. 226). To show that the MAF algorithm is able to be used in bigger scenarios the minimum map size of 500×500 cells was selected, resulting in 250.000 cells for the whole map, represented in the smaller *Plain* variant, *Obstacles*, and *House*. Additionally bigger maps were chosen to showcase that the algorithm is able to also solve these kind of scenarios. These two maps, *Street of Houses* and the bigger *Plain* variant, have a size of 1000×1000 cells, or 1.000.000 cells overall.

These sizes also limit the runtime of the MAF algorithm, as described in the simulation setup subsection. The simulation of the algorithm on the smaller maps will terminate at a maximum of 250.000 steps and on the bigger map the algorithm will shut down as soon as 1.000.000 steps are reached. Those two numbers are the maximum allowed number of steps that the MAF

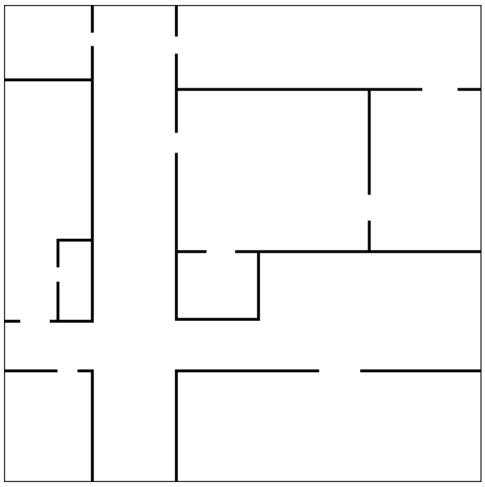
Figure 3.9.: Scenario House, 500×500 cells

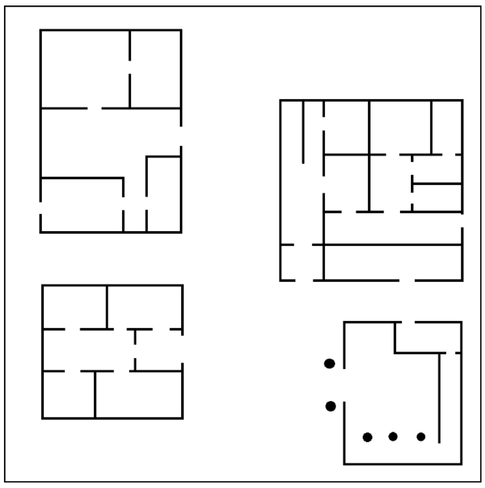
Figure 3.10.: Scenario Street of Houses, 1000×1000 cells

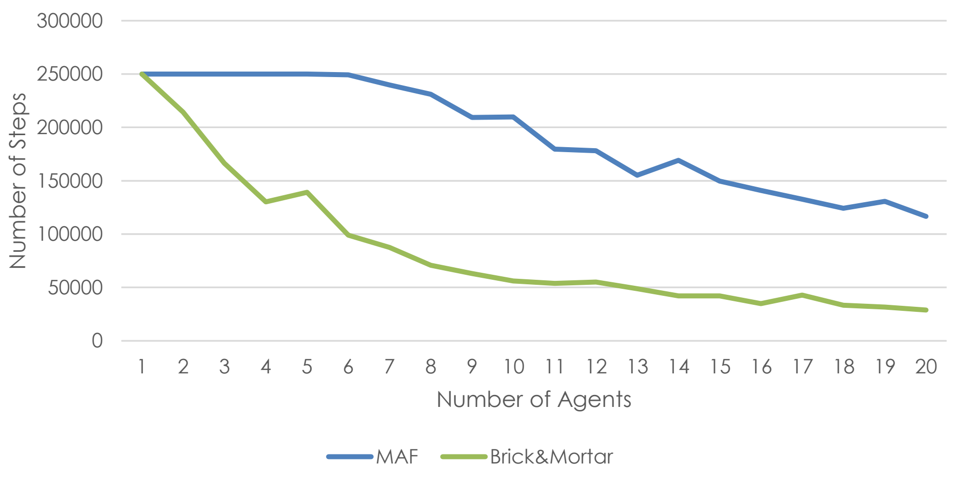

Figure 3.11.: Results MAF and Brick&Mortar, House, 500×500 cells

algorithm is allowed to run. Naturally if it finishes before these limits are reached the algorithm will terminate successfully.

Each map has a finite number of points of interest, or victims in this case, uniformly distributed in any traversable cell in the scenario (there are no points of interest in the starting point). The smaller maps have ten points of interest, while the bigger map has 15 points of interest.

### 3.2.4. Results of the Simulation

The four diagrams illustrated in Figures 3.11 to 3.15 present the results of the simulation of the Multi-Agent Flood algorithm compared to the simulated results of the Brick&Mortar algorithm. The X-axis in these diagrams depict the number of agents and the Y-axis delineates the number of steps. The lines, one for each of the simulated algorithms, will show the steps it took for this specific number of agents to finish the algorithm. The blue line portrays the MAF algorithm and the green line the Brick&Mortar algorithm. A number of steps below the runtime limit will be considered as a successful run of the algorithm. This means for the Multi-Agent Flood algorithm that the agents have found at least 95% of the points of interest and have explored at least 95% of all traversable terrain. The Brick&Mortar algorithm will terminate as soon as the step limit is reached or all the agents will stop moving.

As the diagrams show, depending on the scenario, at least three to five

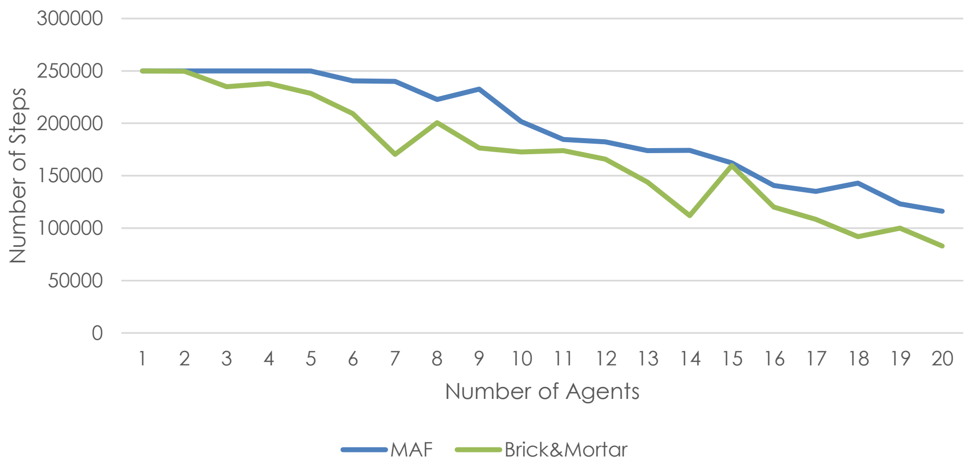

Figure 3.12.: Results MAF and Brick&Mortar, Obstacles, 500×500 cells

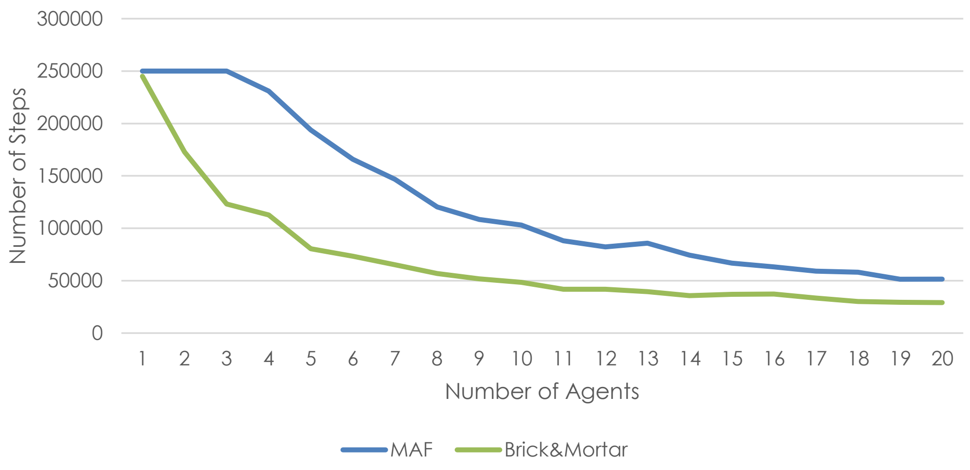

Figure 3.13.: Results MAF and Brick&Mortar, Plain, 500×500 cells

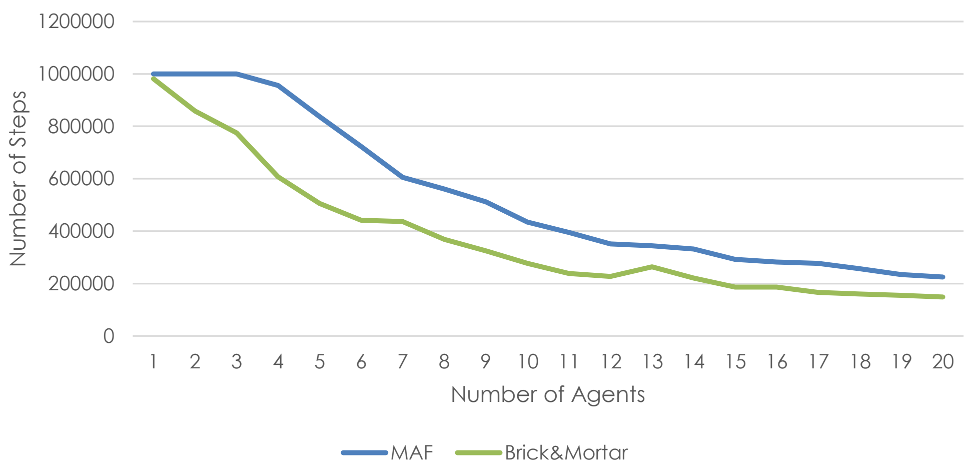

Figure 3.14.: Results MAF and Brick&Mortar, Plain, 1000×1000 cells

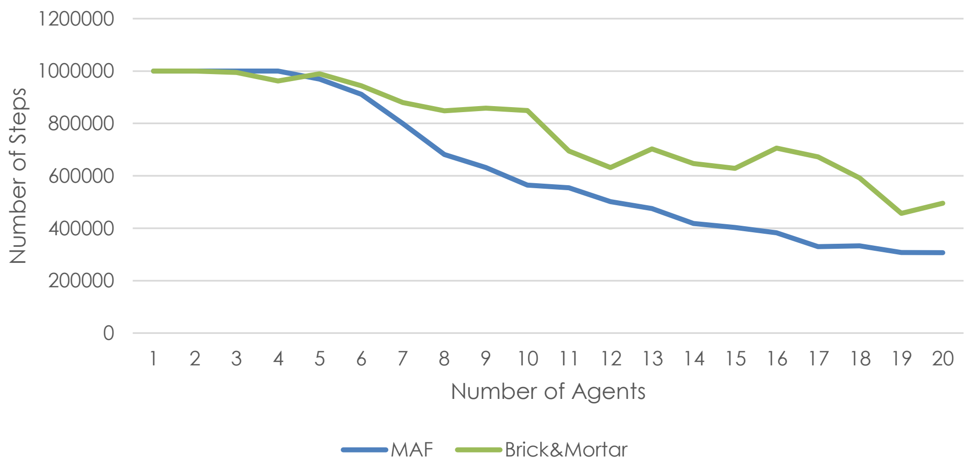

Figure 3.15.: Results MAF and Brick&Mortar, Street of Houses, 1000×1000 cells

agents are needed for the Multi-Agent Flood algorithm to successfully finish the simulation. In comparison the Brick&Mortar algorithm usually terminates below the runtime limit as soon as two agents are available. Adding further agents will improve the search time, shortening the number of steps that it takes for the algorithms to accomplish their goal.

The diagrams in Figures 3.11-3.15 show that both algorithms will reach their search targets as soon as enough agents are available. One can also see that the results from the Multi-Agent Flood algorithm do not deviate that much from the results of the Brick&Mortar method. Although the MAF agents need some more steps to finish successfully in four of five cases. In the fifth case, illustrated in Figure 3.10, the MAF algorithm is faster in the comparison. This is especially interesting as the movement routine of the MAF agents is quite simple and not in any way optimized. Additionally as soon as an agent finds a victim, the mode switch will get triggered and the agent will stop any further exploration and searching until it has returned to the starting point, from which it will start the search anew. The Brick&Mortar algorithm does not have this “drawback” and the agents can continue to move to unexplored terrain. The results reinforce the statement made by Ferranti and his coauthors in Ferranti et al. (2007) that the loop detection and handling mechanism in the algorithm is costly and increases the time that it takes to finish the search process massively. Specifically as the map, Figure 3.10, used in this simulation configuration contains many loops.

If the results of the Multi-Agent Flood algorithm for all five scenarios are taken together, it can be seen, that the algorithm scales very well, almost independently from the structures on the different maps and more important independent from the size of the map. This is depicted in Figure 3.16, the diagram shows the normalized results of all five simulations for the MAF algorithm, where the X-axis represents again the number of agents used and the Y-axis the normalized number of steps it took the algorithm to finish the calculation. Only five agents at a minimum are needed to successfully terminate the algorithm and find 95% of victims, while exploring at least 95% of all available and traversable cells.

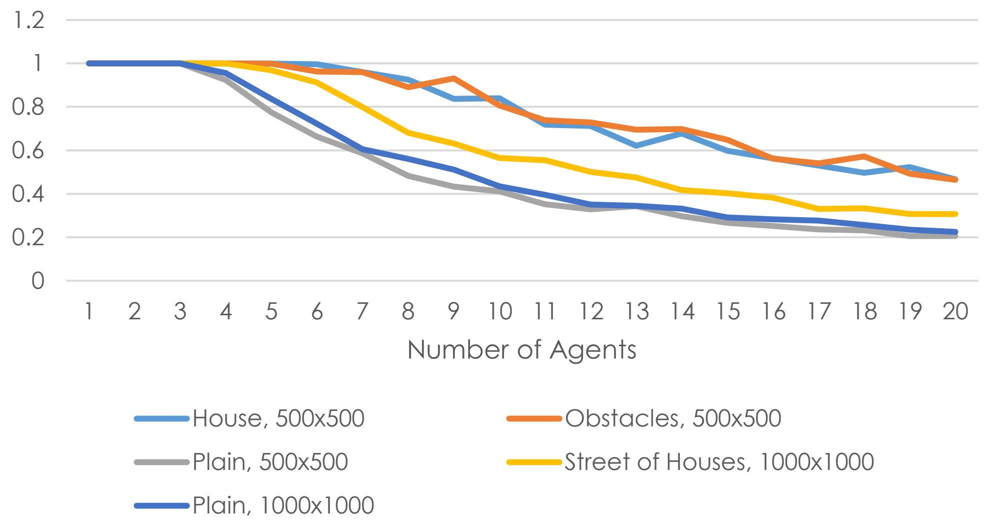

Figure 3.16.: Normalized results of the simulation of the MAF algorithm

## 3.3. Advantages and Disadvantages of the Multi-Agent Flood algorithm

The simulations have shown that this algorithm is able to successfully explore unknown terrain and find possible victims in an urban search and rescue scenario. Specifically the additional movement of the agents by returning to the base to communicate the information about a found victim does not really interfere with the runtime of the algorithm.

The possibility to return to the headquarter of the search and rescue team at any time allows for the maintenance of the agents: repairs can be made, batteries charged, and information exchanged. The main advantage of this approach is the ability to start the rescue process in parallel with the search process.

These results are based on a very simple algorithm, where the agents are not really sophisticated. The movement algorithm is intentionally kept uncomplicated, as not that much computational power is needed to keep these agents running. The algorithm does not depend on wireless communication that may not be possible in these specific scenarios, following the three design constraints stated by Robin Murphy, (Murphy, 2014, p. 5), that rescue robots should follow:

| Name | Goal | Communi-cation | Movement | Information back to HQ | Maintain-ability | Size |
|---|---|---|---|---|---|---|
| Ants | PoIs | Indirect | Grid-based | ✗ | ✗ | ✗ |
| Brick&Mortar | Explore | Indirect | Grid-based | ✗ | ✗ | ✗ |
| HybridExploration | Explore | Indirect | Grid-based | ✗ | ✗ | ✗ |
| Rendezvous | Explore | Direct | Grid-less | ✓ | ✗ | ✗ |
| MAF | Both | Indirect | Grid-based | ✓ | ✓ | ✗ |

Table 3.1.: Comparison of the different algorithms, with MAF added to the list

1. The robot should be able to function in extreme terrain and operating conditions
2. Operate in GPS- and wireless-denied environments
3. Provide appropriate human-robot interaction for the operators and the victims

The first rule is accomplished by being using already existing hardware, for example building on robots described in Chapter 2. By only using indirect communication and no global positioning system, rule number two is satisfied. The data exchange between the agents and the human rescue team in the headquarter, after an agent has returned to the starting point during the *Return Mode*, also helps to realize the second rule. The third point is also based on the information exchange at the headquarter.

Table 3.1 is the same as Table 2.1 from Chapter 2 but with the Multi-Agent Flood algorithm added to the end. The *goal* of the MAF algorithm is a combination of exploring the unknown terrain and finding the points of interest dispersed in the scenario. Additionally the MAF algorithm is the only algorithm, which offers a possible maintainability of the agents. It is also one of the two algorithms that will communicate gathered information back to the base during the runtime of the algorithm.

Nevertheless this version of the algorithm still offers many possibilities of improvement. Additional data exchange could speed up the search process further, decreasing the runtime and allowing for a shorter execution of the algorithm. This can be achieved by adding more data to the markings in the environment or by making the agents more “intelligent”.

CHAPTER 4.

# IDEAS FOR A LAYERED COMMUNICATION

Chapter 3 described the Multi-Agent Flood algorithm and showed that it is a feasible solution for the use of a multi-agent system in an urban search and rescue scenario. The last section of the previous chapter also provided a short conclusion about the advantages and disadvantages of the method. Especially as the simulation results presented for the MAF algorithm revealed that, while the algorithm is viable and practicable, it was only faster in one of five times than the benchmark algorithm. This chapter will offer some improvements that try to aim to speed up the search process of the Multi-Agent Flood algorithm by providing additional data exchange between the agents. Due to diverse restrictions present in the typical urban search and rescue scenario, codified in the second rule by Murphy in (Murphy, 2014, p. 7), using direct communication will not reliably work:

> "Rescue robots typically function in **GPS- and wireless-denied environments**. The material density of commercial buildings interferes with GPS and wireles networks for robots working in the interiors of commercial buildings. However, interference is not limited to interiors; urban structures such as buildings or bridges can create shadows or multiple paths that affect approaching UAVs and UMVs. . . Wireless communication can be boosted with more power, but that leads to a trade-off between making the robot larger to carry the power and it becoming too large to use."

Currently the agents of the MAF algorithm only rely on indirect communication. This is achieved by marking the environment, or as an abstraction the cell in the grid, where the agent is currently located. While this type of data transfer is viable it is in no way optimal. An agent is required to be in the cell or in a neighboring cell to read this marking, thus the agent has to move to or near a specific cell to access the information stored there. This in turn adds increased movement to each of the agents. Resulting in a longer search phase of the algorithm, which is diametrical to the design goal. As the first 48 to 72 hours after a disaster offer the highest chance to find victims still alive, the algorithm has to find as many victims as possible in this time frame.

To improve the runtime of the Multi-Agent Flood algorithm an increased data transfer is needed. On one hand additional data can be stored in the markings, but as explained above, this would only help the algorithm partially. Another possibility would be that agents should be able to exchange data wirelessly. As explained by Murphy, wireless communication my not always be possible. As a consequence implementing wireless transmission has to be done in a specific way or has to adhere to specific restrictions. The algorithm must not depend on this data transfer, if wireless communication is not possible, the algorithm still needs a way to function and to finish the search successfully. Hence additional information gained by communicating by radio signals should be seen as bonus information. With the use of this extra data the algorithm should speed up the search process, but as soon as this input is not available anymore the agents should still be able to finish the run, even if it is at a slower pace.

The Rendezvous algorithm [De Hoog et al. (2010); Spirin and Cameron (2014)] introduced in Chapter 2 relies on an ad-hoc network created by at least two agents, the *Explorer* and the *Relay*. These two agents will meet at specific points in the map, called rendezvous points, giving the algorithm its name. While the agents meet, they are near enough so that they can initiate a wireless data transfer. Murphy describes, in the quote written above, how structures and hazards can hinder the radio waves. By moving the agents near each other, resulting in a line of sight communication, most of these interferences can be safely ignored and the assumption can be made, that as soon as there are no obstacles between two agents and the range between said agents is not too big, wireless communication is possible.

This chapter will describe how wireless communication can be added to the Multi-Agent Flood algorithm, while still keeping the indirect communication between the agents via markings on the floor. This concept of using two communication models and a way to ignore the direct communication part if it is not working at the moment is called layered communication and offers the best of both types of data transfer. Especially as it allows for a graceful degradation of the type of communication, if wireless data transmission should not be available anymore. Thus allowing the MAF agents to finish the search process either way. The whole concept of robust layered communication was first introduced by the author in Becker et al. (2014).

## 4.1. Direct and Indirect Communication

The two different types of communication are direct and indirect communication. Both types offer particular advantages and disadvantages. The first type relies on radio or light signals to transmit data from one agent to the other. The second type changes the environment for other agents to read and exchanges data this way. This section will give a short overview over the two types of information exchange and how each of them is achieved.

### 4.1.1. Ad-hoc Networks

In the simplest case two agents will find each other and will initiate a data transfer between themselves, using for example typical WiFi hardware. Direct communication offers the advantage to create a whole network of agents who are able to transfer data between each member of the network, using the members as relay nodes to send the data onwards from one agent to the other until it arrives at the right recipient. This network creation out of the blue is called an ad-hoc network. Each agent represents a node in this network and is able to forward messages from its neighbors.

This approach offers a lot of flexibility, as agents can join or leave the network as soon as they enter the range of another agent, or move out of the range respectively. On the other hand this type of network may not be very reliable, as the nodes of an ad-hoc network usually form a spanning tree. This means, if an agent from the middle of the tree moves out of range the tree will be halved, resulting in a loss of communication, as the first half cannot communicate anymore with the second half until another agent replaces the

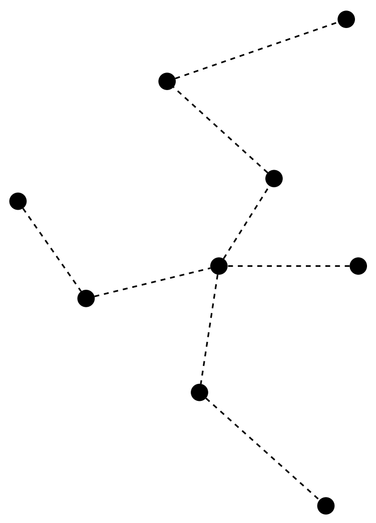

Figure 4.1.: Example of a spanning tree

leaving agent.

