THESIS ON INFORMATICS AND SYSTEM ENGINEERING C129

Covering Algorithms for a Robot Swarm with Limited Information

ANDRES PUUSEPP

TTÜ PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

This dissertation was accepted for the defence of the degree of Philosophy in Computer Science on April, 17, 2017.

Supervisor: Professor Tanel Tammet
Institute of Computer Science
Tallinn University of Technology
Tallinn, Estonia

Opponents: Professor Juha Röning
Computer Science and Engineering
University of Oulu

Professor Alvo Aabloo
Faculty of Science and Technology
Institute of Technology
University of Tartu

Defence of the thesis: December 12, 2017, Tallinn

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology has not been submitted for any academic degree.

/Andres Puusepp/

ISSN 1406-4731
ISBN 978-9949-83-153-1 (publication)
ISBN 978-9949-83-154-8 (PDF)

INFORMAATIKA JA SÜSTEEMITEHNIKA C129

Katvusalgoritmid piiratud teadmusbaasiga robotiparvede jaoks

ANDRES PUUSEPP

TTÜ
KIRJASTUS

Table of Contents

ABSTRACT

The main goal of the thesis is to investigate and develop algorithms using navigational tags for enhancing the performance of a swarm of robots when precise navigation is hard to achieve or not feasible.

We focus our research on robots without communication capabilities, with a limited set of error-prone sensors and little to no information about the surrounding environment at the starting point, as is the case for typical cleaning and lawn-mowing robots. We assume the use of landmark-based navigation and robots equipped with rfid readers as a concrete scenario.

The first part of thesis designs and presents a detailed knowledge architecture for intelligent robots able to use RFID tags both as landmarks and communication channels. This architecture contains a rule system for robots, providing reactive control while a robot is in action. The system has been successfully implemented during the the Roboswarm EU FP6 project for running the physical robot swarm.

Then we develop and compare different coverage algorithms for swarms of robots tasked with cleaning, search or similar activities inside buildings. The main goals are to find more efficient algorithms and to understand the improvements gained by increasing the swarm size.

The robot parameters and capabilities are modeled on the actual rfid-equipped Roomba cleaning robots developed during the Roboswarm EU FP6 project with the participation of the author. We have implemented a custom simulator of the Roomba robot, which takes into account the capabilities and operation times of the robot. Controlled simulations of this robot have been used for testing and comparing the algorithms.

The main part of our research consists in developing and investigating four main approaches for the robot control algorithm, all based on the idea of using recognizable locations (tags) to guide the robot around the mission area. As a background baseline we also consider “ideal” behaviour with the best possible performance of the swarm.

We show that a specific parameter of robot behaviour - the default turning angle - makes a significant difference for the performance of the investigated algorithms.

We show that the algorithms employing landmarks are almost consistently better than the parameter-optimized random algorithm and are, on the average, close enough to the ideal behaviour to be considered as practically sufficient

Most importantly, we show that as the swarm size and density increases, the performance improvements gained by better algorithms and more knowledge decrease quickly: in the other words, the size of the swarm trumps sensors and intelligent behaviour.

ACKNOWLEDGEMENTS

First I would like to sincerely thank Prof. Tanel Tammet for his guidance and motivation during all these years. Without his support the thesis would have not reached the finish line.

The thesis got its kickoff from the middleware development for the Roboswarm project of the EU's 6th Framework Program (FP6).

Additionally, I want to thank people who have contributed their time and knowledge: without them the work presented in the thesis could not have been completed. Special appreciation to Enar Reilent, who has supported with testing and developing communications for the real robots and all what concerns C programming; Madis Puju for developing the in-memory database for the Roboswarm project; Tanel Tammet for reviewing and brainstorming the solutions; Prof Jüri Vain for overall and formal guidance.

LIST OF PUBLICATIONS

The work presented in this thesis is based on the following publications:

A T. Tammet, J. Vain, A. Puusepp, E. Reilent, A. Kuusik. RFID-based communications for a self-organizing robot swarm. In: Proceedings Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2008: 20-24 October 2008, Venice, Italy: (Toim.) Brueckner, Sven; Robertson, Paul; Bellur, Umesh. Los Alamitos, Calif.: IEEE Computer Society, 2008, 45 – 54.

B T. Tammet, E. Reilent, M.Puju, A. Puusepp, A. Kuusik, A. Knowledge centric architecture for a robot swarm. In: 7th IFAC Symposium on Intelligent Autonomous Vehicles (2010). IFAC-PapersOnLine, 2010, (Intelligent Autonomous Vehicles; 7/1). 2010.