The spanning tree approach is fundamental for a lot of ad-hoc network algorithms. An example is provided in Figure 4.1. One can easily see, that the tree is not really robust and the controlling algorithm of the agents needs to keep the agents in range of each other, constricting the movement of each single agent to keep the ad-hoc network intact. Nevertheless this approach is often used in various multi-agent algorithms to create an ad-hoc network [Nguyen et al. (2004); Rekleitis et al. (2004); Rooker and Birk (2005); Witkowski et al. (2008); Takahashi et al. (2012)].

The method presented in Rooker and Birk (2005) keeps the “traditional” spanning tree approach in its multi-agent algorithm, others try different ways to create more robust ad-hoc networks. Ioannis Rekleitis and others try to create a line of robots, Rekleitis et al. (2004), reminding of the typical search line used by human search teams. They are also using some additional robots to circumvent obstacles, these robots are also sending information about these obstacles back to the search line. This idea is based on the Boustrophedon decomposition and agents can only communicate with each other as long as they do have a line of sight. Nguyen proposed an idea in Nguyen et al. (2004), that was later used for the Rendezvous algorithm. He and his coauthors introduced additional relay robots, that act as nodes of the spanning tree, allowing the exploring robots to move further out. Witkowski

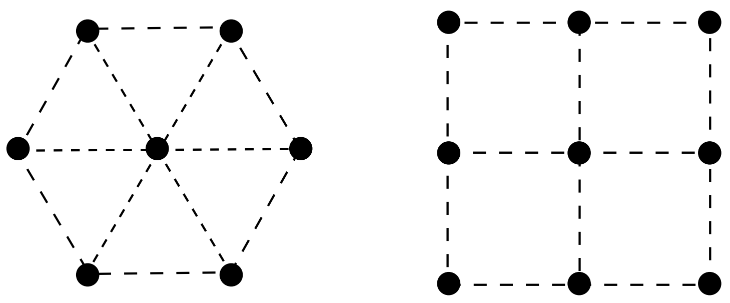

Figure 4.2.: Example of a triangular network on the left side, rectangular search pattern on the right

et al. improved on this idea, but instead of using a spanning tree, the agents form a triangular network between themselves, Witkowski et al. (2008). An example is depicted in Figure 4.2 on the left side.

A similar idea was proposed by Toru Takahashi in Takahashi et al. (2012). He proposed two different versions of a multi-agent system using an ad-hoc network for communication. The first version also declares some rendezvous points, where agents will meet to exchange information. The second version tries to create a rectangular search pattern, shown in the right side of Figure 4.2. The range of this pattern depends on the range of the communication between the different agents. Nevertheless his proposal sacrifices free movement for communication stability.

Another approach to enhance the use of direct communication, especially if there a few nodes in an ad-hoc network or only a few messages are send between those nodes, is the use of specific ad-hoc routing protocols. For example the gossip/rumor protocol family or flooding protocols. This means that the use of the protocol depends on the number of agents in a specific area and how these agents will move. Compare Becker et al. (2007) for an analysis of various routing protocols and how they perform in sparse communication networks.

On the other hand, if sufficient wireless communication is possible between the agents, and there are enough nodes available, then more refined routing protocols can be used. These may include the proactive protocols OLSR/DSDV or on demand protocols AODV/DSR.

This short overview shows, that a lot of different algorithms and protocols are available to support an ad-hoc network sufficiently. But the main drawback remains: if an agent is not in range of the other agents and can not join the network it may not be able to function at all or with a reduced efficiency.

### 4.1.2. Stigmergy

This term was already mentioned in Chapter 2 of this work. It describes the indirect communication between two agents through modification of the environment.

The differences between stigmergy and ad-hoc networks are profound. Ad-hoc networks allow for almost instantaneous data transfer between two agents. Even if larger chunks of data are transmitted, the transmission speed is staggering, going up to 6.77 Gbit/s using the newest IEEE 802.11ac standard, which is equivalent to 846.25 Megabyte every second, Verma et al. (2013). While the indirect communication does not offer this sort of data transmission rates, it has other advantages.

The use of indirect information exchange is very decentralized, meaning that agents do not have to keep track of each other and, contrary to the direct communication, they do not have to form a network to achieve cooperation. Each agent is able to mark the environment and leave this marking for any other agent to read. This frees the agents from the inherent movement constraints enforced by the ad-hoc network. The downside of this decentralized communication is the time that it takes one agent to get the information from another agent. For another agent to read the mark of a specific agent, the first one has to find said mark in the first place. This may take a while or, under certain circumstances, this may take forever. Another disadvantage is the amount of information that can be stored in a mark. This largely depends on the technique used to mark the environment. The work in Koenig and Liu (2001) describes how agents can use a chemical marker. This restricts the amount of information severely. The only way information can be stored using this sort of marking is the intensity of the chemical marker. This allows for a small range of numbers to be encoded, using a small intensity for one boundary and a very big intensity for the other end of the range. A similar way to mark things would be to spray some kind of bar code on surfaces, but which may be harder for other agents to find again.

A more sophisticated approach is the use of Radio-frequency identification

(RFID) chips. The multi-agent algorithms from Ferranti et al. are using this method to create an indirect communication between their agents, Ferranti et al. (2007, 2009). Toshiki Sakakibara and coauthors show in Sakakibara et al. (2007), and Vittorio Ziparo and others describe in Ziparo et al. (2007) how RFID chips can be used as markings in the environment and how the agents can drop them at regular intervals to exchange the data stored in them. In addition to Ettore Ferranti, Marco Baglietto proposed a multi-robot coordination system based on RFID chips, Baglietto et al. (2009).

The advantage of RFID chips is the variability that the different types of chips offer and the size of the chips. The types range from completely passive to active chips using their own batteries. This also shows in the size of the chips. Some are as small as a rice kernel, others are a lot bigger, needing the space for a battery and other additional equipment. There are currently five different classes of RFID chips available, taken from Weis (2007) and Sen et al. (2009):

1. completely passive, smallest available class of chips, range of some centimeters

2. passive, including a read-only chip, effective range is also only some centimeters

3. offers read and writable storage in the size of some kilobytes, can still only be accessed in the range of some centimeters

4. semi-passive, first class to use a battery, which allows for higher storage values and higher access range (up to 4 meters)

5. active, permanently powered by battery, through powering the antenna the access range is raised to about 150 meters

Although the class 5 chips would offer the best of both worlds, increased storage and range, the main drawback is the same that applies to conventional direct communication: the range can not be guaranteed. Furthermore these chips are the largest chips available, decreasing the number of chips that an agent could carry around. For this reason the use of class 4 chips offer the most practicable solution. Especially as the quantity of the data that needs to be exchanged in most algorithms is not that big. As the previously introduced algorithms that propose the use of RFID chips have shown, this data is usually in the range of some integers. The Brick&Mortar algorithm stores the state of the cell in the chip and the Multi-Agent Flood algorithm only needs to store a step counter. Only the HybridExploration algorithm uses active chips to allow for a communication between the chips themselves.

## 4.2. Layered Communication

The idea behind the concept of layered communication is to create a combination of direct and indirect communication. Thus trying to get the advantages from both types of information transfers, while negating the disadvantages, as described by the author in Becker et al. (2014) and Blatt et al. (2015). Direct communication offers fast exchange of a lot of information, while suffering from outside influences created by the environment. Indirect communication, on the other hand, allows for a robust exchange of information, although the time of the exchange can not be guaranteed. A combination of both approaches should result in a communication that enables the agents to transmit a lot of data in a small time frame, but if this sort of transmission is not possible, reverts to the use of indirect communication to still allow the agents to finish their mission. Figure 4.3 illustrates an example of this. On the left side is a spanning tree, usually found in ad-hoc networks. Each node represents a single agent. The middle picture shows agents moving out of range of the base tree, being able to explore without having to rely on the tree structure. The third picture on the right depicts how additional networks can be formed between other agents, thus allowing the creation of multiple networks, if these agents are not in range of the rest of the agents. This decentralized behavior allows each agent maximum freedom of movement, while still keeping the advantages of creating communication structures on the fly.

By adapting this concept to the Multi-Agent Flood algorithm only the direct communication part of the concept has to be realised. The indirect part can be used as it is and serves as a fallback mechanism, in the case that the direct communication is not possible or only rarely available. This means, that the agents will still disperse their markings in the terrain, using them to cooperate and to mark the currently shortest known path from any position

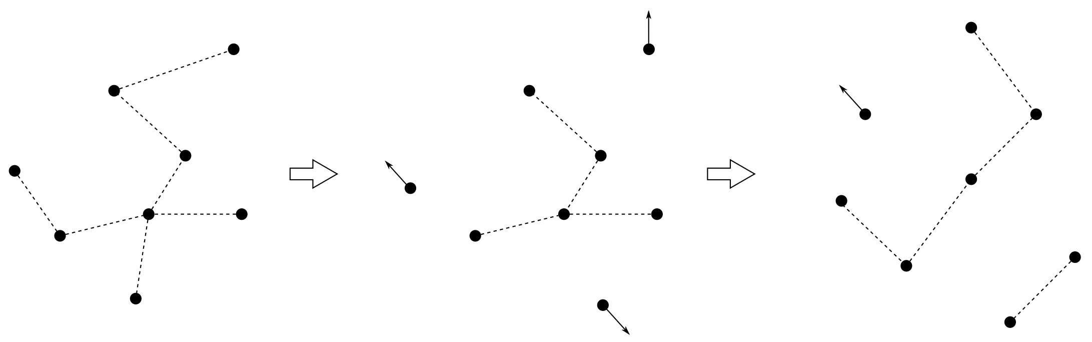
Figure 4.3.: Example of a multi-agent system using direct and indirect communication

back to the starting point. The ability to communicate directly between the agents offers the possibility to exchange additional data, for example maps created during the exploration of the unknown terrain. The agents can use their sensor data to create maps during the exploration of the environment. This is called Simultaneous Localization and Mapping (SLAM). These maps can then be shared between the agents, increasing the amount of knowledge that an agent has about the terrain, without ever visiting some parts of it.

Simultaneous Localization and Mapping is crucial for robots without any access to a global positioning system. The original version of the Multi-Agent Flood algorithm did not store any information about the environment and the agents could only act on the currently available sensor data from the immediate neighborhood. Different approaches to solve this problem already exists, see for example Grisetti et al. (2010).

If two agents are now in range to initiate direct communication they will be able to exchange the collected map data and the agents can merge their own map with the newly received information, as long as one point on each of the maps is the same, see Pfingsthorn et al. (2008). The point on each map that is always at the same coordinates for each agent is the starting point. This is also the reason, why multiple starting points would not be really feasible using this approach.

The ANSI committee defines *robustness* as follows, ANSI/IEEE (1991):

> ... the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.

The use of both types of communication also offers this robustness to the MAF algorithm, as it does not depend on one sort of communication anymore. By allowing a graceful degradation of the direct communication, should it not be available anymore, the algorithm still has the means to finish its calculation and will be able to terminate successfully. The first part, invalid input, can be ignored for the application to this concept. Much more interesting is the second part: "stressful environmental conditions". If the direct part of the combined communication is not available anymore, the indirect will still work.

### 4.2.1. Adapting the Multi-Agent Flood algorithm

To adapt the Multi-Agent Flood algorithm to work with layered communication the agents simply need to include a WiFi capability and they have to be able to use SLAM techniques to create maps based on their sensor inputs and store these maps. As the environment in this model is represented as a grid, no specific algorithms will be needed. The agents will simply store the environment of their current position in an internal map. Additionally any other useful information can be stored and forwarded through this system.

This means, that the usual behavior of the agents will not change. They will continue to mark cells with their step counter and use these markings if no other information is available. The new communication capabilities will come into play as soon as two agents are in a specific range and have established a line of sight. These two restrictions enable a guaranteed data transmission using radio signals between two agents. This adaption also changes the way how the navigation process works for each agent, as it now has more data available and can make more informed decisions.

The main advantage of this approach is, that the extra information provided by the data transmission between the agents is only seen as a bonus and the agents are still able to find currently known shortest paths from their positions back to the starting points without having access to current map data. Even if an agent should not encounter another agent in the whole runtime of the algorithm it will still be able to function. Another asset is that agents will return to the base from time to time, allowing the rescue team at the base to not only get information about a recently found victim but also to get an update of the map currently stored in these agents. This allows the base to merge all map data it receives from the agents and in turn it can also start to distribute this data to the outgoing agents again. Thus even if an agent will never encounter another agent during the execution of the algorithm, it will gain access to shared map data as soon as it returns to the starting point.

**Algorithm 5** Protocol for the use of the direct communication part

```
 1: table recentCommunication;
 2: **if** another agent B in range **then**
 3:    **if** B $\notin$ recentCommunication **then**
 4:       initiate communication;
 5:       exchange data;
 6:       recentCommunication.add(B,timestamp);
 7:       merge maps;
 8:    **else if** B $\in$ recentCommunication **and** recentCommunication.get(B) +
       cooldown < currentTime **then**
 9:       recentCommunication.remove(B);
10:    **end if**
11: **end if**
```

The code provided in Algorithm 5 gives an overview how the communication protocol between the agents $A$ and $B$ work. Each agent will check, whether another agent is in communication range. If this is the case, each agent will check his table of recent communications, checking if the new found agent is already in this table. If there is no entry representing the agent $B$ in the table, then agent $A$ will initiate communication with $B$ and a map exchange will occur. $A$ will then add $B$ to its table including a time stamp, $B$ respectively will do the same for $A$.

If $B$ is already represented in $A$'s table, then $A$ will check the time stamp stored in the table and will compare it with the current time. In the case that the time stamp plus a additional cooldown time is equal or less the current time, $A$ will remove the entry about $B$ from its table. This will stop the same agents to always initiate a map exchange as long as they are near each other, until some time has elapsed and the changes in the internal maps may be significant enough to warrant another data transmission. The enforced pause between the same two agents will keep the count of map merges down and enables the agent to process multiples merges from multiple agents without getting an information overload through the request for numerous map merges.

This approach can by easily expanded to use one of the various ad-hoc network algorithms to disseminate the stored information from one agent to all other agents that are in range of each other. Thus information assembled by agent $A$ can be forwarded through agents $B$, $C$, and $D$ until it finally reaches agent $E$. This means in turn that the map data from $A$ will be shared with four other agents, although $A$ may be only in range of $B$, as long as $A$ and $B$ are part of an ad-hoc network connecting all five agents together at the same time. The use of possible ad-hoc networks allows for a faster dispersion of information through the agents without the need of two agents to be in range of each other, as in this example agents $A$ and $E$.

The communication between agents will happen in both modes: *Search Mode* and *Return Mode*. Although the map information will only be used during the return to the starting point. The returning agent will be able to compute the currently shortest known path from its internal map, either using a flood algorithm or other path planning algorithms, like Dijkstra, A*, et cetera, see also Cherkassky et al. (1996). Additionally as the agent follows the computed path back to the starting point, it will still read the markings that it will encounter on its way and update the map with new found information, that may differ from the data stored in the map. Using this new information will lead to a recalculation of the path.

## 4.3. Simulating the layered communication algorithm

For the simulation of the layered communication algorithm the assumptions made for the wireless communication to work have to be specified. This means that agents will only be able to initiate a transmission if they have a line-of-sight connection. Although a radio signal is able to travel through walls or obstacles, or even reflect around obstacles, simulating this would be very complicated. As the behavior of waves depends on the material and the surface of the walls and obstacles, the simulation would have to know which wall consists of which type of material and has which kind of surface. These attributes also affect the range of the radio emitter. Using a WiFi connection on an open field would allow for a much farther range than using the same connection in a rubble strewed building. To simplify these calculations the

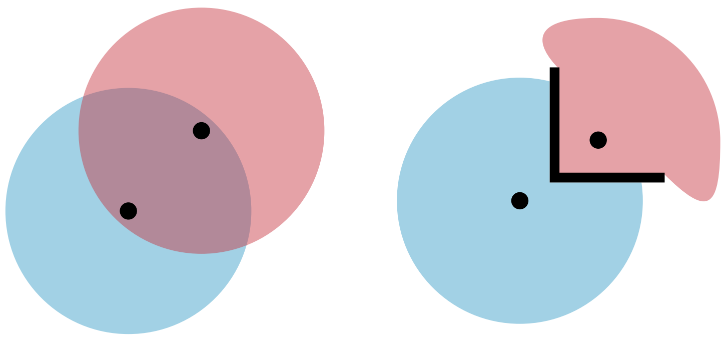
Figure 4.4.: Example of the radio emitter range of modified agents

signal strength is assumed to be the same in every direction, resulting in the distance being represented as the euclidean distance between two agents. As the agents are operating on a two dimensional map, this is simply calculated using the following function:

$$distance(p_1, p_2) = \sqrt{(p_2.x - p_1.x)^2 + (p_2.y - p_1.y)^2}$$

In the simulation communication using WiFi is allowed, as soon as the distance between two agents is smaller or equal to five cells. An example of the ranges of the emitter of modified agents is presented in Figure 4.4.

This way allows the simulation to simply ignore any reflection, refraction, or interference of any radio waves. Further a high enough throughput is assumed for the direct data transmission. Using the WiFi protocol standard 802.11g provides a minimum throughput of 6 MBit per second up to a maximum throughput of 150 MBit per second if the current standard, 802.11n, is used, see [IEEE (2003, 2009)].

During the simulation the agents will now be able to build a map of their sensed surroundings. As the map in the simulation is based on a matrix calculating a merge of two maps is a rather simple process. The common starting point is used to pin the two matrices together. If the algorithm uses real sensor input and builds a map based on these inputs by applying the SLAM techniques a map merging will be possible using already existing algorithms, for example one proposed by Pfingsthorn et al. (2008).

These changes to the agents can be simply achieved by adding a WiFi module to a real robot and increasing the computation power. The onboard processor should be fast enough to merge maps in a reasonable time and the

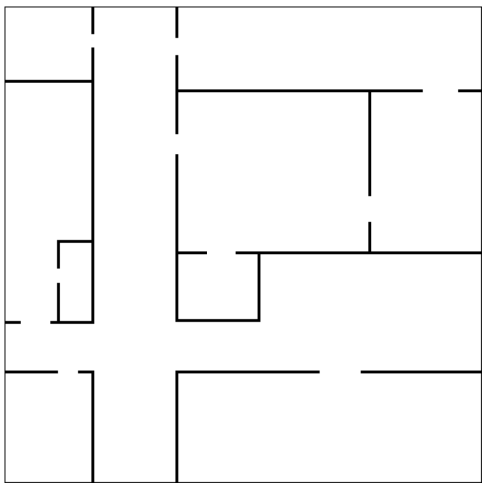

Figure 4.5.: Scenario House, 500×500 cells

data storage should be big enough to hold enough data for the map.

## 4.3.1. Changes to the experimental simulation setup

The base experimental simulation setup is the same as the one described in the third chapter. Four new test cases were chosen to simulate the layered communication version of the Multi-Agent Flood algorithm, these are illustrated in Figure 4.5-4.8. These test cases were picked again due to the similarity to typical urban terrains.

For the comparison of this version of the algorithm also the indirect communication version was simulated and an additional version using only direct communication was run. The termination rules of the algorithms stay the same as those described in the third chapter: finish successfully if 95% of the traversable cells are explored and if 95% of the victims are found. Otherwise the runtime is capped based on the size of the used map (250.000 steps and 1.000.000 steps respectively). As in the previous simulation ten points of interest were dispersed in the smaller maps and 15 points of interest in the bigger map.

Again the three algorithms were simulated in different configurations. Each simulation was started by using one agent and going up to twenty agents. As before each configuration was simulated thirty times and the results are the average of these calculations.

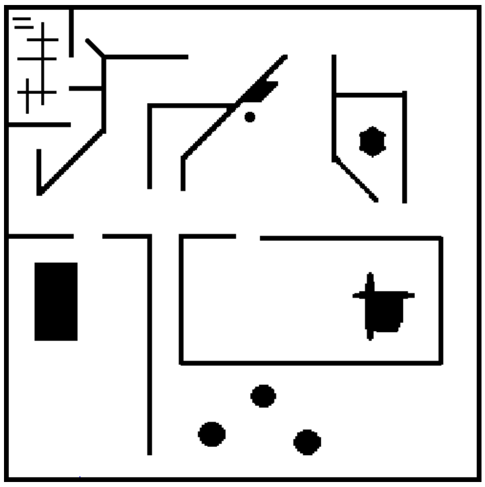
Figure 4.6.: Scenario Office, 500×500 cells

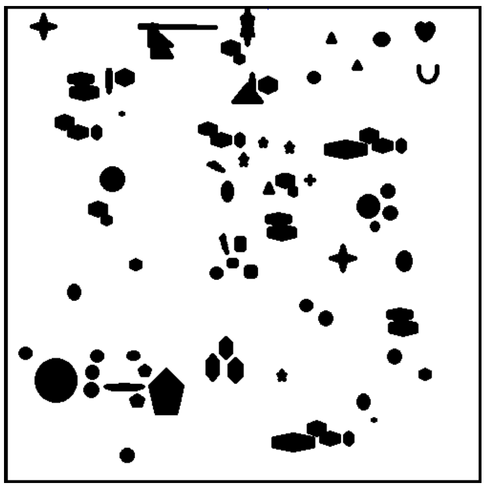
Figure 4.7.: Scenario Park, 500×500 cells

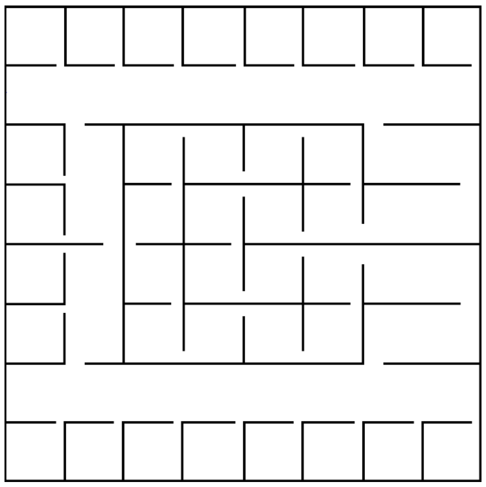

Figure 4.8.: Scenario Cubicles, 1000×1000 cells

## 4.4. Comparison of the algorithms

The diagrams in Figures 4.9 to 4.12 illustrate the results of the simulation. As before the X-axis depicts the number of agents used in this configuration and the Y-axis the average number of steps it took for each algorithm to finish. The blue line shows the indirect variant, the yellow line the direct algorithm and the green line represents the results of the layered algorithm, usually titled as both.

The indirect version of the Multi-Agent Flood algorithm is able to terminate successfully in each test case but the last, as shown in Figure 4.12. On average it was able to do this using around half the allotted time. The diagrams also show that the indirect version of the algorithm needed at least five or even ten agents to explore at least 95% of the map and find 95% of the victims distributed through the terrain. The algorithm was not able to terminate successfully in the big map, Cubicles, reliably, as even twenty agents were not enough to explore enough of the unknown terrain for the algorithm to end without reaching the maximum number of allowed steps.

The direct algorithm based on direct communication, simply provided for the comparison, was faster than the version using only indirect communication. This result was expected, but using only direct communication is not really feasible in the scenario and was only simulated for the contrast and the comparison. This algorithm was able to finish the calculation in a fourth of

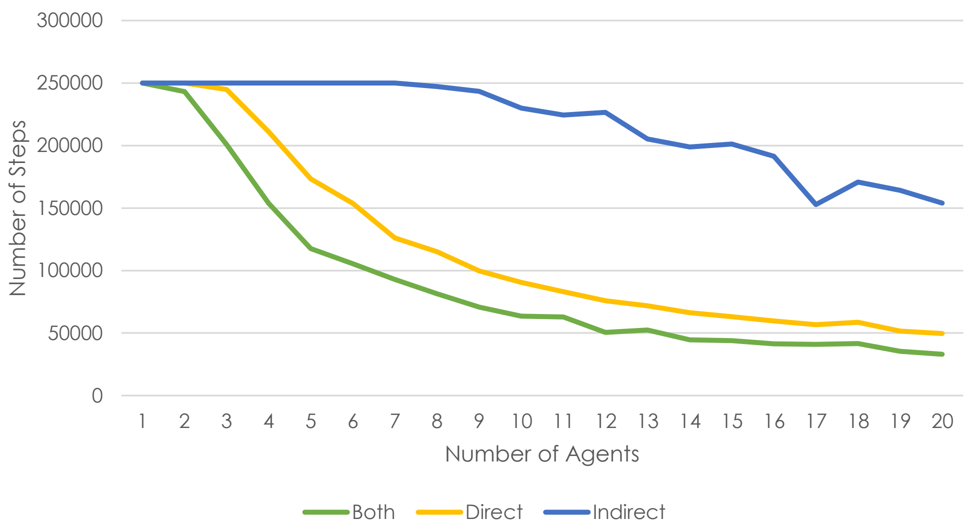

Figure 4.9.: Results of the MAF variants, House, 500×500 cells

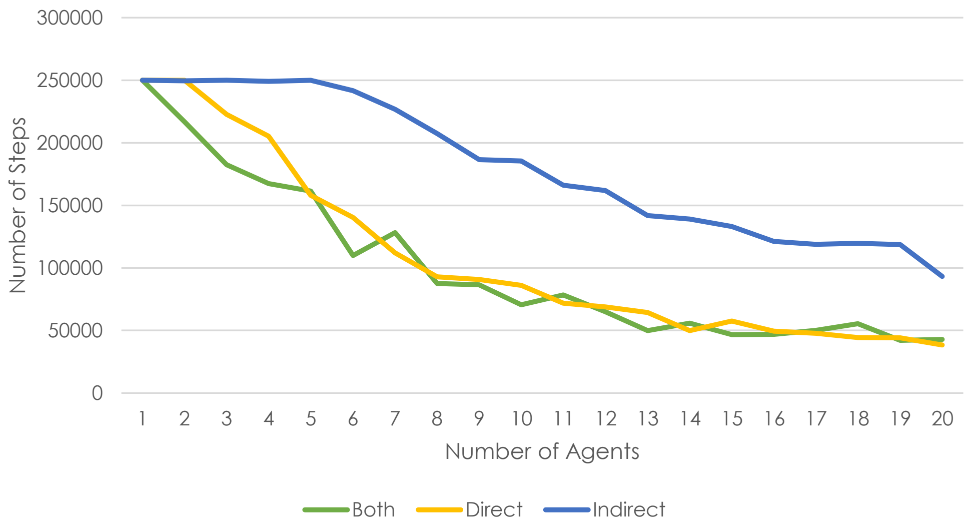

Figure 4.10.: Results of the MAF variants, Obstacles, 500×500 cells

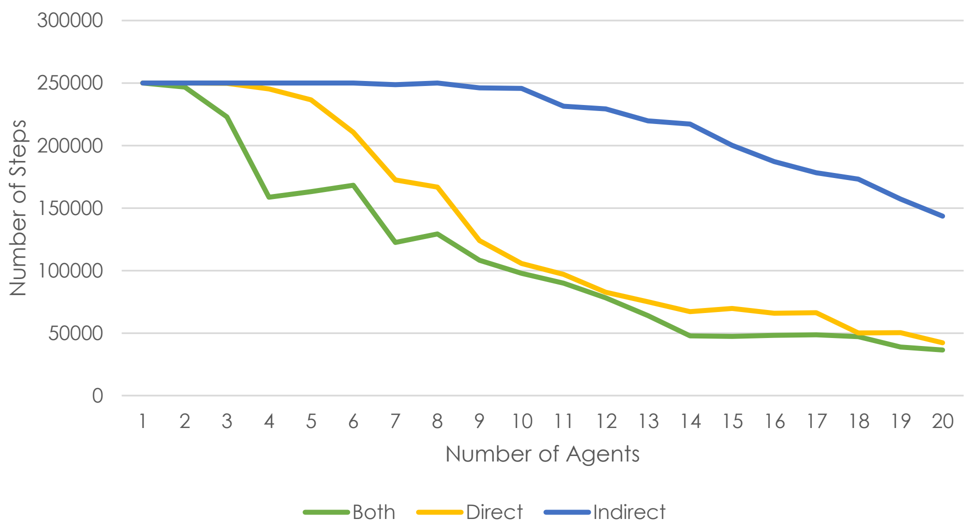

Figure 4.11.: Results of the MAF variants, Park, 500×500 cells

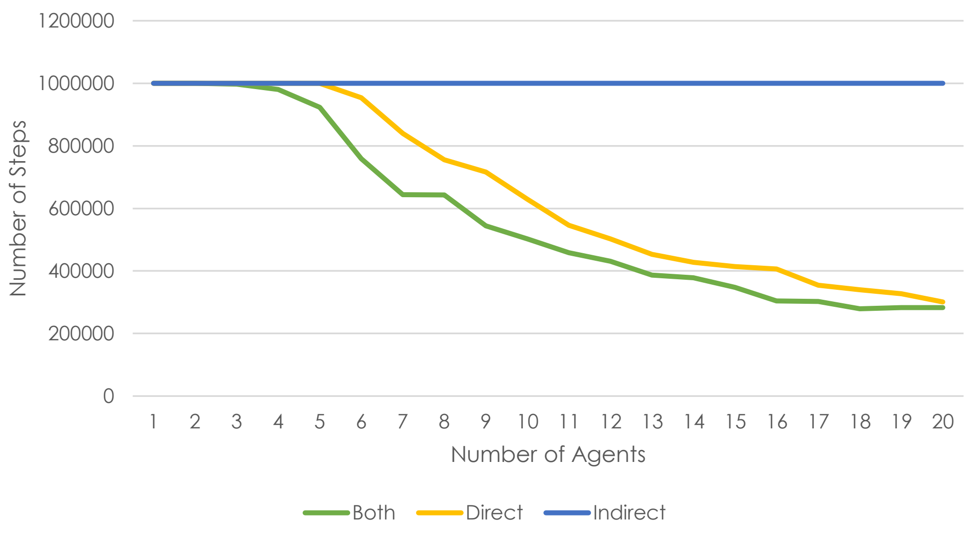

Figure 4.12.: Results of the MAF variants, Cubicles, 1000×1000 cells

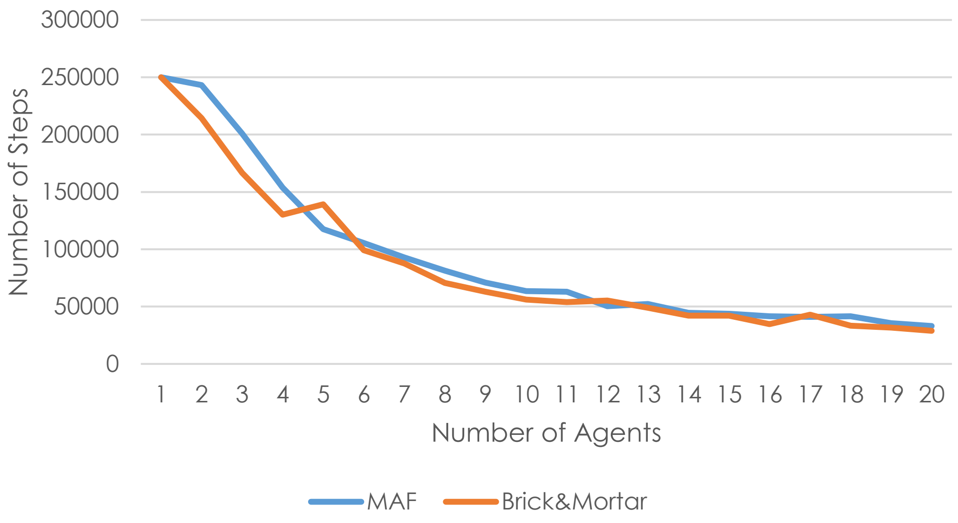

Figure 4.13.: Comparison between the combined MAF algorithm and Brick&Mortar, House, 500×500 cells

the allowed time, 50.000 steps, on the smaller maps and a fifth on the bigger map, or 200.000 steps. The number of explored cells is also slightly larger, compared to the indirect variant. Also it only needed at a minimum around four agents to finish the exploration in all the test cases.

Theoretically the new version of the algorithm, implementing the concept of layered communication should be at least as good as either the direct or the indirect variant, depending on which is faster. The results in all the four diagrams validate this assumption. Even if the average results of the direct algorithm in Figure 4.10 are in some cases a little faster. Another advantage of the combined algorithm is that it needs the minimal number of agents of all three variants to function, as in some cases two agents are enough to allow the algorithm to terminate in time. Additionally the combination also has the steepest progression of explored cells of all three algorithms, thus offering the fastest exploration in comparison with the other two variants.

The results of the simulation show that the concept of layered communication is a feasible optimization of the Multi-Agent Flood algorithm and enables the agents to achieve a valuable speed up in the search and exploration time. Especially as the agents will still return back to the starting point as soon as a point of interest is found. Figure 4.13 shows a diagram of the direct comparison between the new version of the MAF algorithm, blue line, and the Brick&Mortar algorithm, red line. The test case used for this comparison is the map *House*, depicted in Figure 4.5. The diagram shows that both algorithms run in more or less the same time, while the MAF algorithm offers additional advantages not available to the Brick&Mortar algorithm.

The simulations have shown that the combination of the two communication methods offers a robust framework for information exchange. The use of direct communication will speed the algorithm up, as long as the agents have access to this means of transmission, but they will also have the indirect communication available, should the terrain prohibit the use of radio signals for contact between two or more agents. Even by asserting very conservative assumptions for the use of direct communication, in this case only allowing communication as long as both agents have established a line of sight between each other and as long as they keep in a specific range, the speed up of this combined method is very visible.

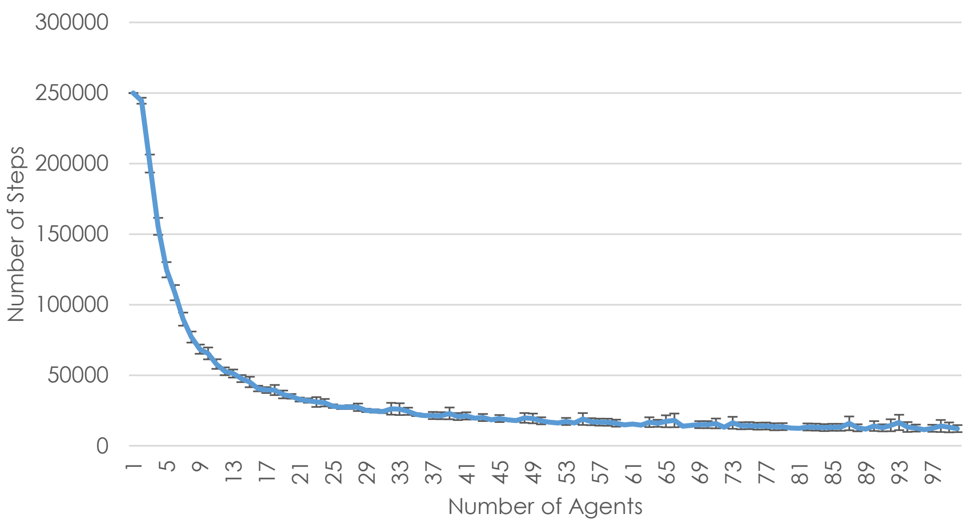

Figure 4.14.: Results of the MAF algorithm, House, 500×500 cells

To further demonstrate the viability of the algorithm additional simulations were run. Instead of repeating the simulation for each configuration 30 times, each configuration was repeated 250 times. Additionally the maximum number of agents was raised from 30 agents to 100 agents. The former simulations resulted in 600 simulated test runs for a single map and a single algorithm, thus the indirect version of the Multi-Agent flood algorithm was simulated 3000 times. The new combinations of the different configurations

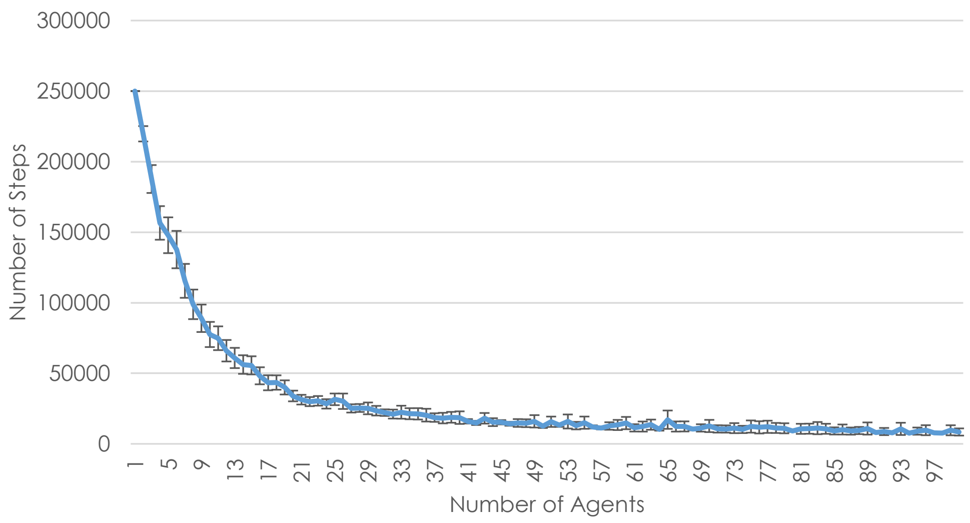

Figure 4.15.: Results of the MAF algorithm, Obstacles, 500×500 cells

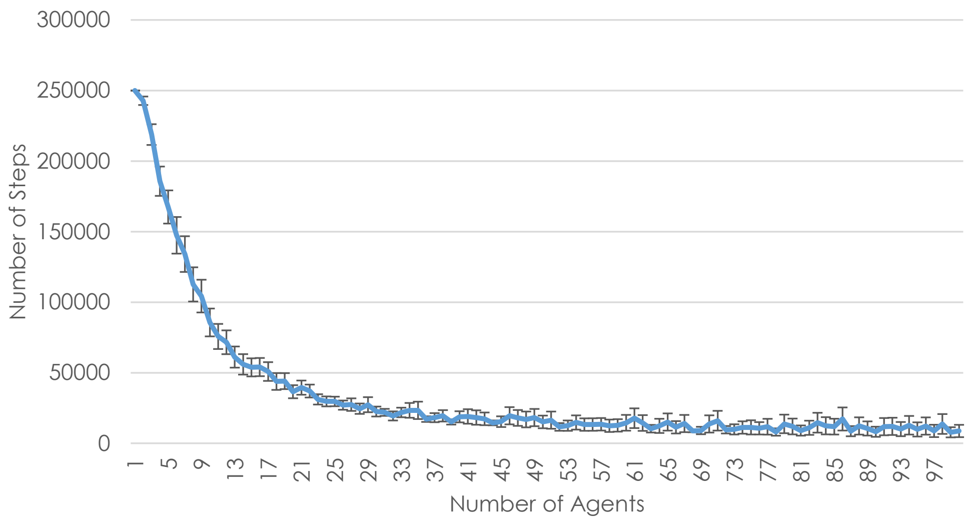

Figure 4.16.: Results of the MAF algorithm, Park, 500×500 cells

and repetitions culminated in 250.000 simulations for a single test case or 1.000.000 simulations for the four test cases combined.

The diagrams shown in Figures 4.14 to 4.17 show the average outcome of each of the runs. On each data point the 99% confidence interval is also given for all 250 simulations of this configuration, illustrated by the usual T-capped line.

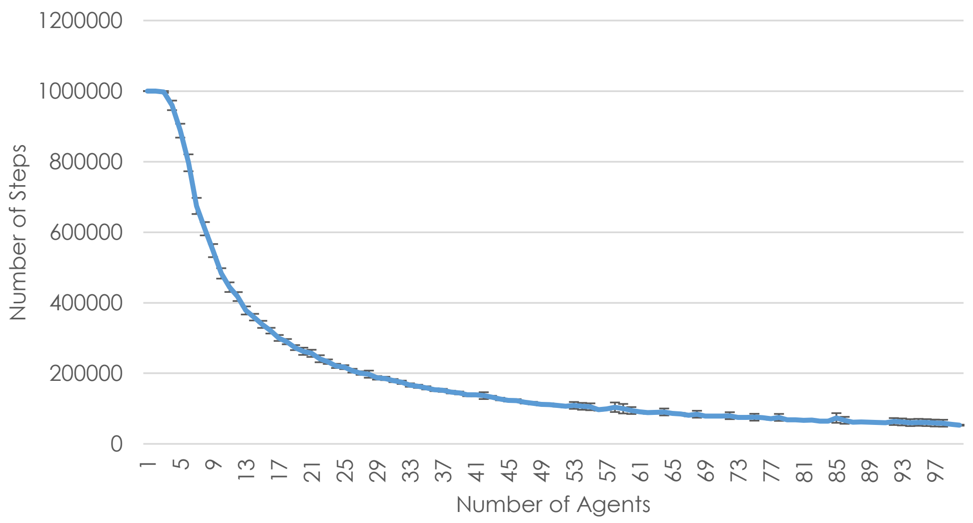

Figure 4.17.: Results of the MAF algorithm, Cubicles, 1000×1000 cells

In all four diagrams it can be easily seen, that the results of the simulations are rather stable, as the confidence intervals are relatively small. This means that the variance of the 250 simulations for each data point is low.

## 4.5. Chapter Summary

The introduction of a new model of communication allows for a robust framework for data transmission in scenarios, where a working WiFi connection can not be guaranteed. The possibility to degrade gracefully and fall back on the use of indirect communication allows agents to still function and to be able to finish the calculation of the search process, without being dependent on direct communication. This in turn is essential for a working search and rescue multi-agent system. Further it allows the agents to use the possible speed up offered by the use of direct data transfer to finish the search process much faster.

The table depicted in Table 4.1 presents the new comparison between the different algorithms. The change to this table is printed in bold and shows that the MAF algorithm is the only one that is able to use both communication models.

New simulations have also shown that the Multi-Agent Flood algorithm is rather stable, delivering on average rather constant results without any big

| Name | Goal | Communi-cation | Movement | Information back to HQ | Maintain-ability | Size |
|---|---|---|---|---|---|---|
| Ants | PoIs | Indirect | Grid-based | ✗ | ✗ | ✗ |
| Brick&Mortar | Explore | Indirect | Grid-based | ✗ | ✗ | ✗ |
| HybridExploration | Explore | Indirect | Grid-based | ✗ | ✗ | ✗ |
| Rendezvous | Explore | Direct | Grid-less | ✓ | ✗ | ✗ |
| MAF | Both | **Both** | Grid-based | ✓ | ✓ | ✗ |

Table 4.1.: Comparison of the different algorithms, MAF now uses both communication models

outliers, and thus confirming that this method is sufficiently robust.

But even the enhanced version of the MAF algorithm, implementing the layered communication concept, offers various possibilities of improvement. This version still relies on a grid to work. This abstraction was used to hide complications in the position measurement and to make the use of markings in the environment easier. The downside of this specific abstraction is, that it is not really applicable for the use in real robots. There are two possible ways on how to adapt the Multi-Agent Flood algorithm to leave this restriction behind: either simulate a virtual grid in the robots or replace the grid with another mechanism, one that is usable by robots without trying to press the "real world" in specific abstractions.

CHAPTER 5.

# LEAVING THE GRID

Usually in computer science problems get broken down into smaller problems that are either easier to handle or where some solutions already exists. This is called divide and conquer. An alternative to this method is to simplify things and to create abstractions, so that some points that do not seem very important for the general concept of the algorithm to work can be either ignored or replaced by other ideas.

The Multi-Agent Flood algorithm is just one example for latter of these two concepts. At the inception of the method the model used to represent the surrounding terrain was chosen to be a matrix $M^{m \times n}$, c.f. Becker et al. (2013). This sort of abstraction allowed for a simple movement of the agents, as they had just eight different directions in which to move. Additionally setting the speed was also very easy, just specify how many cells can be crossed in one step. Another advantage of this model was the way on how the markings of the agents can be stored. By using a grid, made out of single square cells, each marking could simply belong to a single cell. Thus the abstraction of the grid model served in three different ways. But in the end it is an abstraction and if the Multi-Agent Flood algorithm should be applicable in the "real world", controlling real robots this abstraction does not work anymore and will hinder further progress.

There are now two possible methods to handle this abstraction and to allow the adaption of the algorithm. The first possibility would be to program the robots to see the world as a grid. This would be very cumbersome as the sensors of the robot have to be programmed to deliver their data in such a way, that the robot would still move in a grid. But this is not why this solution becomes impracticable. By changing the robots to see the world in a grid and dividing the terrain into separate virtual cells the agents still would have to leave markings in the environment. In general this approach would be awkward and unwieldy.

The second way to solve this abstraction would be to adapt the algorithm itself to remove the grid metaphor. This way robot movement and positioning would not change all that much. Just use a gradient vector as movement direction and simply set a specific movement speed. This way the robot does not need to create a virtual grid to keep track of its own position.

Either way, both solutions need to mark the environment to allow the indirect communication part of the algorithm. How would these markings get stored? The introduction of the Multi-Agent Flood algorithm by Becker et al. (2013) already mentioned to disperse RFID chips in the environment to store these markings. The concept is not a new idea in general, see respectively [Sakakibara et al. (2007); Ziparo et al. (2007); Baglietto et al. (2009); Khaliq et al. (2014); Chinnaiah et al. (2016)], but the current implementations may not be really applicable for a search and rescue scenario, as they in turn fall back to the use of certain assumptions. For example Ali Abdul Khaliq proposes to use already existing RFID tags embedded in the floor for the robots, Khaliq et al. (2014). The same applies to the work of Chinnaiah and others introduced in Chinnaiah et al. (2016). Both of these proposals assume that some sort of RFID tags are already available in the terrain and the robots just have to find some sort of path between the different tags.

A solution for this idea would be to remove the grid abstraction from the Multi-Agent Flood algorithm and replace it with a loose net formed out of dispersed RFID tags. The agents will explore the unknown terrain and will drop the RFID chips at specific intervals. More and more tags will be scattered through the scenario resulting in a dense mesh, that will replace the grid and each chip will be able to act as a cell, storing data for the use of the different agents. This concept to use dynamically dispersed RFID tags in search and rescue scenarios was introduced by the author in Blatt and Szczerbicka (2016).

After the change to the communication channel in Chapter 4 the modifications proposed in this chapter are the next step in creating a realistic multi-agent system.

## 5.1. A mesh of RFID tags

To create this mesh consisting of the different tags the agents of the Multi-Agent Flood algorithm have to be adapted. Furthermore the kind of RFID chip which will be used has to be specified.

### 5.1.1. Different kinds of RFID chips

Chapter 4 provided an introduction into the different types of RFID tags, that are currently available. For this idea to work, the agents will need chips whose access range is greater than some centimeters. This means, that all the passive classes, 1 to 3, will not be usable in this approach. This leaves the semi-passive type 4 RFID tag and the active type 5 chip. The robots will have to carry a higher number of chips with them so that they can be distributed in the unknown terrain, as a consequence, these chips should be cheap and in the end more or less disposable. Thus the best type of chip to use would be the class 4 chip. It allows the robots to store data in it and it has a read and write range of about four meters.

### 5.1.2. Changes to the Multi-Agent Flood agent

As described above a grid is currently used to abstract the movement and the positioning of the agents in the algorithm. Each agent can move from cell to cell and has eight possible neighboring cells, the Moore neighborhood, to choose from. This also allows the agent to do any turn in the cell without needing additional time to turn around. By removing this abstraction, and in turn the grid, the algorithm has to be adapted to include a turning mechanism that takes the time to finish the turn into account. This also depends on the type of robot used in the scenario. The robots introduced in the second chapter were all tracked vehicles except one of the older team Hector variants. Tracked vehicles have the advantage that they can turn more or less on the spot, but they offer a slower movement speed. Wheeled ground vehicles can achieve faster speeds, but they can not really turn on the spot anymore, needing to switch back between moving forwards and backwards to turn slowly around.

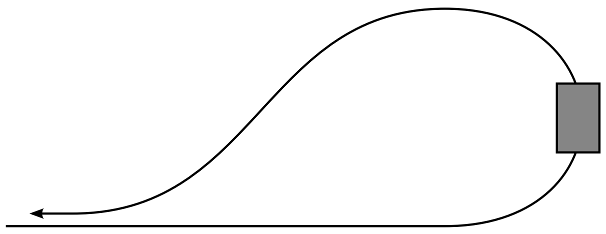

Figure 5.1.: Turning in form of a teardrop

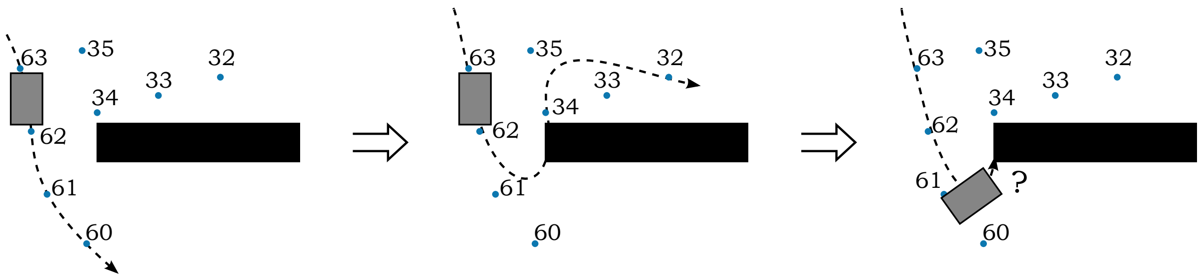

Figure 5.2.: Problem while turning with sensing RFID tags near obstacles

The change to the current Multi-Agent Flood agents was to include a movement direction in degree and a movement speed. Currently the movement speed is constant, as is the current turning angle to also allow for the simulation of wheeled robots. Simulating tracked robots with this setup is also possible, the turning angle has to be raised for this and the movement speed has to be reduced as long as the agent turns around.

This introduced some further changes in the algorithm, as the agents will now turn in some kind of tear drop shape, as soon as they want to reverse their direction, as shown as a gray rectangle in Figure 5.1. Additionally the agents will now encounter some movement problems as soon as they are trying to turn near walls and other obstacles. This is illustrated in Figure 5.2. The agent, again represented as a gray box, tries to change its path. The middle picture shows the new path and the right hand picture exhibits the problem: the agent tries to move towards the tag with the number 34. Due to the increased turning radius the agent cannot turn in place and encounters the wall (represented as the black rectangle). As it stands in front of the wall it will loose the range towards the tag with 34 stored in it.

Another new point that has to be addressed is the dispersal of the RFID tags in the environment. The current drop rate of the chips is static and will

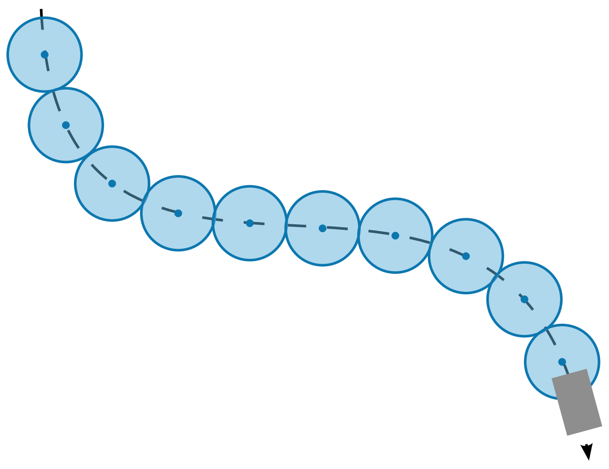

Figure 5.3.: Trail of RFID chips, dropped at a specific interval

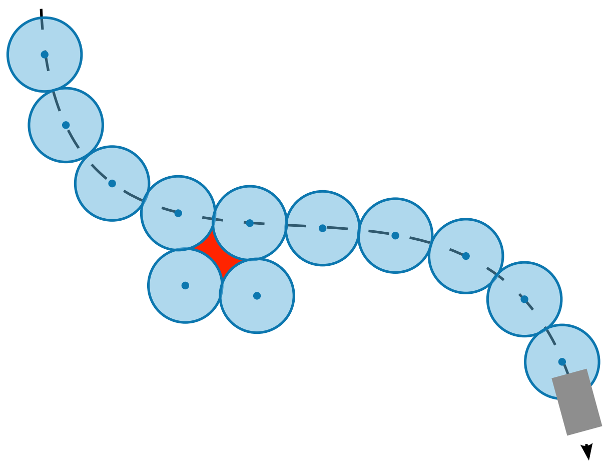

Figure 5.4.: Partial mesh of RFID chip trails, with gap

depend on the speed of the robots and the read and write range offered by the chips. Figure 5.3 shows an example of a trail created out of RFID tags, dropped at regular intervals by an agent. Each tag is represented by a smaller dark blue circle, surrounded by the communication range in a lighter blue circle. The agent is shown as a gray rectangle, moving along its path (the black dashed line).

The overlap between the communication range of the different chips should be as small as possible. This will in turn lead to the problem that there may be some gaps in the resulting mesh. Which is a natural result as long as a plane is covered in not overlapping circles, as depicted in Figure 5.4. Two

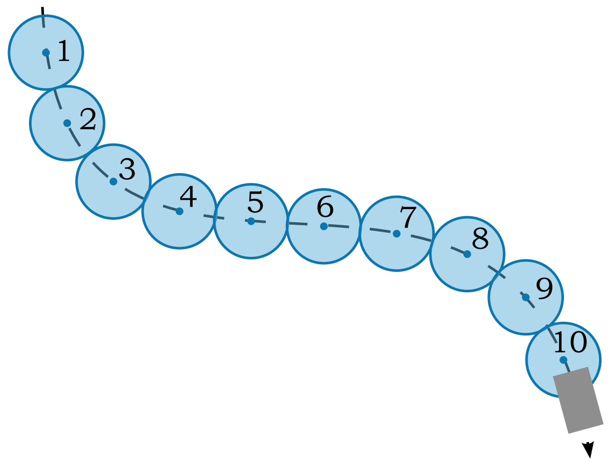

Figure 5.5.: Trail of RFID chips with corresponding stored data

additional RFID tags are shown near the trail, illustrating the resulting gap between the four chips in red.

Each cell in the former grid held the information about the currently known distance from said cell to the starting point. This is the same information that will be stored in the dropped RFID tags. The behavior of the agents does not change. An agent still reads the tag at the current location and compares the value it read with its own step counter. Should the current step counter of the agent be lower than the data stored in the tag, the robot will overwrite the value stored in the tag with its own step counter. An example of this is depicted in Figure 5.5.

Another problem that is introduced to the algorithm by using the mesh approach is that an agent may be in range of two different chips simultaneously. If this is the case, the agent may not be able to triangulate one or both of the chips correctly. Thus the agent has no way to pick the right direction in which to move depending on the data read from both chips.

As stated above the drop interval is static and does not change during the execution of the algorithm. It also depends on the movement speed of the agent and the communication range of the chips used. Therefore the calculation of the interval has to be done by hand, so that the overlap of the radii is kept as small as possible, while still offering the most coverage of the terrain without creating too many and/or to big gaps between the tags.

The code presented in Algorithm 6 describes how the changes to the *Search Mode* of the Multi-Agent Flood algorithm are realised. The agents checks for

**Algorithm 6** Adaption of the *Search Mode* of the MAF agents

```
 1: list tags = getTagsInRange();
 2: tag minimum = getNearestTag(tags);
 3: if currentDistance >= interval and not isInRange(minimum) then
 4:    drop next RFID tag t;
 5:    t.data = tagCounter;
 6:    currentDistance = 0;
 7: else if isInRange(minimum) then
 8:    if minimum.data > tagCounter then
 9:       minimum.data = tagCounter;
10:    end if
11:    currentDistance = 0;
12: else
13:    currentDistance++;
14: end if
```

all tags in its range and calculates the tag which is nearest to its own position. If the drop interval is reached (line 3) and there is no tag available in the minimum drop range of the agent, then a new RFID chip will be dropped, storing the current tag counter of the agent (lines 4-6). If there is a chip in range, then the data will be read and changed if the current counter is smaller than the stored counter (lines 8-11). Otherwise just increase the distance of the agent.

**Algorithm 7** Changes to the *Return Mode* of the MAF agents

```
 1: list tags = getTagsInRange();
 2: tag minimum = tags.firstElement();
 3: for all i ∈ tags do
 4:    if i.data < minimum.data then
 5:       minium = i
 6:    end if
 7: end for
 8: if mapUpdate?(tags) then
 9:    updateMap(tags)
10: end if
```

The changes to the *Return Mode* are shown in Algorithm 7. The algorithm checks for all surrounding tags and searches through that list to get the tag with the lowest counter (lines 3-7). After it has found this tag, it checks if a map update based on the information in the surrounding tags is required. If there is new information available the map will be updated (lines 8-10).

### 5.1.3. Changes to the simulation setup

There also have to be made some changes to the simulation setup of the Multi-Agent Flood algorithm. To get a close enough comparison to the grid based variant of the MAF algorithm most of the settings were kept the same, although some things had to be changed, so that the simulation of the grid-less algorithm was possible.

The test cases are the same as proposed in the previous chapter and can be seen in Figures 4.5 to 4.8. The first difference between the grid-based and the grid-less algorithm is the way on how the cells are perceived. In the former variant each cell on the map was seen as one cell in the grid. The grid-less algorithm will still be able to drop a RFID tag into one cell on the map, but the range of the chip will cover more cells in a circular radius, where the range depends on the setting.

The different states of the cells also remain the same: wall or open space. Whereas the open space can either contain an agent or a point of interest. Additionally each cell can now also have a RFID tag, which is also not blocking the cell, thus agents can simply occupy this location.

As before three different versions of the algorithm were modified to remove the grid abstraction. These three variants are the indirect communication algorithm, the direct communication algorithm, and the combined version. The first two variants were only included as a benchmark for the combined, or layered communication, version.

Based on the previous results the number of agents was also changed for this simulation. The configurations started with one agent and added an additional agent for each new configuration until the last configuration was run with 30 agents. Each configuration was repeated 30 times, thus 10.800 simulations overall were run.

Comparisons with other algorithms are not very easy to do or rather not that practicable. For example the Brick&Mortar algorithm or the HybridExploration can not be compared to this version of the MAF algorithm, as those agents still act on a grid. Thus a direct comparison is not possible between the different algorithms and no other algorithms are shown in the result diagrams.

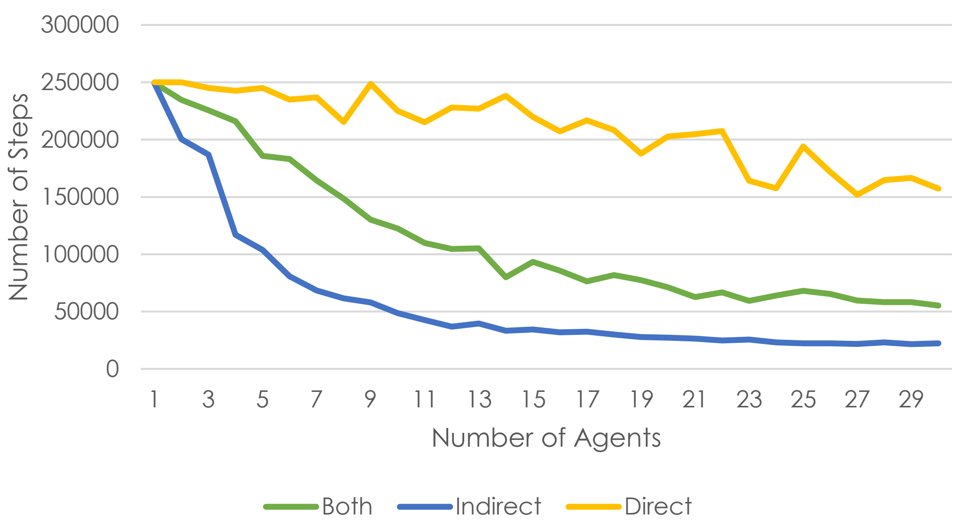

Figure 5.6.: Results of the grid-less MAF algorithms, House, 500×500 cells

## 5.1.4. Results of the new simulation

The four diagrams depicted in Figures 5.6 to 5.9 show the results of the simulation using the adapted Multi-Agent Flood algorithm.

The layout of the diagrams is the same as before, the X-axis shows the number of agents used and the Y-axis the amount of steps needed for the algorithm to finish the search process. Three different versions of the algorithm are represented in each diagram: a yellow line for the direct variant, a blue line for the indirect version, and a green line for the combined algorithm.

The results presented in the four diagrams show that the adaption into a grid-less variant of the algorithm works, although these versions are not as fast as the older versions working on a grid. This is a consequence caused by the changed movement algorithm, as agents now will need longer to turn in the desired directions. The time it now takes to turn the agent around was completely ignored in the gris-based iterations of the Multi-Agent Flood algorithm.

The results of the direct variant of the algorithm show, that using only this way to exchange data between the robots, especially under the assumed constraints (only while having a line of sight connection and a small maximum range), is not really feasible. Additionally this is also expanded by the newly introduced movement problem already described above. This problem occurs

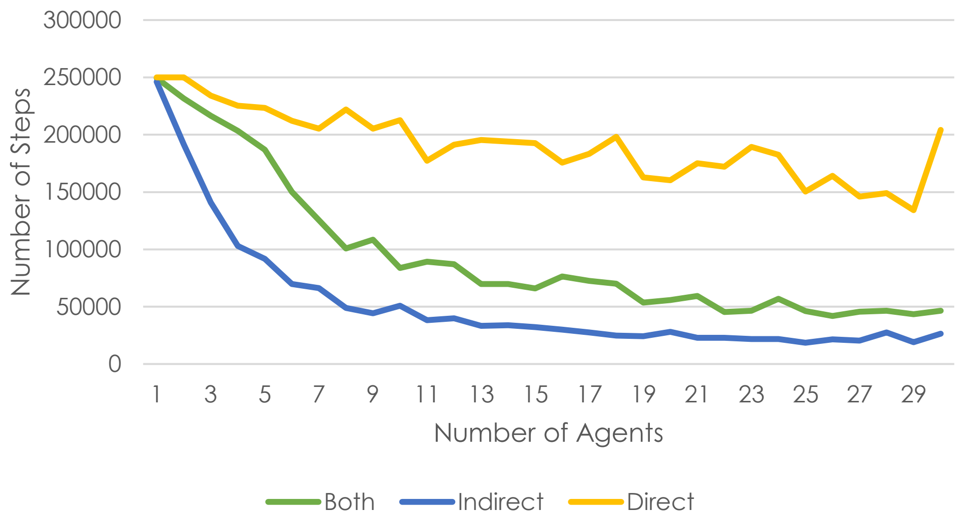

Figure 5.7.: Results of the grid-less MAF algorithms, Obstacles, 500×500 cells

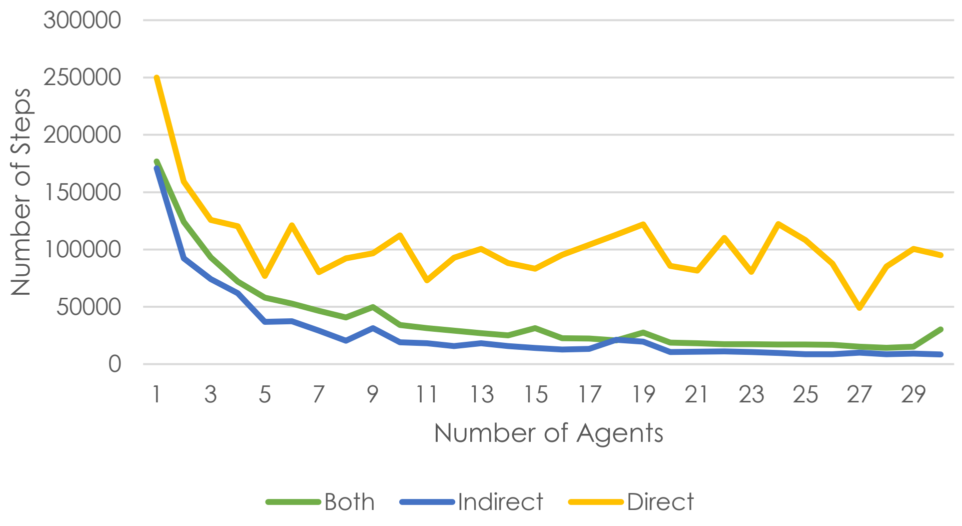

Figure 5.8.: Results of the grid-less MAF algorithms, Park, 500×500 cells

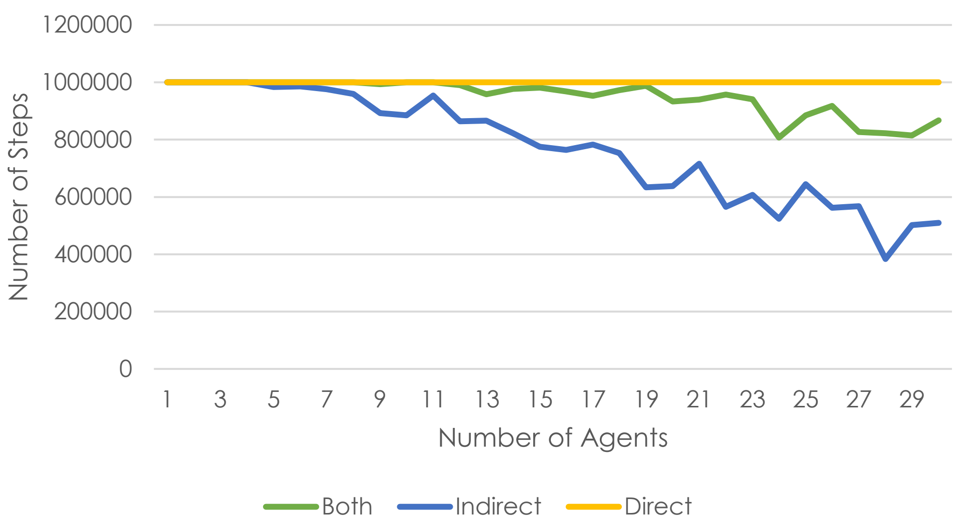

Figure 5.9.: Results of the grid-less MAF algorithms, Cubicles, 1000×1000 cells

as soon as an agent is near an obstacle and tries to move towards a specific RFID tag. If the agent tries to turn around and thus the chip gets covered by a wall, the agent will loose the signal from the tag and in turn it is not able to find the tag anymore. Currently the agent will try to get back to the old position to find the chip again and in the worst case, this will start a cycle, where the agents tries to move in the direction of the tag, gets blocked by a wall and tries to revert the movement.

The diagrams depict only the average of thirty simulation runs per configuration. As mentioned previously they also include the runs that did not terminate successfully. The grid-less algorithm was only able to finish half of the simulations illustrated in Figure 5.9 in time.

Nevertheless the results of the different simulations show, that this concept and its adaption to the Multi-Agent Flood algorithm is working. This allows for a removal of the grid abstraction from the algorithm and enables the agents to disperse RFID tags through the terrain to use as a mesh to store the data needed for the indirect communication. Although this is a workable solution and enables the use of the MAF algorithm in “real world” problems, it is still far from optimized. Especially as the map data, that the agents are currently collecting and sharing, is not really used. The agents are unable to plan strategies ahead based on the collected and exchanged data.

## 5.2. Reliability of the RFID tags

The changes towards the communication model used by the Multi-Agent Flood algorithm presented in Chapter 4 assume very conservative values for the range and the accessibility of the direct communication to provide an error free transmission channel. The grid-less version of the MAF algorithm introduced a more applicable model of the indirect communication, relying solely on the distribution of RFID chips in the environment to store information used by the agents to find their way back to the home base. These changes now raise the question on how reliable the dispersed RFID tags are.

Four general sources of errors in RFID tags are the following, taken from VDI4472 Blatt 10 (2008):

- Surfaces, including reflective, absorbing, or conductive surfaces
- Radio frequencies sources
- Surrounding environment, for example rain or temperature
- Process conditions, for example the amount of data or bulk reading

Each of these sources can be encountered in a disaster area, for example a rubble strewn part of a destructed building or a street. This means that each RFID tag that a robot will drop has a chance of either being unable to respond to requests, and counting as lost, or having a severely limited range. Thus restricting the use each agent can get out of a chip. An additional source of errors concerning the tags is the movement of the robots itself. As a robot traverses the already explored terrain, it will encounter already dropped RFID chips and it will move right over them. This can either kick the specific chip away from its former position, disrupting the laid path, or simply destroying it.

The fourth source of errors, generally defined as process conditions, usually stems from the fact that multiple RFID tags may be in the same range and as soon as the agent tries to read a single chip it will access all available chips in range. This is a known problem called bulk reading. The agent is not able to differentiate the tags without help. Various approaches already exists to solve this, c.f. [Cha and Kim (2005); Cheng and Jin (2007); Klair et al. (2010)].

### 5.2.1. Modelling the different errors

The first three error sources can be modelled by simply giving each dropped RFID tag a chance to malfunction. A malfunctioned chip will not work anymore and thus will leave a hole in the path. Additionally as soon as an agent moves over an already dispersed chip there exists a chance that the agent will flip the tag away from its original position. The distance of this flip is randomly determined in a range from zero to ten units, which in the case of the MASON simulation means from zero to ten cells.

The bulk reading problem can be ignored as enough algorithms exist, that an agent is able to read multiple chips at the same time [Cha and Kim (2005); Cheng and Jin (2007); Klair et al. (2010)].

In the end the model of the emanating RFID tag errors can either result in a missing chip, or in a chip that was moved from its original position. This either leaves a hole in an existing path, as described above, or in overlapping chips, in which one of those has a "wrong" step counter stored in it.

### 5.2.2. How to handle these malfunctions

The agents of the Multi-Agent Flood algorithm have two work with these two cases of malfunctions. Missing chips can simply be replaced in the mesh of RFID tags as soon as another robot will encounter the free space. This newly dispersed chip will contain the current step counter of the robot. Thus closing the gap in the path. The same applies to breaks in the path caused by chips that were kicked away.

Chips that were flipped away will be simply handled like other chips. If an agent is in range it will be read, or updated, if the conditions are met.

### 5.2.3. Evaluating the reliability of RFID tags

For the evaluation of the possible RFID errors a new variable was introduced into the simulator. This variable specified the chance for an error to occur in the RFID chip, either removing the chip completely from the simulation, or moving the chip away from its original position.

The tests were done on three different maps, depicted in Figure 5.10 with three different sizes: 300×300 cells, 500×500 cells, and 1000×1000 cells.

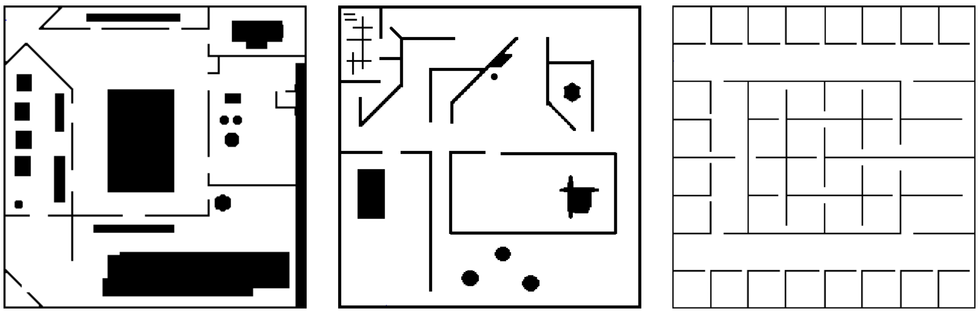

Figure 5.10.: Test cases for the reliability evaluation; Factory, 300×300 cells on the left, Office, 500×500 cells middle, and Cubicles, 1000×1000 cells on the right

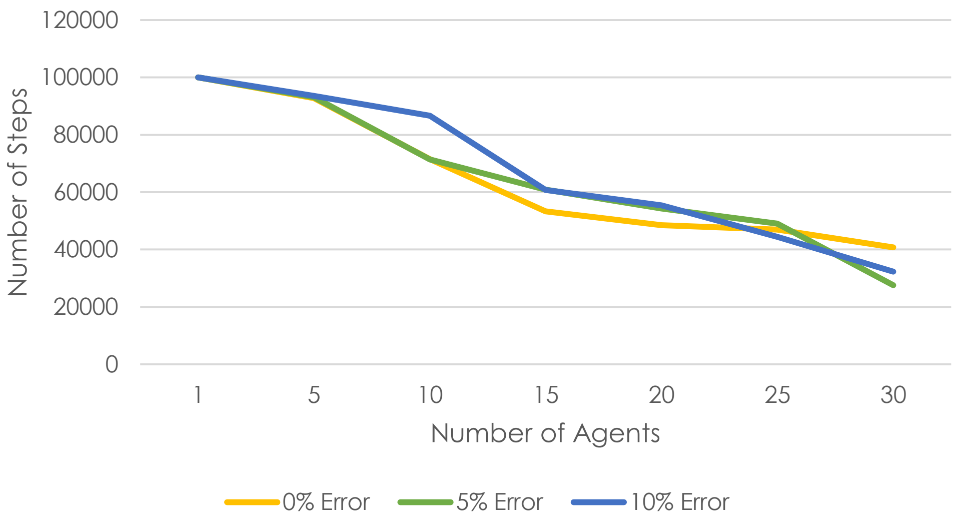

Figure 5.11.: Results of the RFID reliability evaluation, Factory, 300×300 cells

Each map was tested with 1, 5, 10, 15, 20, 25, and 30 agents, while using an error rate of five percent and respectively ten percent. Each configuration was repeated 30 times. Otherwise the same setup as in the previous experiments was applied. The algorithm will stop as soon as 95% of the terrain was explored and 95% of all points of interest were found.

The following diagrams in Figure 5.11 to 5.13 depict the results of these simulations. Each diagram contains three lines, the green one shows the algorithm with an applied error rate of five percent, the blue line illustrates an error rate of 10%, and the yellow line represents the originial algorithm with an error rate of 0.

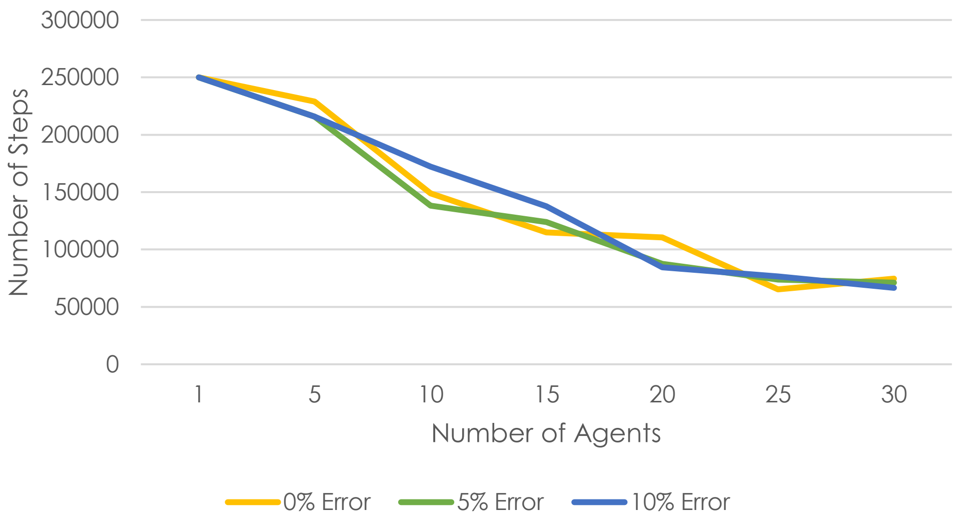

Figure 5.12.: Results of the RFID reliability evaluation, Office, 500×500 cells

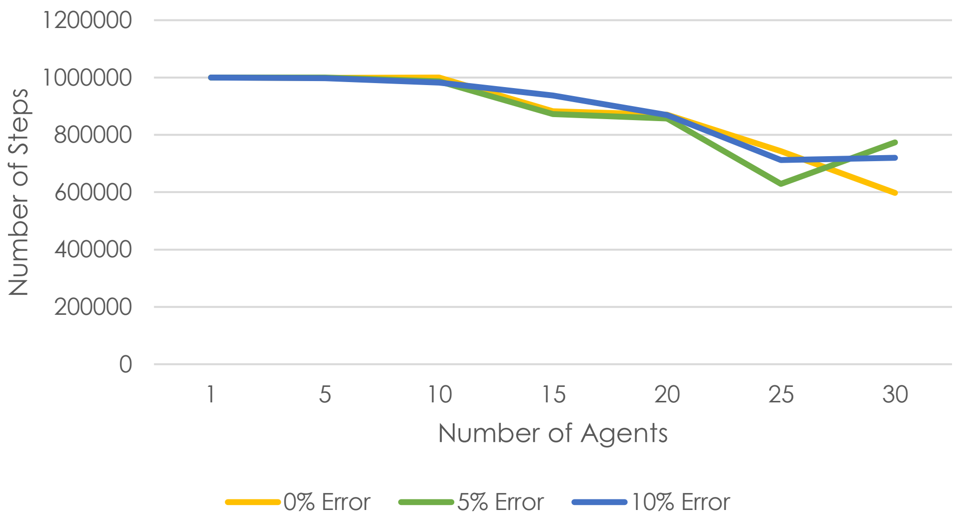

Figure 5.13.: Results of the RFID reliability evaluation, Cubicles, 1000×1000 cells

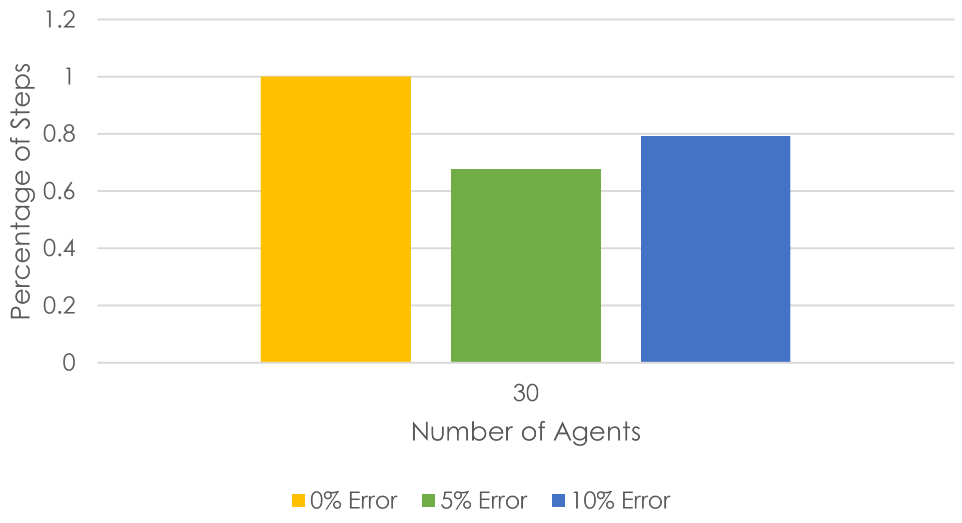

Figure 5.14.: Comparison of 30 agents between the different error rates, Factory, 300×300 cells

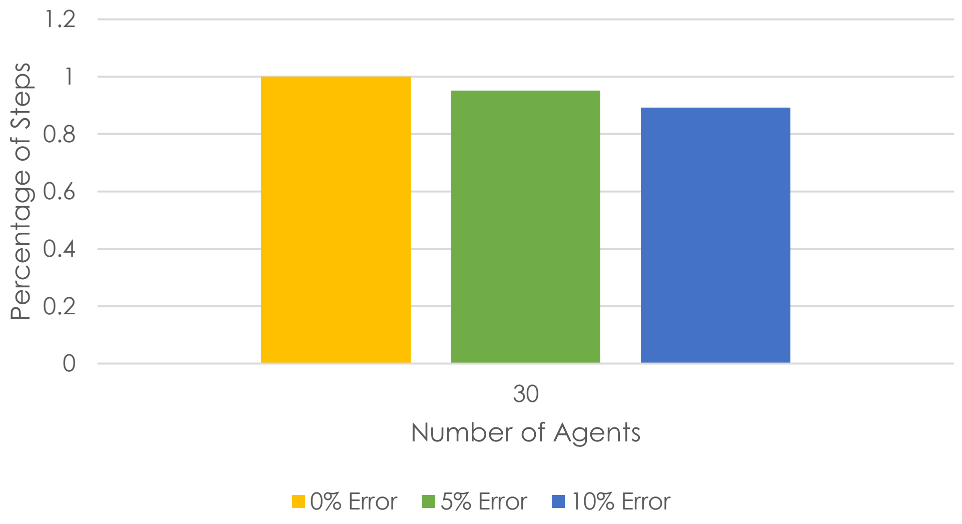

Figure 5.15.: Comparison of 30 agents between the different error rates, Office, 500×500 cells

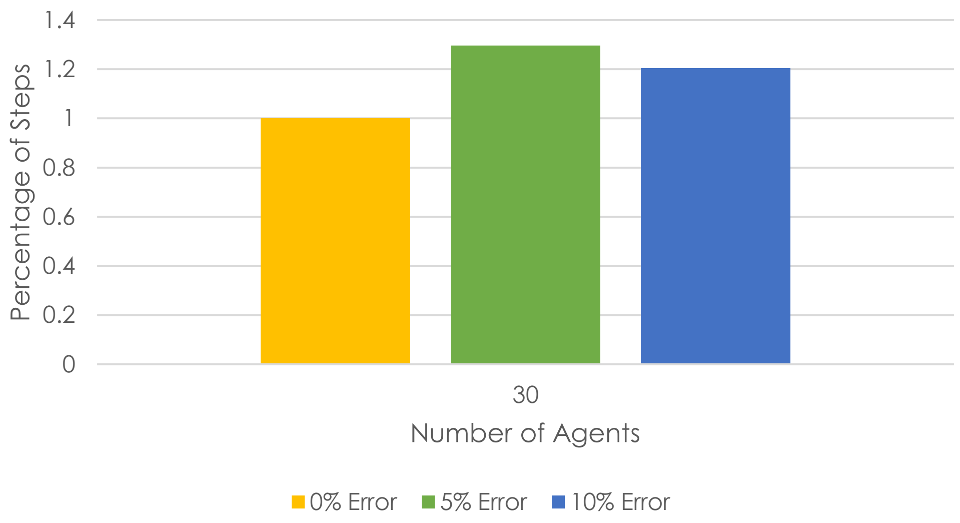

Figure 5.16.: Comparison of 30 agents between the different error rates, Cubicles, 1000×1000 cells

The outcome of this simulation shows that even a ten percent error rate, which is quite big for any equipment used in these kind of scenarios (compare the failure rate which lead to aborted procedures of the Da Vinci medicinal robot, Andonian et al. (2008), which was 0.05%), does not influence the algorithm all that much. Only in the larger scenario, as the results in Figure 5.13 show, is the discrepancy really visible. On the other hand in the smaller maps the algorithm with the applied error rate was even faster than the algorithm without any malfunctioning RFID tags. This can be seen in the diagrams in Figure 5.14-5.16. The Y-axis of these diagrams show the percentage of the steps it took the algorithm to finish the calculation without any errors as a baseline corresponding to 100%. Diagrams in Figures 5.14 and 5.15 clearly present that the algorithm using malfunctioning RFID tags is even faster than the original algorithm, which was not expected. The simulation using the larger test case, shown in Figure 5.16, conforms more to the expected values.

Nevertheless the outcome of the simulated Multi-Agent Flood algorithm with malfunctioning RFID tags highlights the robustness of this approach of using RFID chips for the indirect communication. Combined with the very conservative assumptions for the direct communication proposed in Chapter 4 this offers a robust framework for a multi-agent system for the use in search and rescue scenarios.

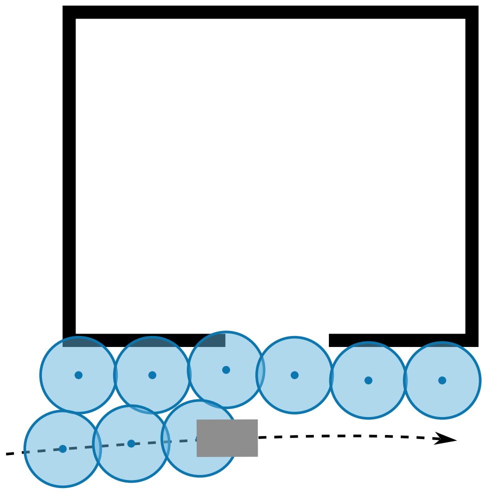

Figure 5.17.: Markings cutting off unexplored terrain

## 5.3. Optimizing the search time of the agents

The next step to improve the performance of the Multi-Agent Flood algorithm would be to enable additional capabilities in the agents, as introduced by the author in Blatt and Szczerbicka (2017). The current agents are kept intentionally very simple and the behavior of them has not changed all that much from the first variant of the algorithm presented in Chapter 3. Chapter 4 introduced a new communication model, thus giving the agents the potential to store information about the sensed environment and saving this information in an internal map. Due to the same changes the agents are also capable to exchange this stored map information between themselves, although the data was transmitted between two or more different agents, it was not really used to improve the performance of the algorithm.

The current implementation of the algorithm offers a working solution, but there is still room for improvement. For example the agents will try to ignore already visited terrain which is already covered in markings, these markings can prohibit them from scouting unexplored terrain which may be on the other side of the markings.

The drawing in Figure 5.17 depicts the problem of already dropped RFID chips cutting off yet unexplored terrain. This can lead to spaces that will be ignored for a while, thus decreasing the efficiency of the algorithm. The agents will prefer to move to unexplored terrain and will only move to space already covered with tags if they do not have any other choice.

## 5.3.1. Frontier-based exploration

The map data stored and exchanged by the agents should help the robots to decide which part of the terrain is still unexplored and would yield the most "new information" if an agent would visit it. This divide between explored and unexplored terrain is called a frontier and the agents select a point on this frontier to move towards to. By repeating this behavior, choosing a frontier point, moving towards this position, and choosing a new point based on the gathered data at the actual position the agent is able to explore the yet unexplored terrain. This way of exploring the area is called frontier-based exploration, c.f. [Yamauchi (1997, 1998)].

**Algorithm 8** Frontier Detection

```
1: Frontier s ← $\emptyset$
2: for all $v \in$ grid cells do
3:    if $v$ is frontier edge cell and not processed yet then
4:       FrontierRegion = getFrontierNeighbors(v);
5:       Frontiers = Frontiers $\bigcup$ FrontierRegion;
6:    end if
7: end for
8: return Frontiers
```

The map stored by the agents is saved as a two dimensional grid and the state of each cell is either explored (or more specifically visited terrain or an obstacle) or unexplored. Listing 8 shows how the frontiers are selected based on the map data. The algorithm searches for unexplored cells which have already explored cells as neighbors. These cells are called frontier edge cells. If a frontier edge cell is detected all neighboring edge cells will be merged to frontier region. These cells will be marked as processed, as the next iteration will ignore already processed cells.

After creating a list of available frontiers an agent has to select the "right" frontier as its next target destination. To achieve this the agent simply calculates the distance from its actual position towards the different frontier points. As shown in Listing 9.

Is the agent not able to reach its designated point in a specific time frame then it will add this point to a list of unreachable destinations and compute a

**Algorithm 9** Frontier Assignment

```
1: targetPoint = null;
2: for all FrontierRegion ∈ Frontiers do
3:    for all $v$ ∈ FrontierRegion do
4:       if targetPoint == null or getDistance(v) < getDistance(targetPoint) then
5:          targetPoint = v
6:       end if
7:    end for
8: end for
9: return targetPoint
```

new frontier point from the actual position.

This approach is also applicable to a multi-agent system without any problems. Each agent will use its own data to generate the list of frontier points based on its own knowledge. As the agents share the map data between themselves, new frontier points will be available and former unexplored terrain may suddenly be already explored. One of the main drawbacks of this method is the amount of calculation that is needed to recalculate the frontier points as soon as new information is available, as the algorithm will always iterate over all cells stored in the map. This will lead to a higher computation runtime if the maps are bigger.

### Wavefront Frontier Detection

Frontiers exist only between known and unknown terrain. This enables an optimization of the basic algorithm proposed by Yamauchi (1997). Instead of iterating over the whole map the optimized version of the algorithm only iterates over already explored terrain, reducing the number of cells which have to be processed. This variant, proposed by Keidar and Kaminka (2012), was named the Wavefront Frontier Detection algorithm (WFD). The newer algorithm will find the frontier points in less time than the basic frontier-based exploration algorithm, especially if the map is largely unexplored.

The algorithm will return again a list of all available frontier points. The MAF agent will pick the point depending on two criteria: the euclidean distance from the point to the robot and the amount of unexplored terrain surrounding the frontier point.

Figure 5.18 depicts a list of different points to choose from and the point

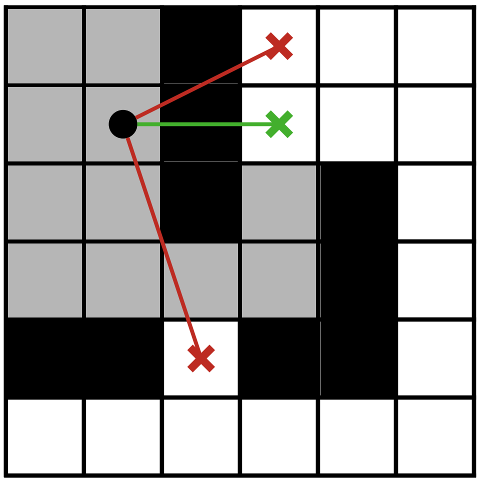

Figure 5.18.: List of available frontier points, with the chosen point in green and discarded points in red

fulfilling the criteria in green. Although the green point matches the rules of the algorithm it may result in a longer way than other points, in this example the red point at the bottom of the picture. To counter this a time limit was introduced, [Yamauchi (1998); Baranzadeh and Savkin (2016)]. If the time limit is overstepped the agent will chose a new frontier point. This also helps with navigational errors, which could trap the agent. The point that could not be reached in time will be marked as unreachable and the WFD algorithm will compute the list of points again, ignoring any point marked as unreachable.

The frontier-based algorithm helps the Multi-Agent Flood algorithm to improve the exploration of the unknown terrain. Another point which can be upgraded is the navigation strategy of the agents. Currently there is no targeted movement by a single agent and the multiple agents do not really cooperate with each other to really use the existing team to speed up the search process. Each agent searches alone and the current cooperation exists solely through the use of the markings on the floor. With the introduction of the frontier points the agents are now able to move towards these points in a targeted manner, instead of moving more or less randomly through the terrain. The next point of improvement would be the navigation algorithm used by the agents to travel to the selected frontier points.

### 5.3.2. Bug algorithms

Finding a way between two points is a known problem and many different algorithms which solve this problem exist already. As long as there are no dynamical changes in the environment the well-known A* algorithm may be used to calculate a way from the agent towards the frontier point, as described in Hart et al. (1968). The problem with this algorithm is that the agent may have to recalculate the way multiple times. This will increase the computation time as no information from a recently calculated way can simply be reused. Another drawback is, as mentioned above, that the algorithm will only work in a static environment. A change in the environment will force a recalculation of the path. An improved version of the A* algorithm, the Jump Point Search algorithm, Harabor et al. (2011), needs less computation time, but on the other hand will work only on a grid.

Another way finding algorithm, that does not need as much computations, was proposed in Lumelsky and Stepanov (1986). The authors proposed two variants of the same algorithm, called *Bug1* and *Bug2*. These algorithms are continually planning instead of computing the way one time, thus allowing for a dynamic environment.

The algorithm devised by Lumelsky et al. is rather simple. An agent starting from one point trying to get to another point will move in a straight line until the agent encounters an obstacle, which it will try to circle around and continue on its way to the target destination.

#### Bug1

The behavior of the *Bug1* is shown in Figure 5.19, c.f. (Rao et al., 1993, p. 12). Starting from point $S$ the agent will encounter the first obstacle at $H_1$. It will circle around the object clockwise and as soon as it reaches point $H_1$ again it will turn back and circle the obstacle counter clockwise until it reaches a point which is closest to the destination $T$. This is the leave point $L_1$. The agent will resume its movement in direction to $T$ until it will arrive at the second obstacle. The same steps will be applied again until the agent will also leave this object at the leave point $L_2$.

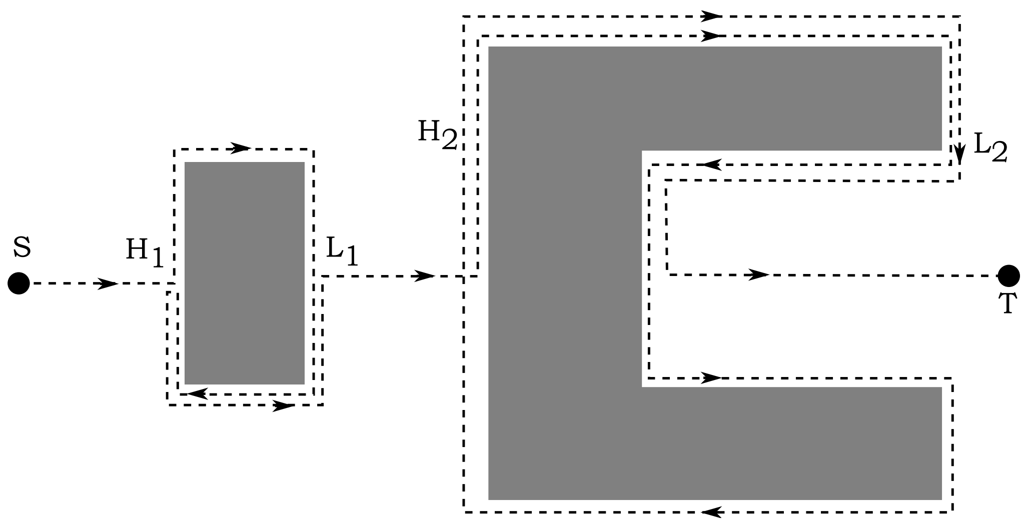

Figure 5.19.: Result of the *Bug1* algorithm

## Bug2

The second variant of the algorithm is an optimized version of the *Bug1* algorithm called *Bug2*. In this algorithm the agent will only move on certain segments of the path instead of circling the whole obstacle, resulting in less time being used to get to the destination point.

This is shown in Figure 5.20, see also (Rao et al., 1993, p. 13). Two additional distances will be calculated $d(H_i)$, the distance from the entry point of obstacle $i$, and $d(Q)$. Where $Q$ is a possible leave point $L_i$. If $d(Q) < d(H_i)$ and the line $QT$ does not cross obstacle $i$ at the point $Q$, then $L_i = Q$. The resulting path that the agent will take is shorter than the path calculated by *Bug1*.

As mentioned previously the *Bug2* algorithm allows an agent to act rather dynamically on its environment and does not really need a recalculation of the whole path. This method, combined with the Wavefront Frontier Detection algorithm, enables the use of the collected and shared map data. Agents now identify frontier points, these being points that promise the most “new information” while still being near enough for the agent to reach, and they are now capable of reaching a specific point in a timely manner with the help of the second variant of Lumelsky’s algorithm.

The last point that is still open for optimization is the cooperation between the agents. As each agent will compute the list of available frontier points

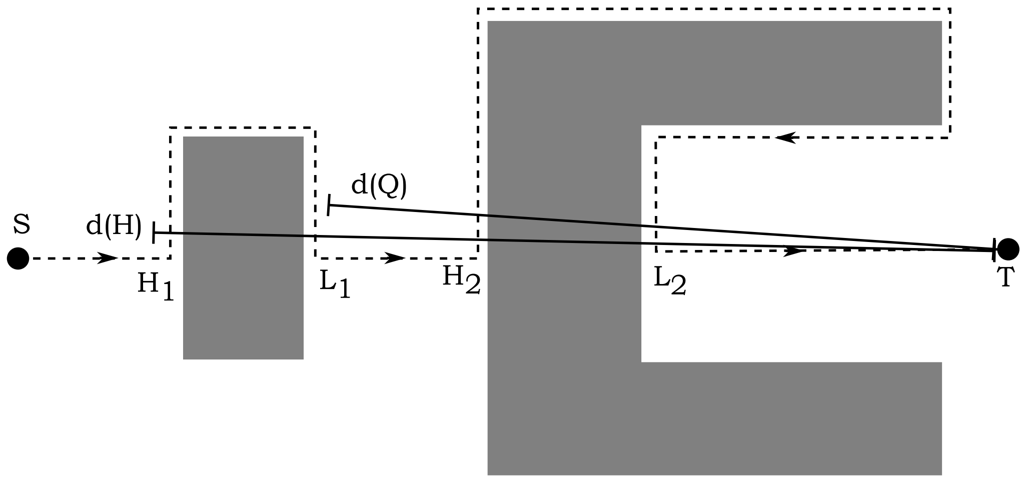

Figure 5.20.: Result of the *Bug2* algorithm

by itself it may happen that two or more agents will travel to the same spot. This would decrease the coverage of the terrain, as only one agent is needed to explore this position. At the same time the other agents could explore other still unexplored points.

## 5.3.3. Improving the cooperation

By introducing the concept of frontier-based exploration and subsequently the way finding through the use of the *Bug2* algorithm, the map stored in each agent is now being incorporated into the search and the return process of each agent. The agents now only have to share their intentions or in this case their next frontier point that they want to explore with other agents to deter multiple agents to converge on the same spot.

This mechanism is added after the map transmission between two agents. As explained in the previous chapter, as soon as two agents meet and certain conditions are met a data exchange will be initiated. Additionally to the data transmission the agents now also exchange their current targeted frontier point. The agent checks if the currently targeted spot was already explored by the other agent. If this is the case said agent initiates a recalculation of the frontier points and chooses a new point. Afterwards the agents swap the newly chosen points again. These points will get compared while keeping in mind a radius $r$, which is based on the range of the sensors. If both agents chose different points, then nothing happens. In the case that both chose the same spot to investigate, the agent with the lower range to this spot keeps this destination and the other agent starts to select a new point. The listing in Algorithm 10 describes how this mechanism works.

**Algorithm 10** Coordination of the frontier points between the agents

```
 1: if otherAgent.isInRange() then
 2:    exchangeMap(otherAgent);
 3:    if goal is explored then
 4:       frontierpoints = WFD();
 5:       goal = extractTargetPoint(frontierpoints);
 6:    end if
 7:    while goal is in range of otherAgent.getGoal() and otherAgent.getDistance() < this.getDistance() do
 8:       ignoreList.add(goal);
 9:       goal = extractTargetPoint(frontierpoints);
10:    end while
11: end if
```

The agent that does not change its targeted point has chosen a spot that is still unexplored, at least by this pair of robots, and the other agent will chose another spot, thus increasing the diversity of the chosen targeted positions. This approach does not rule out completely that two different agents target the same spot, but it decreases the amount of agents converging on the same position. This in turn increases the speed of the coverage of the terrain, as it forces the agents to explore different parts of the map.

A slight adjustment that was also introduced, to speed up the initial coverage of the agents, was a pre-selection of the first target points for the agents. This facilitates a fast spread of the agents at the start of the search process, as each agent is forced to explore a different part of the map right from the start. Should one of these points not be reachable or located within an obstacle the specific agent will switch the targeted position after a certain time frame has passed.

### 5.3.4. Results of the modified algorithm

The simulation setup used for the computation of these results is the same as before. The main difference this time was, that instead of simulating the behavior of the changed agents with only one agent and adding an additional

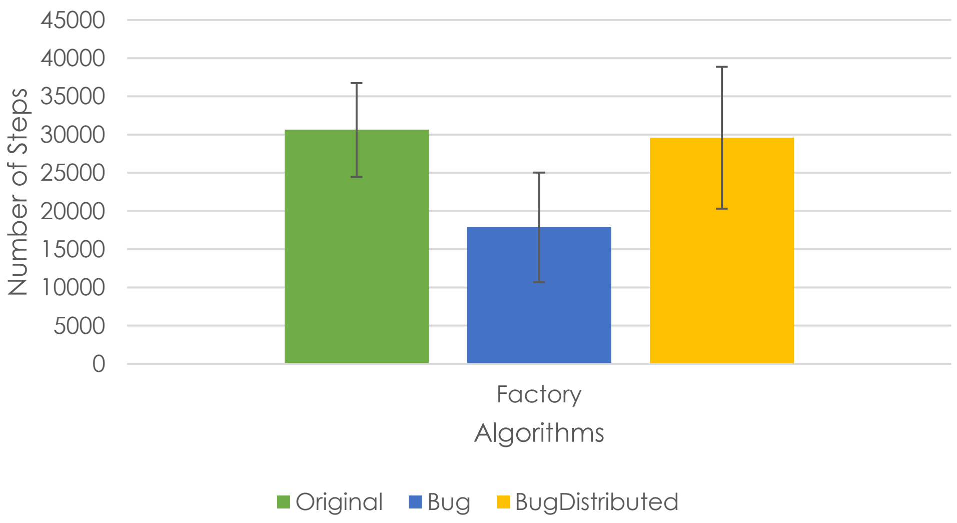

Figure 5.21.: Results of Frontier Detection adaption, Factory, 300×300 cells

agent each time until the number of thirty agents was reached, the simulation was done for only thirty agents. The maps used in this simulation are shown in Figures 4.5, 4.6, 4.7, and 4.8. These are respectively the House map, the Office scenario, the Park environment, and the bigger Cubicles outline. Additionally the agents were also tested on the smaller Factory map, shown on the left of Figure 5.10.

The following charts shown in Figures 5.21 to 5.25 display the results of these computations. Each diagram depicts the results of three different algorithms. The grid-less algorithm without any adaptions (*Original*), the modified algorithm as described above (*Bug*), and the adapted algorithm using predefined starting points for a better initial distribution of the agents (*BugDistributed*). The green bar shows the results of the original algorithm, the blue bar illustrates the *Bug* algorithm, and the yellow bar delineates the *BugDistributed* version. The Y-axis represents the number of steps each algorithm needed until it terminated successfully. These step numbers are the average of thirty iterations. Additionally Figures 5.21-5.25 include a T-capped line which depicts the 99% confidence interval for the simulations.

As expected with the use of the Frontier Detection algorithm the adapted versions of the Multi-Agent Flood algorithm could achieve a runtime speed up in the Factory map, which can be seen in Figure 5.21. The *Bug* algorithm needs only about the half of the runtime of the original algorithm. Furthermore

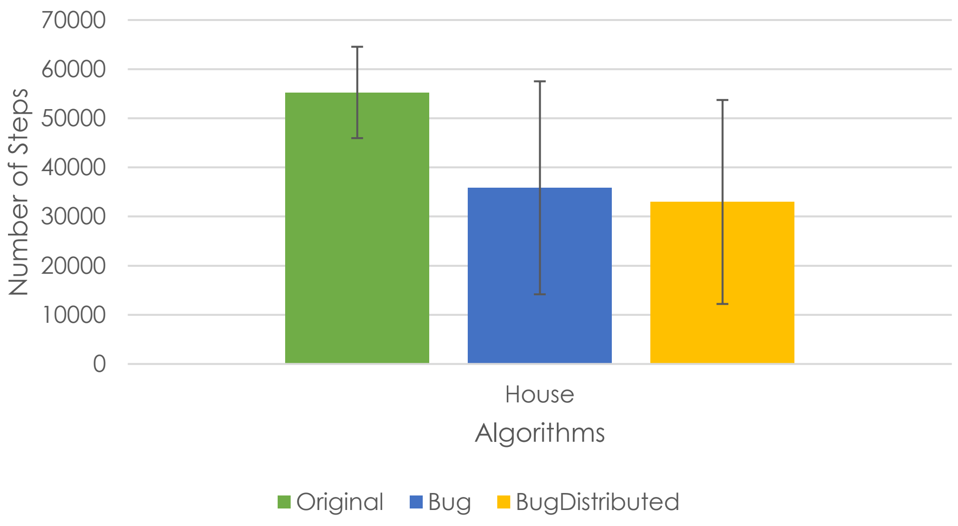

Figure 5.22.: Results of Frontier Detection adaption, House, 500×500 cells

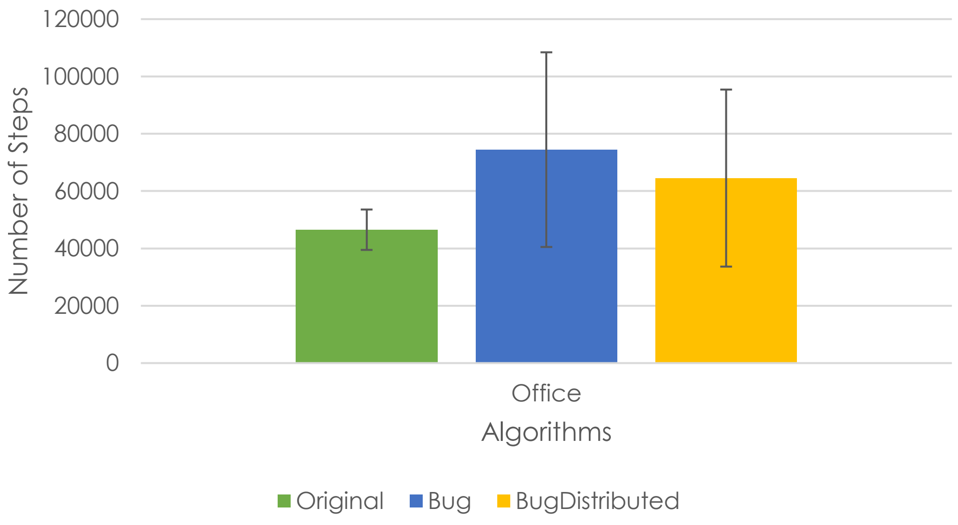

Figure 5.23.: Results of Frontier Detection adaption, Office, 500×500 cells

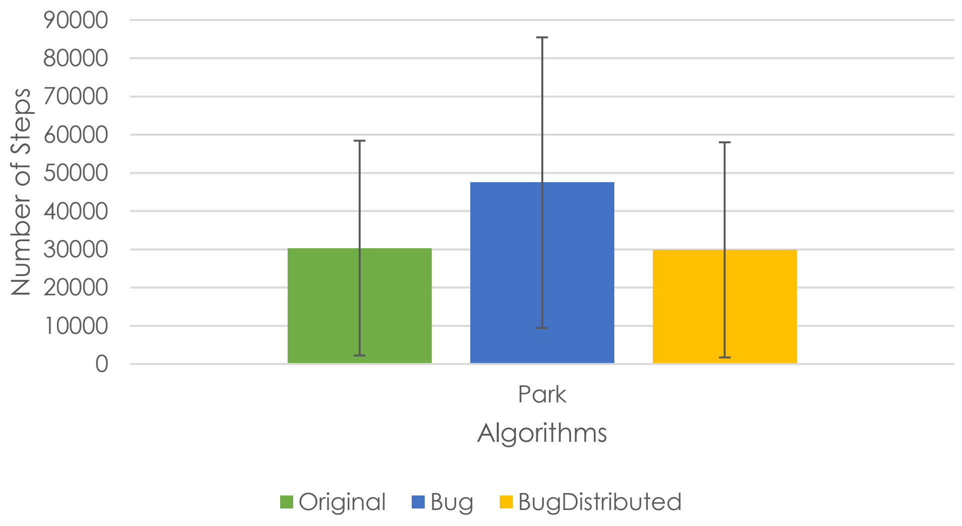

Figure 5.24.: Results of Frontier Detection adaption, Park, 500×500 cells

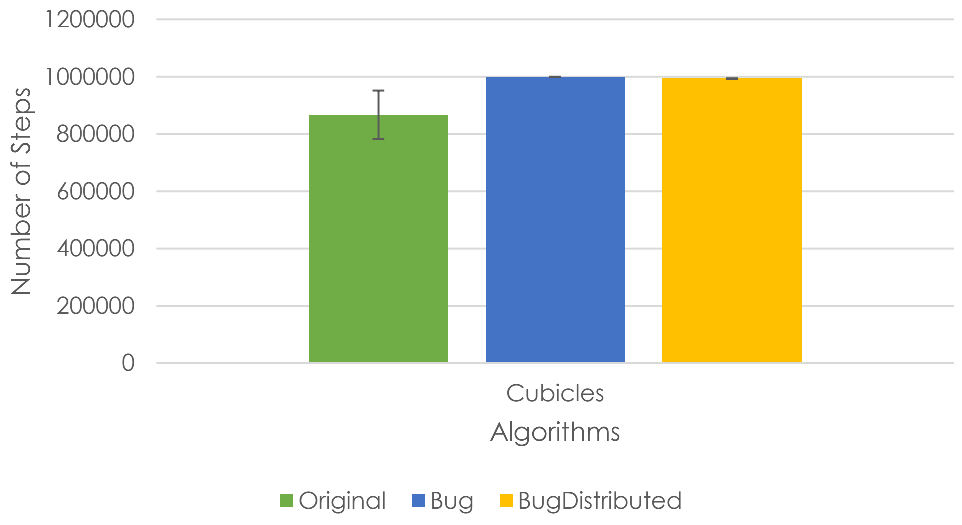

Figure 5.25.: Results of Frontier Detection adaption, Cubicles, 1000×1000 cells

the worst case of the *Bug* variant needed only about as much steps as the best case of the original version. The *BugDistributed* algorithm had about the same run time as the original algorithm, but the variance of these simulations is a little bit higher.

The results of the House scenario also show, Figure 5.22 a speed up of the newer algorithms. In this case both newer variants offer a better runtime speed up then the original algorithm. The *Bug* and *BugDistributed* algorithms need only around half the number of steps that the original algorithm needed to finish the simulation. The variance exhibited by the newer versions of the algorithm is also greater in this test case, then the variance of the older version of the algorithm.

Figure 5.24 on the other hand illustrates that the original algorithm is as fast as the *BugDistributed* variant in the case of the Park map. The *Bug* algorithm has an even worse runtime than the other two algorithms. The computation using the last 500×500 map, Office, turns the result of the House simulation on its head, as the original algorithm is the fastest, nearly using only half the steps that the *Bug* version needed. Although the best cases of each algorithm needed about the same steps, 20815 and 21765 steps for the *Bug* and *BugDistributed* algorithm respectively, and 23974 steps for the original algorithm.

This outcome is also true for the bigger map, Cubicles, as can be seen in Figure 5.25. The original algorithm is the fastest in this case. Although only half the simulations of the older algorithm finished in time, only one simulation of thirty of the *BugDistributed* algorithm was able to terminate successfully. All other simulations run into the step number termination cap of 1,000,000 steps.

To investigate these results further an additional simulation was started on the Cubicles map. This time the number of agents was increased to 45, leaving the other parameters the same. The diagram in Figure 5.26 shows the best case results from the different simulations in comparison with the previous results.

The colors are the same as in the previous diagrams, the green bar represents the grid-less, *Original*, algorithm, the blue bar the *Bug* algorithm, and the yellow bar the *BugDistributed* algorithm. On the left hand of the chart are the results using 30 agents and on the right hand the results using 45 agents

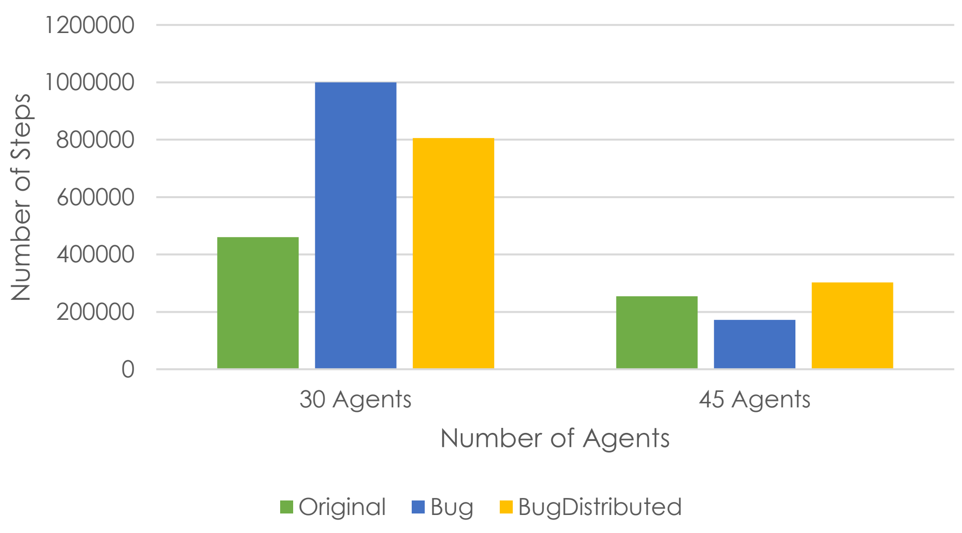

Figure 5.26.: Best case comparison between 30 and 45 agents, Cubicles, 1000×1000 cells

are depicted. The main difference between this chart and the previous diagrams is, that it compares the best cases, meaning the results which needed the least steps. This difference in illustration was chosen, as only about 3.3% to 6.6% of the simulations of the newer algorithms finished successfully, while about 96.6% of the simulated runs of the original algorithm finished successfully.

As expected the additional number of agents yielded a better result for each variant of the algorithm. Even if most of the simulations could not finish either in time or with not enough victims found, the simulations which were able to finish successfully did so while using only about 200.000 to 300.000 steps compared to the previous simulation. The best case from the simulation using only 30 agents needed about 800.000 steps. Even the *Original* algorithm was able to offer a veritable speed up while using 50% more agents: the results dropped from about 450.000 steps to 250.000 steps.

## 5.3.5. Summary of the Results

The simulations using the newer and adapted algorithms have shown that the changes made to the Multi-Agent Flood algorithm are working, especially if there are enough agents available. The inherent drawback of these changes is that the Frontier Detection algorithm will force the agents to spread out. This is on one hand the desired behavior, as the agents will be able to cover more ground this way, without exploring the same area again and again. On the other hand this also introduces a new drawback due to the conservative restrictions applied to the direct communication model. The agents have to create a line-of-sight connection between themselves and need to be in a specific range to initiate a direct communication. The increased spread of the agents will work against this and in turn decrease the cooperation and communication between the agents. The results on the House map, Figure 5.22, show that if there are enough agents available to often enable direct communication between the various agents, these changes lead to an impressive speed up. But if the map is too big and the agents are too far away from each other no communication between the agents may be possible, thus reducing the cooperation and decreasing the speed up or leading to an even worse result, as depicted in Figure 5.25.

Adding further agents yields a better result, but the number of agents has to be increased a lot, to turnout a constant improvement. Using 45 instead of 30 agents has shown improvements, compare Figure 5.26, but not enough to offer a the same impressive result that can be seen on the House map.

The introduction of these adaptions to the Multi-Agent Flood algorithm changed the cost-value ratio that additional agents offer. This also in turn changed the ratio of the agents towards the size of the area. Previous simulations in Chapter 5 show that around thirty agents are enough, even in the $1000 \times 1000$ map, to offer usable results. By using the Frontier Detection algorithm the number of agents has to be increased, especially on the bigger maps, to gain better results reliably.

## 5.4. Changing the size of the agents

The next step to introduce increased realism to the agents is to allow them to have a specific size. Changing the Multi-Agent Flood algorithm to include variables for the size of possible robots allows the scenarios to reflect the scale of different areas.

The change to the size of the agents also includes some changes in the behavior and in the way the sensors are modeled. Currently an agent only sees the nearest environment with its sensors and will only mark this seen environment in its stored map. In the first iteration of the Multi-Agent Flood algorithm, proposed in Chapter 3, the agents have a sensor range of one cell. Meaning that the agent can only see the surrounding cells of the current position and can only act on this information. Due to the changes to the algorithm, and inherently the agents, in Chapter 5 this sensor range was converted but not really extended. In current version of the algorithm the agents also can only act on the surrounding environment.

Another development which is introduced due to this change is the possibility to stagger the start of the agents in the starting point. As the starting point is needed for the map merging algorithm the agents can not start from different points. There are two reasons for the introduction of a staggered start from the same point: first, if an agent starts at a later time, it can use the information about the terrain already explored by its predecessors; second, it removes the abstraction that multiple agents can exist in the same place.

### 5.4.1. Modifications to the agents

Introducing a size is rather straight forward. Each agent now has a specific length and width in the algorithm. This in turn means that the movement of the agents had to be modified and the scan range for walls in front of the agents has been expanded, based on the chosen size of the agents. If an agent encounters a wall in front of it, it stops its movement and the agent backs up for some steps. This reverse movement takes some steps, to ensure that the distance between the obstacle and the agent is big enough to allow the agent a successful turn. To prevent a loop in the algorithm and a continuous cycle between moving towards the wall and backing up, the reverse movement includes a random angle.

The increased sensor range is implemented by using the line drawing algorithm presented in Bresenham (1965) and the midpoint circle algorithm, c.f. Van Aken (1984). The current position of the agent is the center of the circle with the sensor range as the radius. The midpoint circle algorithm rasterizes the circle using the center and the radius and returns a set of points. This set of points is used with the line drawing algorithm to scan the surrounding area around the agent by drawing a line from the agent to each of

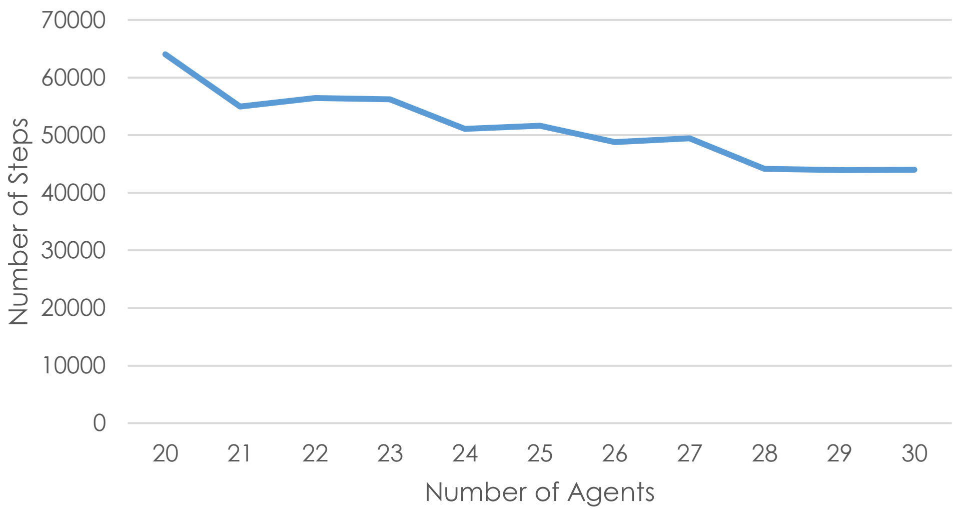

Figure 5.27.: Results of agent size adaption, House, 500×500 cells

the points contained in the set. The line is drawn from the center of the circle and stops if either the end point is reached or an obstacle is encountered. This behavior is similar to a laser range finder and allows for a variable sensor range.

### 5.4.2. Simulation of the algorithm with bigger agents

The revised algorithm was simulated with a size of 3×3 for the agents, instead of a size of 1×1 for the previous variants. Additionally a scan range of 12 instead of 1 was used. The test cases for this simulation are the same as previously used: House, Office, Park (each with a size of 500×500 cells), and Cubicles (1000×1000 cells). The number of agents ranged from twenty to thirty agents in the different configurations.

The charts depicted in Figures 5.27 to 5.30 present the results of the simulations. As usual the X-axis represents the number of agents used in the different configurations and the Y-axis shows the number of steps. Each point is the resulting average of thirty simulations.

The results of all four diagrams show that the modified algorithm works and the number of steps that are needed for a successful termination of the algorithm (as soon as 95% of the victims are found and 95% of the terrain is explored) is far lower than before. The greater sensor range allows for a

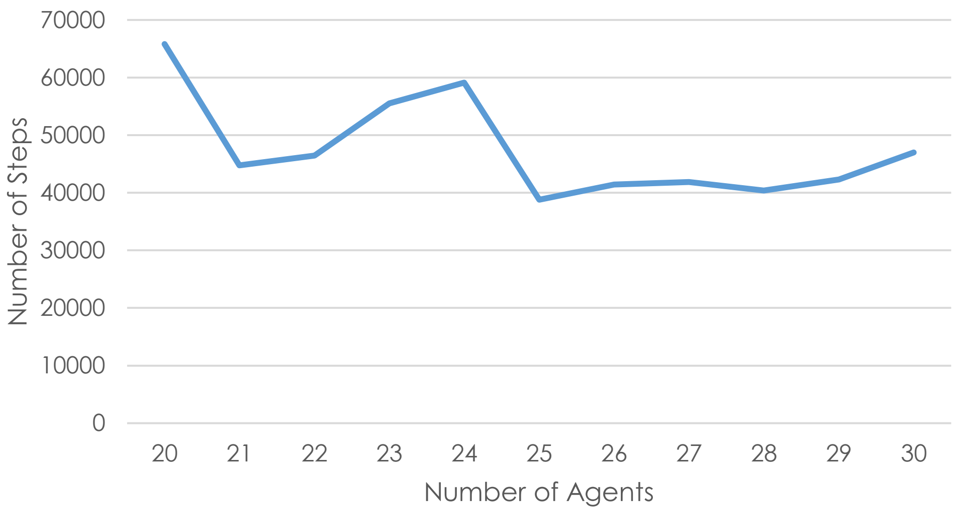

Figure 5.28.: Results of agent size adaption, Office, 500×500 cells

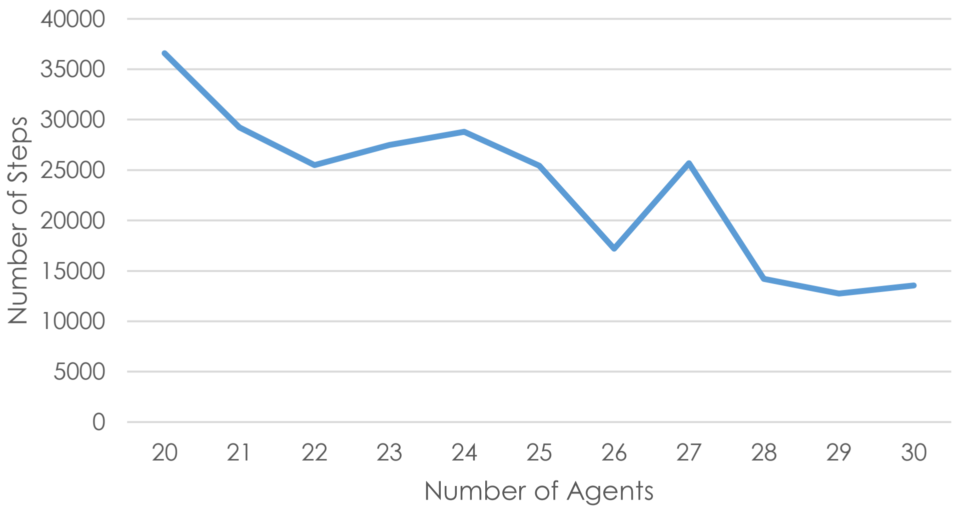

Figure 5.29.: Results of agent size adaption, Park, 500×500 cells

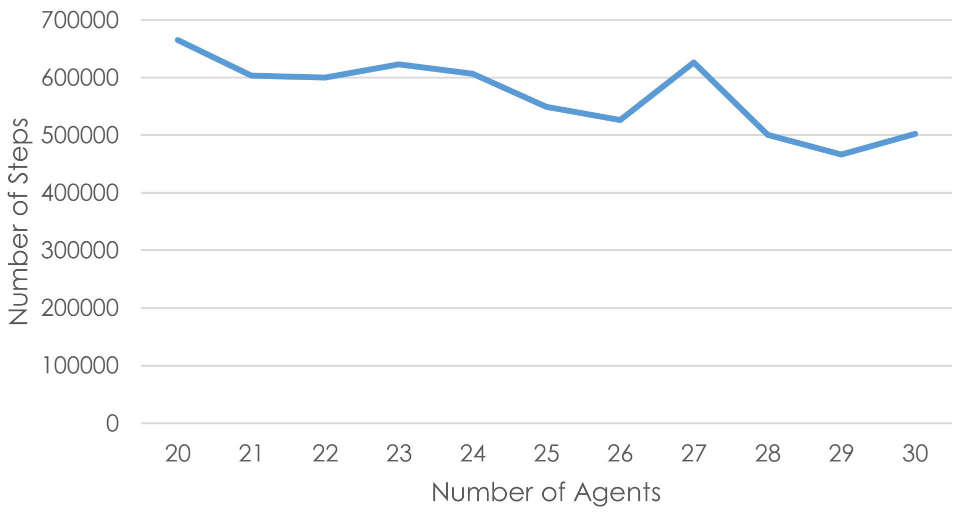

Figure 5.30.: Results of agent size adaption, Cubicles, 1000×1000 cells

faster exploration of the terrain, yielding better results in all four cases. This is especially the case in diagram 5.29, displaying the results from the Park test case (which is introduced in Figure 4.7). Comparing the new results with the results of the former iteration of the algorithm signifies the advantage of the modifications: Figure 5.24 shows that the *BugDistributed* algorithm needs about 30.000 steps to finish the search process successfully; in turn the outcome of the newer simulations yield an average result of around 14.000 steps for a successful termination of the modified algorithm. The modifications halved the number of steps despite the size of the changed agents, as the larger agents may not reach into all cracks and crannies available on this test case.

### 5.4.3. Comparison with Rendezvous

The modifications presented in this section make the Multi-Agent Flood algorithm in some ways similar to the Rendezvous algorithm, introduced in Chapter 2. The main difference is still the different way how the communication back to the base is achieved. The Rendezvous algorithm uses the multiple agents to create some sort of relay network, calculating specific points in the already explored terrain which would offer the best position for a relay agent. If anything of interest is found by an exploring agent, this agent will try to contact the next relay agent, which in turn will try to use the existing network of relay agents to transfer the information back to the base.

This different approach also reflects the way how the Rendezvous agents are able to communicate. The MAF agents face two specific constraints before a direct communication channel is allowed: first, to have a line of sight connection, and, second, to be in a specific range of each other. The Rendezvous algorithm uses "... a standard path loss model with a wall attenuation factor...", c.f. (De Hoog et al., 2010, p. 6) and Bahl and Padmanabhan (2000). With the use of this method the agents are able to communicate through walls combined with a loss of communication range.

The comparison between the two algorithm was done on the map illustrated in Figure 5.31. The map also includes ten points of interest for the MAF algorithm to find. The Rendezvous algorithm will ignore these points and will simply try to explore the terrain. Each algorithm used four agents, whereas the agents of the MAF algorithm were changed to resemble the Rendezvous agents. This was done by increasing the sensor range and increasing the direct communication range, although without removing the line of sight restriction. To compare the two different algorithms the number of steps each algorithm took to finish the exploration was measured. For each simulator the definition of a step is the same: each agent can move once in the terrain during one step. The results of the simulation of the Rendezvous algorithm was taken from the work published in De Hoog et al. (2010), as the Multi-Robot Exploration Simulator [Spirin et al. (2014)], introduced in Chapter 2, did not work reliably. Especially the number of agents could not be increased without problems and the simulator stops the simulation after 3000 steps. Due to this restriction other exhaustive simulations were not really possible. Additionally further analysis of the source code revealed that a Rendezvous agent tries to move ten cells in a single step compared to the speed of a MAF agent, which tries to move one cell in a single step. As soon as the sensor range of the Rendezvous agents is turned down to the same level as the MAF agents the simulation will terminate unsuccessfully as the cap of 3000 steps is reached without exploring 98% of the terrain.

The chart in Figure 5.32 shows the result of the simulation. The X-axis represents the two different algorithms and the Y-axis depicts the number of steps it took the algorithms to either explore 98% of the environment

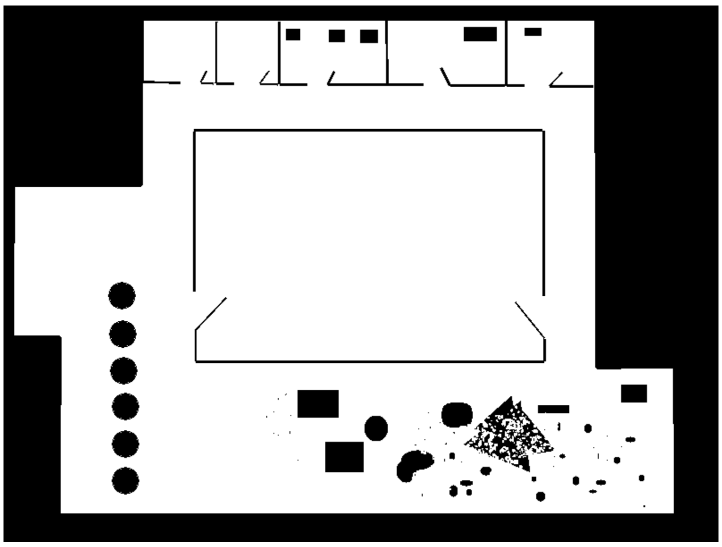

Figure 5.31.: Scenario for the MAF and Rendezvous comparison

or reach the respective cap of the simulation run. The cap for the MAF algorithm was set at 480.000 steps (the length multiplied by the width of the scenario), while the MRESim terminates the simulation after 3000 steps. The green bar depicting the Multi-Agent Flood algorithm shows the mean of 30 simulations. The blue bar shows the result of the Rendezvous algorithm, taken from (De Hoog et al., 2010, Fig. 6), as a comparable simulation using the MRESim was not possible. This publication was also not very clear about the number of simulations that were calculated to reach this result. Due to these restrictions the verification of the MAF algorithm using the MRESim was not really possible.

Figure 5.32 illustrating the results of the modified MAF agents shows that a simple comparison between the two algorithm is not really straight forward. The results demonstrate that the Rendezvous algorithm needed fewer steps than the MAF algorithm. The main reason for this big discrepancy is the ability of the Rendezvous algorithm to communicate through walls and the need of the MAF agents to return to the base after a point of interest was found. Additionally the distance crossed during a step of a single Rendezvous agent was ten times the length of a single MAF agent. The Rendezvous algorithm ignored the points of interest and only tried to achieve a 98% exploration of the terrain.

Although the comparison was possible it is not quite as clear as the data

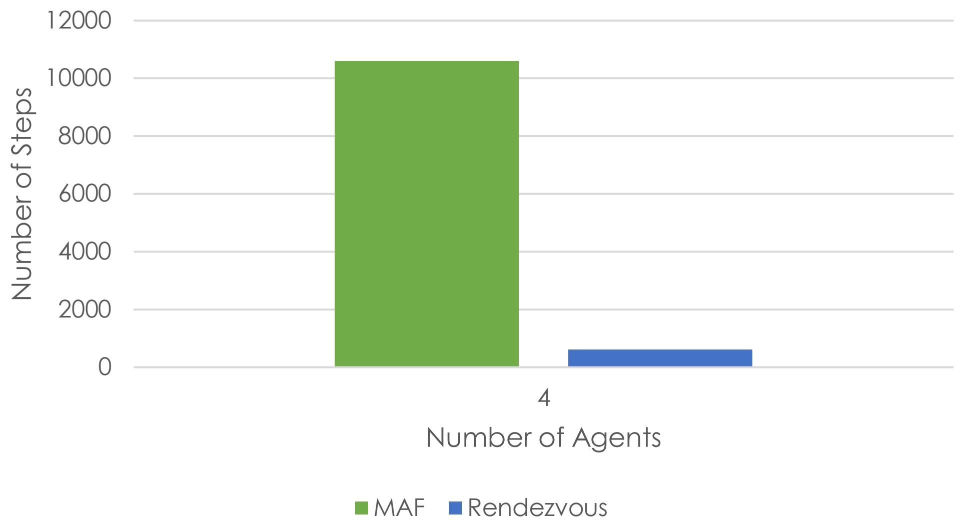

Figure 5.32.: Results of the comparison between MAF and Rendezvous

suggests. The different approaches still vary in the end. On the other hand the Multi-Agent Flood algorithm would also benefit from the direct communication model used by the Rendezvous algorithm, instead of simply restricting the transmission by line of sight connection. Another point is the robustness of the algorithms. If the agents of the Rendezvous algorithm are not able to communicate directly they will start to move towards each other to exchange messages using a line-of-sight connection.

## 5.5. Chapter Summary

The previous chapters in this work introduced the first concepts of the Multi-Agent Flood algorithm and offered some steps to reduce the time needed to explore the unknown terrain and to find as much victims as possible in this time frame. The modifications proposed in this chapter, Chapter 5, tried not to improve the search time of the algorithm but tried to adapt the algorithm for the use with real hardware.

The first step in this direction was to remove the grid abstraction introduced in the first version of the MAF algorithm. Additionally another way to store the markings had to be introduced and the concept of a mesh of semi-passive RFID chips was conceived for the dispersion of markings that are read and

| Name | Goal | Communi-cation | Movement | Information back to HQ | Maintain-ability | Size |
|---|---|---|---|---|---|---|
| Ants | PoIs | Indirect | Grid-based | ✗ | ✗ | ✗ |
| Brick&Mortar | Explore | Indirect | Grid-based | ✗ | ✗ | ✗ |
| HybridExploration | Explore | Indirect | Grid-based | ✗ | ✗ | ✗ |
| Rendezvous | Explore | Direct | Grid-less | ✓ | ✗ | ✗ |
| MAF | Both | Both | **Grid-less** | ✓ | ✓ | ✓ |

Table 5.1.: Comparison of the different algorithms, MAF agents now move in a grid-less environment and a size can be specified

write able. Simulations did show that the use of RFID chips is a viable alternative for the grid abstraction, which is also quite robust, depending on the rate of dispersal of the chips in the environment.

The next change to the algorithm was the addition of the frontier-based exploration concept to the agents. With the help of this concept the agents are now able to use the collected map data more efficiently as they are now able to choose specific spots to explore instead of exploring the area randomly. This change also introduced a more wide spread movement pattern of the agents, which turned out to be a disadvantage towards the direct communication due to the range restrictions introduced in Chapter 4. As a consequence more agents are needed to reap all benefits offered by the frontier-based exploration approach.

The last change introduced in this chapter was the modification of the agent size. Allowing for an increased size of the agent grants the possibility of more realism. This modification also included a more variable sensor range.

All of the alterations introduced in this chapter allow for an adaption of the Multi-Agent Flood algorithm on real hardware agents. The removal of the grid abstraction and the addition of a variable size of the agents eliminated the main points which made its application unfeasible.

The comparison with the Rendezvous algorithm offered some additional insights how the MAF algorithm could be further improved. Implementing a better model for the radio signals used during the direct communication would remove the line of sight restriction from the MAF algorithm. Also the way the frontier points are chosen could be evaluated, so that the agents choose more beneficial points, which in turn may increase the knowledge about the unknown terrain.

Table 5.1 depicts the changes to the MAF algorithm: instead of a grid-based movement the agents are now able to move in a grid-less environment, and the size of the agents is now configurable.

Each multi-agent algorithm for search & rescue applications should follow the points introduced at the end of Chapter 2:

1. The agents should be able to operate in extreme terrain and operating conditions
2. The algorithm has to function in GPS- and wireless-denied environments
3. The agents have to act autonomously without any help of a human teleoperator
4. The algorithm should find as many victims as possible in the shortest time frame

The Multi-Agent Flood algorithm is able to operate in extreme terrain and operating conditions, as this is mostly dependent on the hardware used. The model of layering communication allows for a robust communication channel. This in turn allows the algorithm to keep working even if the direct communication using radio signals is not available. The algorithm does also not rely on any additional help from human team members for the search process. The algorithm is also the only one that incorporates a possible maintenance of its agents and allows for a configuration of the agent size.

CHAPTER 6.

# CONCLUSION

## 6.1. Summary of the Thesis

Advances in robotics allow for an ever increasing way on how to use these robots to help society in numerous ways. From automating whole factories and industries to performing medical surgery on its own without any human intervention. The point of research would be on how robots can aid and help further. This work introduced a new concept of a multi-agent system of robots for the use in search and rescue scenarios.

The first steps of this concept were the creation of a new algorithm for a multi-agent system, which is able offer a decentralized control structure for multiple agents. As this algorithm was intended for a search and rescue scenario specific constrains had to be kept in mind. One of the main constraints which severely limit the way on how the algorithm can work is the complete lack of a global positioning system. This means that the various agents can not really specify at which point in the scenario they are. They can only rely on their own sensors to gather enough information to pinpoint their location. The second constraint that hinders the agents is the way how they can communicate with each other. Due to the same reason that a GPS is not available a communication via radio signals may not always be possible in a search and rescue scenario. This is a huge drawback as a multi-agent system relies heavily on the possibility of communication between its agents, as without any communication not cooperation is possible and each agent would work on its own.

The Multi-Agent Flood algorithm proposed in Chapter 3 of this work addresses these two constraints and offers a multi-agent system that is able to find possible victims in unknown terrain without relying on a global positioning system and without communicating via radio signals. As direct data exchange is not possible between the different agents, each agent uses the concept of stygmergy which can also be seen in nature, as it is also used by ants to communicate with each other and the rest of the hive. Ants will spray pheromone marks in the environment which other ants can perceive and act upon. Using this concept the agents are able to leave markings in the environment, in this case information about the current known distance to the starting point, which can be used by other agents to find the current known shortest path back from this point to the headquarter. By using this means of indirect communication the agents have no need of pinpointing their exact location in the terrain and they are able to communicate with each other, resulting in a cooperation and a working multi-agent system. This system is able to find points of interest in an unknown terrain and relay the information about the found points of interest back to the starting point, as the rest of the agents are still searching. This allows a possible human rescue team to start the rescue of victims in parallel to the still ongoing search process of the agents. The ability to always find the currently shortest known path back to the base also allows for an easy maintenance of the various robots. For example if the battery charge of one agent is getting low, it can decide to head back to the base in time to switch the battery or recharge it.

The next improvement to the performance of the Multi-Agent Flood algorithm is to take a look at the constraints of the direct communication. Sending radio signals from one agent to another is usually not a foolproof way to initiate a direct data transfer between two agents. Various protocols exist to facilitate this exchange of information using wireless communication, but in unknown terrain there are just too many variables to rely only on this kind of communication. For example an agent in an urban scenario, moving through a broken down building can not depend on this to be able to cooperate with the other agents. Thus wireless data transmission is not ruled out from the start but for the agents to use it some very conservative assumptions have to be made. The first presumption which has to be stated is that direct communication between to agents is only reliably working as long as a line of sight between these two agents exists. If this is the case one can assume that the radio signals are not hindered by either one of the following: absorption, refraction, reflection, or other occurrences that will distort the wave and essentially will prevent the fault free sending of information. The second assumption, besides the line of sight between two agents, is the range between said agents. An error free data transmission can be assumed if the agents are both in a specific range of each other, coupled with a line of sight this should provide an accurate and precise connection between those two agents using radio signals. Using these assumptions allows the merge of two communication models: indirect communication using markings on the ground and direct communication via radio signals. This in turn enables the agents of the Multi-Agent Flood algorithm to store a map of the sensed environment and exchange this map via direct communication as soon as the requirements for the data transmission are met. If direct communication is not possible the agents are still able to use the markings on the ground to find their way back to the starting point to share the information about any found points of interest. The additional possibility of sending data from one agent to another creates a speed up of the search time, which in turn allows for a faster recovery of any possible victims found by the agents. Another advantage of this approach is, that it creates a robust communication framework, which will use radio signals as far as it is possible and will gracefully degrade to rely on markings on the floor to disseminate information between the agents and to allow cooperation for the multi-agent system to keep working.

The first variant of the algorithm introduced in this work was based on some abstractions to simplify the simulation and the creation of the first version of this concept. These abstractions need to be removed, so that this method is applicable to real world robots. For this to work the algorithm needs a way to store the markings on the ground for the indirect exchange of information in an other way. The current variant of the algorithm to this date used the cells of the grid for the movement of the agents and to store the markings. Decoupling the movement from the grid is rather easy to do, simply add a turning angle to each agent and give each of them a directional vector, which describes the movement direction. The markings on the ground were replaced by dropped RFID chips. Each agent will now drop a RFID tag at specific intervals, creating a breadcrumb trail, which allows the agent to follow it back to the starting point. As there are multiple agents moving through the terrain each of those breadcrumb trails will be combined into a loose mesh of semi-passive RFID tags. Each of those chips is able to store the distance measured from this point by the robot who dropped it back to the base, or if another robot visits the chip by an updated number, according to the algorithm. Simulations have shown that this approach is a viable proposal to replace the grid abstraction and allows for an adaption to “real world” problems. Furthermore the reliability of the dispersed RFID tags was evaluated. The environment could introduce faulty readings or chips could malfunction in general, disrupting the indirect communication part of the algorithm, which also is the backbone of the cooperation between the agents, as the direct communication part is seen only as a benefit and allows for additional information exchange.

With the creation of a robust communication model, using direct and indirect methods of data transmission between the agents, the agents are now able to exchange map information about the already explored environment. This in turn enables the agents to create more informed decisions about which parts of the map should be explored next to maximise the gain of “new information” from the hitherto unknown terrain. The introduction of the frontier-based exploration in Chapter 5 of this thesis enables the agents to calculate frontier points. These points offer target destination for the agents as they are exploring the scenario and promise the most gain in points of newly explored terrain. Coupling this navigation algorithm with a dynamic way finding algorithm allows the agents to reach these frontier points faster and enables the whole Multi-Agent Flood algorithm to spread the agents into more diverse parts of the yet unexplored parts of the map. This in turn also offers a speed up of the whole search process, as the various agents are now explicitly targeting unexplored terrain and further try to ignore already visited terrain. The use of the frontier points gained from the frontier-based exploration also enables the agents to share target points with each other as they are in range for direct communication. As soon as the agents are able to compare their target points they can now decide whether to keep the assigned target point or, if another agent is nearer this designated point, to recalculate the points and chose another target to explore. This way the agents can separate further and exploration is more evenly divided between the various agents.

Other adaptions towards the use with real hardware include the implementation of a specific size for the agents and more realistic variable sensor range. A further comparison with another existing algorithm showed that there are still possibilities for performance optimizations available, even if the two algorithms differ in the way their mission was accomplished.

## 6.2. Future Work

The concepts and the algorithm proposed in this thesis should be seen as a base for a working multi-agent system for search and rescue scenarios.

Every method used in the Multi-Agent Flood algorithm was specifically chosen to keep the agents as simple as possible. Current search and rescue robots are moving away from specialised hardware, which made the first variants of these robots very expensive and hard to maintain. As a consequence a controlling algorithm should not rely on special hardware, in particular if the controlling algorithm is based on a multi-agent system which needs a significant number of agents to be able to function. The ingrained simplicity and modularity of the agents also allows for the incorporation of future algorithms, that may offer more advantages or alternative ways of achieving things, for example other navigation algorithms.

The concept of layered communication offers the advantage of a very robust communication model, as the graceful degradation inherent in this method allows the algorithm to still finish the exploration even if one part of the possible data exchange between agents is not available anymore. Due to the very conservative restrictions applied to the direct part of the communication, there are still ways to improve this algorithm. For example new research or algorithms could offer a more reliable way for the direct communication to work. In this case the constraints placed on the Multi-Agent Flood agents could be lightened or even removed. Using more sophisticated radio signal models allows the agents to communicate through walls for example, removing the line of sight restriction. This would offer a tremendous advantage, as the agents could create ad-hoc networks at longer ranges. The resulting speed up of the data exchange would allow for an even faster exploration of the terrain and it would also allow the agents to keep in touch with the home base. This in turn could also remove the need to return to the base to forward the information about any victims found in the terrain.

Chapter 5 introduced algorithms and methods to enhance the intelligence of the agents. In the first few variants of the Multi-Agent Flood algorithm the agents were kept really simple, just being able to react to the immediate environment. With the help of the Wavefront Frontier Detection algorithm the agents are now able to specify points in the map which will offer the uncover the most unexplored terrain. In addition, as the agents will now target a specific point, they are now able to use the myriad variants of way finding algorithms. Other algorithms may offer additional advantages, by simply computing the optimal path between the current location and the targeted point. The map data created and stored by the agents can also be used for other things. With the use of additional sensors the robots could enhance the stored data and this in turn would allow the robots create a more detailed map, which would offer more data for other algorithms to use. These additional sensors could also lead to a faster exploration and faster recognition of victims in the surrounding environment.

Another part, which invites further research is the use of the MAF algorithm in a dynamic environment. How would the agents handle disappearing walls or react to newly created opening. The current model assumes victims that are remaining in their place without moving around. What would happen if victims are able to move, especially if they may see a searching robot before the robot could sense them, or on the other hand that the victim unwittingly escapes the searching agent?

Last but not least further advances in the RFID technology could decrease the size of the active chips. By switching from semi-passive chips to active chips the dispersed tags in the environment could form an ad-hoc network of its own. This would increase the communication capability of the whole system. The robots would now be able to disperse a whole network step by step in the already explored terrain. Depending on the throughput of this network the human rescue team could receive a live sensor stream from each of the agents in range of the RFID ad-hoc network, allowing the team members to get a immediate update on the found victims. This would enable the rescue team to start an already informed rescue process.

A dispersed ad-hoc network could also offer victims who are still able to move by themselves and who are in possession of a mobile phone to access this network and call for aid or receive detailed instructions from the rescue team were they can reach the nearest first aid station.

In the end the concepts presented in this thesis are also applicable in other scenarios. For example the agents could be used to help to simply find points of interest in an area where a wireless network will not work reliably. Changing the sensor layout would also allow the agents to search for mines in a minefield for example.

APPENDIX A.

# REFERENCES

Andonian, S., Z. Okeke, D. A. Okeke, A. Rastinehad, B. A. VanderBrink, L. Richstone, and B. R. Lee
2008. Device failures associated with patient injuries during robot-assisted laparoscopic surgeries: a comprehensive review of FDA MAUDE database. *Canadian Journal of Urology*, 15(1):3912.

ANSI/IEEE
1991. *Standard Glossary of Software Engineering Terminology (ANSI).* The Institute of Electrical and Electronics Engineers Inc.

Baglietto, M., G. Cannata, F. Capezio, A. Grosso, and A. Sgorbissa
2009. A multi-robot coordination system based on RFID technology. In *International Conference on Advanced Robotics*, Pp. 1–6. IEEE.

Bahl, P. and V. N. Padmanabhan
2000. Radar: An in-building rf-based user location and tracking system. In *INFOCOM 2000, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, volume 2, Pp. 775–784. IEEE.

Baranzadeh, A. and A. V. Savkin
2016. A distributed control algorithm for area search by a multi-robot team. *Robotica*, Pp. 1–21.

Becker, M., F. Blatt, and H. Szczerbicka
2013. A multi-agent flooding algorithm for search and rescue operations in unknown terrain. In *Multiagent System Technologies*, Pp. 19–28. Springer.

Becker, M., F. Blatt, and H. Szczerbicka
2014. A concept of layered robust communication between robots in

multi-agent search & rescue scenarios. In *IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Pp. 175–180. IEEE.

Becker, M., S. Schaust, and E. Wittmann
2007. Performance of routing protocols for real wireless sensor networks. In *Proceedings of the 10th International Symposium on Performance Evaluation of Computer and Telecommunication Systems*. Citeseer.

Bellman, R.
1972. *Dynamic Programming*. Princeton, New Jersey: Princeton University Press.

Birk, A.
2004. Fast robot prototyping with the cubesystem. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04.*, volume 5, Pp. 5177–5182. IEEE.

Birk, A., S. Markov, I. Delchev, and K. Pathak
2006a. Autonomous rescue operations on the IUB rugbot. In *IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR). IEEE Press*. Citeseer.

Birk, A., K. Pathak, S. Schwertfeger, and W. Chonnaparamutt
2006b. The IUB Rugbot: an intelligent, rugged mobile robot for search and rescue operations. In *IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR). IEEE Press*.

Blatt, F., M. Becker, and H. Szczerbicka
2015. Optimizing the exploration efficiency of autonomous search and rescue agents using a concept of layered robust communication. In *IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, Pp. 1–6. IEEE.

Blatt, F. and H. Szczerbicka
2016. Realisation of navigation concepts for the multi-agent flood algorithm for search & rescue scenarios using rfid tags. In *IEEE/ACM 20th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Pp. 112–115. IEEE.

Blatt, F. and H. Szczerbicka
2017. Combining the multi-agent flood algorithm with frontier-based exploration in search & rescue applications. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2017*. IEEE.

Bresenham, J. E.
1965. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30.

Cha, J.-R. and J.-H. Kim
2005. Novel anti-collision algorithms for fast object identification in rfid system. In *Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on*, volume 2, Pp. 63–67. IEEE.

Cheng, T. and L. Jin
2007. Analysis and simulation of rfid anti-collision algorithms. In *Advanced Communication Technology, The 9th International Conference on*, volume 1, Pp. 697–701. IEEE.

Cherkassky, B. V., A. V. Goldberg, and T. Radzik
1996. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical programming*, 73(2):129–174.

Chinnaiah, M., S. Dubey, L. Vineela, K. Bindu, and E. B. Babu
2016. An unveiling path planning algorithm with minimal sensing using embedded based robots. In *2016 International Conference on Advances in Human Machine Interaction (HMI)*, Pp. 1–5. IEEE.

Crespi, A., A. Badertscher, A. Guignard, and A. J. Ijspeert
2005. Amphibot i: an amphibious snake-like robot. *Robotics and Autonomous Systems*, 50(4):163–175.

Davids, A.
2002. Urban search and rescue robots: from tragedy to technology. *IEEE Intelligent Systems*, 17(2):81–83.

De Cubber, G., D. Doroftei, D. Serrano, K. Chintamani, R. Sabino, and S. Ourevitch
2013a. The eu-icarus project: developing assistive robotic tools for search

and rescue operations. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Pp. 1–4. IEEE.

De Cubber, G., D. Serrano, K. Berns, K. Chintamani, R. Sabino, S. Ourevitch, D. Doroftei, C. Armbrust, T. Flamma, and Y. Baudoin
2013b. Search and rescue robots developed by the european icarus project. In *7th Int. Workshop on Robotics for Risky Environments*. Citeseer.

De Hoog, J., S. Cameron, and A. Visser
2010. Selection of rendezvous points for multi-robot exploration in dynamic environments. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Dijkstra, E. W.
1968. Cooperating sequential processes. In *The origin of concurrent programming*, Pp. 65–138. Springer.

Dorigo, M. and L. Gambardella
1997. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66.

Dorigo, M., V. Maniezzo, and A. Colorni
1996. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41.

Durrant-Whyte, H. and T. Bailey
2006. Simultaneous localization and mapping: part i. *Robotics & Automation Magazine*, 13(2):99–110.

Ferranti, E., N. Trigoni, and M. Levene
2007. Brick & Mortar: an on-line multi-agent exploration algorithm. In *IEEE International Conference on Robotics and Automation*, Pp. 761–767. IEEE.

Ferranti, E., N. Trigoni, and M. Levene
2009. Rapid exploration of unknown areas through dynamic deployment of

mobile and stationary sensor nodes. *Autonomous Agents and Multi-Agent Systems*, 19(2):210–243.

Grassé, P.-P.
1959. La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6(1):41–80.

Grisetti, G., R. Kümmerle, C. Stachniss, and W. Burgard
2010. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43.

Harabor, D. D., A. Grastien, et al.
2011. Online graph pruning for pathfinding on grid maps. In *AAAI*.

Hart, P. E., N. J. Nilsson, and B. Raphael
1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.

Howard, A., L. Parker, and G. Sukhatme
2006. Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *The International Journal of Robotics Research*, 25(5-6):431–447.

Icking, C., T. Kamphans, R. Klein, and E. Langetepe
2005. Exploring simple grid polygons. In *International Computing and Combinatorics Conference*, Pp. 524–533. Springer.

IEEE
2003. Wireless lan medium access control (MAC) and physical layer (PHY) specifications - amendment 4: Further higher-speed physical layer extension in the 2.4 ghz band. *IEEE Std 802.11g-2003*.

IEEE
2009. IEEE standard for information technology – local and metropolitan area networks – specific requirements – part 11: Wireless lan medium access control (MAC) and physical layer (PHY) specifications amendment 5: Enhancements for higher throughput. *IEEE Std 802.11n-2009 (Amendment*

*to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, Pp. 1–565.

Ito, K. and H. Maruyama
2016. Semi-autonomous serially connected multi-crawler robot for search and rescue. *Advanced Robotics*, 30(7):489–503.

Jünger, M., G. Reinelt, and G. Rinaldi
1995. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330.

Keidar, M. and G. A. Kaminka
2012. Robot exploration with fast frontier detection: theory and experiments. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, Pp. 113–120. International Foundation for Autonomous Agents and Multiagent Systems.

Khaliq, A. A., M. Di Rocco, and A. Saffiotti
2014. Stigmergic navigation on an RFID floor with a multi-robot team. In *Workshop on Multi-Agent Coordination in Robotic Exploration, Prague, Czech Republic*, Pp. 1–6.

Kitano, H. and S. Tadokoro
2001. Robocup rescue: A grand challenge for multiagent and intelligent systems. *AI Magazine*, 22(1):39.

Kitano, H., S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada
1999. Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics*, volume 6, Pp. 739–743. IEEE.

Kitano, H., M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda, and M. Asada
1998. The robocup synthetic agent challenge 97. In *RoboCup-97: Robot Soccer World Cup I*, Pp. 62–73. Springer.

Klair, D. K., K.-W. Chin, and R. Raad
2010. A survey and tutorial of rfid anti-collision protocols. *IEEE Communications Surveys & Tutorials*, 12(3):400–421.

Koenig, S. and Y. Liu
2001. Terrain coverage with ant robots: a simulation study. In *Proceedings of the fifth international conference on Autonomous agents*, Pp. 600–607. ACM.

Koenig, S., B. Szymanski, and Y. Liu
2001. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, 31(1):41–76.

Kohlbrecher, S., J. Meyer, T. Graber, K. Kurowski, and O. von Stryk
2015. Robocuprescue 2015-robot league team hector darmstadt (germany). Technical report, Technische Universität Darmstadt.

Luke, S., G. C. Balan, L. Panait, C. Cioffi-Revilla, and S. Paus
2003. Mason: A java multi-agent simulation library. In *Proceedings of Agent 2003 Conference on Challenges in Social Simulation*, volume 9.

Luke, S., C. Cioffi-Revilla, L. Panait, and K. Sullivan
2004. Mason: A new multi-agent simulation toolkit. In *Proceedings of the 2004 swarmfest workshop*, volume 8, P. 44.

Luke, S., C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan
2005. Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527.

Lumelsky, V. and A. Stepanov
1986. Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE transactions on Automatic control*, 31(11):1058–1063.

Marsh, L. and C. Onof
2008. Stigmergic epistemology, stigmergic cognition. *Cognitive Systems Research*, 9(1):136–149.

Murphy, R. R.
2014. *Disaster robotics*. MIT press.

Murphy, R. R., S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, and A. M. Erkmen
2008. Search and rescue robotics. *Handbook of Robotics. Springer*, Pp. 1151–1173.

Nagatani, K., S. Kiribayashi, Y. Okada, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, and Y. Hada
2011. Redesign of rescue mobile robot Quince. In *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Pp. 13–18. IEEE.

Nguyen, H. G., N. Pezeshkian, A. Gupta, and N. Farrington
2004. Maintaining communications link for a robot operating in a hazardous environment. Technical report, DTIC Document.

Pfingsthorn, M., B. Slamet, and A. Visser
2008. A scalable hybrid multi-robot SLAM method for highly detailed maps. In *RoboCup 2007: Robot Soccer World Cup XI*, Pp. 457–464. Springer.

Rao, N. S., S. Kareti, W. Shi, and S. S. Iyengar
1993. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical report, Citeseer.

Rekleitis, I., V. Lee-Shue, A. P. New, and H. Choset
2004. Limited communication, multi-robot team based coverage. In *IEEE International Conference on Robotics and Automation*, volume 4, Pp. 3462–3468. IEEE.

Rohmer, E., T. Yoshida, K. Ohno, K. Nagatani, S. Tadokoro, and E. Koyangai
2010. Quince: A collaborative mobile robotic platform for rescue robots research and development. In *Proc. of Int. Conf. on Advanced Mechatronics*, Pp. 225–230.

Rooker, M. N. and A. Birk
2005. Combining exploration and ad-hoc networking in robocup rescue. In *RoboCup 2004: Robot Soccer World Cup VIII*, Pp. 236–246. Springer.

Sakakibara, T., D. Kurabayashi, et al.
2007. Artificial pheromone system using RFID for navigation of autonomous robots. *Journal of Bionic Engineering*, 4(4):245–253.

Sen, D., P. Sen, and A. M. Das
2009. *RFID for energy & utility industries.* Pennwell Books.

Spirin, V. and S. Cameron
2014. Rendezvous through obstacles in multi-agent exploration. In *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Pp. 1–6. IEEE.

Spirin, V., S. Cameron, and J. de Hoog
2013. Time preference for information in multi-agent exploration with limited communication. In *Conference Towards Autonomous Robotic Systems*, Pp. 34–45. Springer.

Spirin, V., J. de Hoog, A. Visser, and S. Cameron
2014. Mresim, a multi-robot exploration simulator for the rescue simulation league. In *Robot Soccer World Cup*, Pp. 106–117. Springer.

Svennebring, J. and S. Koenig
2004. Building terrain-covering ant robots: A feasibility study. *Autonomous Robots*, 16(3):313–332.

Takahashi, T., Y. Kitamura, and H. Miwa
2012. Organizing rescue agents using ad-hoc networks. In *Highlights on Practical Applications of Agents and Multi-Agent Systems*, Pp. 139–146. Springer.

Tarjan, R.
1972. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160.

Van Aken, J. R.
1984. An efficient ellipse-drawing algorithm. *IEEE Computer Graphics and Applications*, 4(9):24–35.

VDI4472 Blatt 10
2008. *Anforderungen an Transpondersysteme zum Einsatz in der Supply Chain; Testverfahren zur Überprüfung der Leistungsfähigkeit von Transpondersystemen (RFID).* Beuth Verlag.

Verma, L., M. Fakharzadeh, and S. Choi
2013. WiFi on steroids: 802.11 ac and 802.11 ad. *IEEE Wireless Communications*, 20(6):30–35.

Von Neumann, J. and A. Burks
1966. Theory of self-replicating automata. *Urbana: University of Illinois Press.*

Weis, S. A.
2007. *RFID (radio frequency identification): Principles and applications.* MIT CSAIL.

Witkowski, U., E. Habbal, M. A. Mostafa, S. Herbrechtsmeier, A. Tanoto, J. Penders, L. Alboul, and V. Gazi
2008. Ad-hoc network communication infrastructure for multi-robot systems in disaster scenarios. In *Proceedings of IARP/EURON Workshop on Robotics for Risky Interventions and Environmental Surveillance.*

Wooldridge, M.
2009. *An Introduction to MultiAgent Systems.* John Wiley & Sons.

Wright, C., A. Johnson, A. Peck, Z. McCord, A. Naaktgeboren, P. Gianfortoni, M. Gonzalez-Rivero, R. Hatton, and H. Choset
2007. Design of a modular snake robot. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pp. 2609–2614. IEEE.

Yamauchi, B.
1997. A frontier-based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Pp. 146–151. IEEE.

Yamauchi, B.
1998. Frontier-based exploration using multiple robots. In *Proceedings of the second international conference on Autonomous agents*, Pp. 47–53. ACM.

Ziparo, V. A., A. Kleiner, B. Nebel, and D. Nardi
2007. RFID-based exploration for large robot teams. In *IEEE International Conference on Robotics and Automation*, Pp. 4606–4613. IEEE.

## Curriculum Vitae

| | |
|---|---|
| Name: | Florian Andreas Blatt |
| Place of Birth: | Berlin-Steglitz |
| Date of Birth: | March 12, 1984 |

**Experience**

| | |
|---|---|
| 2011–2017 | Research Assistant, Leibniz University Hannover |

**Education**

| | |
|---|---|
| 2011–2017 | Ph.D. student, Computer Science, Leibniz University Hannover |
| 2009–2011 | Master of Science, Computer Science, Leibniz University Hannover |
| 2004–2009 | Bachelor of Science, Computer Science, Leibniz University Hannover |
| 2003–2004 | Alternative Civilian Service, Lothar-Wittko-Werkstatt, Stadthagen |
| 1995–2003 | Matura, Bundesgymnasium/Bundesrealgymnasium Sillgasse, Innsbruck, Österreich |
| 1994–1995 | Gymnasium Stormanschule, Ahrensburg |