C A. Puusepp, T. Tammet, M. Puju, E. Reilent. Robot movement strategies in the environment enriched with RFID tags. 16th International Conference on System Theory, Control and Computing, Sinaia, Romania, 12-14 October 2012.

D Puusepp, A.; Tammet, T.; Reilent, E. (2014). Covering an Unknown Area with an RFID-Enabled Robot Swarm. Applied Mechanics and Materials, 490-491, 1157 – 1162.

AUTHOR'S CONTRIBUTION TO THE PUBLICATIONS

A [Contributing to the development of the robots inner rule language. Contributing to the creation of data protocol formats, including data encoding on RFID. Co-writing the paper.]

B [Further development of the rule language. Contributions to the memory datastore development.]

C [Developing the custom simulation environment. Introducing Roomba protocol for the simulation communication. Developing the coverage algorithms. Conducting and comparing the test experiments with the simulated robots.]

D [Improving the coverage algorithms previously developed. Conducting and comparing a larger amount, different and more thorough test experiments with the simulator.]

List of Figures

List of tables

INTRODUCTION

The area of this work is swarm robotics: making a set of separate robots work together as efficiently as possible. While the spectrum of complexities and capabilities of robots is potentially very wide, we focus on relatively simple robots with limited sensors, suitable for simple indoor activities like floor cleaning and searching objects.

This thesis starts with a detailed overview of the knowledge architecture of actual robots developed during the Roboswarm EU FP6 project with the participation of the author. After that it concentrates on developing various coverage algorithms employable by swarms of these or any other robots with a limited set of error-prone sensors and little or no previous information about the environment.

Background: robots and landmarks

The background of this research is an actual robot swarm developed during the Roboswarm EU FP6 project. The solutions reached at during our work generalize to swarms of robots with similar capabilities.

We assume the use of simple robots with very limited sensory capabilities and no communication or central coordination. We use the dynamic cleaning problem [1], [2] as a testbed for our algorithms. In general, such tasks can vary from cleaning [3], mowing [4], tour guidance [5] and rescue [6] to complex surveillance [7] assignments and even providing butler services [8]. There exist plans to use exploration swarms to reach to outer space. NASA is developing a robot swarm for mining [9].

There exist several approaches for indoor navigation like using a laser range finder or magnetometer [10]. It is also possible to guide robots using landmark-like radio beacons (e.g. [11]) or less expensive passive landmarks. Roomba family's flagship, the Roomba 980 released 17.09.2015, utilizes the onboard camera and image recognition to create visual landmarks for itself and cover the area using the vSLAM algorithm [12].

For guidance in the working area – assumed to be indoor environment - we use RFID tags marking objects, doors and covering locations which need to be visited. The same RFID chips can be used to leave messages to other robots, inspired from ants' communication using pheromone trace known as stigmergy [13]. The usage of RFID tags in such a way can reduce the communication overhead related with coordination [14].

Figure 1 Real Roombas equipped with 3 RFID readers each, developed during the Roboswarm project.

An iRobot Roomba cleaning robot is used together with a small ARM-based Gumstix computer (500MIPS) running on BusyBox 2.6 Linux distribution and a stock RFID reader/writer.

Our robots use RFID tag sequences and potentially other similar landmarks to detect their location and direction. While moving around the mission area the robots can take advantage of the tags and adjust their behavior to perform better compared to the environments without RFIDs. Tags can be placed on either on the walls [18], [19] or on the floor [20] and positioned as a regular grid [21] or just randomly. Tags are serving mostly as a navigational graph for robots which are driving from tag to tag trying to reach their destination [22].

Communication between the robots can be implemented through messages on RFID[1] chips planted across the working area or via a central server. In our experiments we use the central server only for the simulation and analysis purposes. Communication between the robots is implemented through messages on RFID[2] chips planted across the working area or via a central server. Otherwise a central server is used only for the simulation and analysis purposes.

[1] Radio-frequency identification

[2] Radio-frequency identification

Motivation and problem statement

We want to decrease the time it takes for a swarm of robots to either clean a set of interconnected rooms or to find a tagged object located in some room.

The knowledge engineering goal of the thesis is designing a suitable knowledge architecture for all layers of the robot swarm – tags, robots, server, rules and communication - and validating its usability in a real indoor swarm.

The main research goal of the thesis is to find robust and efficient algorithms and principles for practical area coverage in a swarm of indoor robots relying mainly on navigational landmarks, with limited and inaccurate sensors. We also want to compare the algorithms in different room complexity scenarios and analyse the effects of the swarm size on the coverage time.

The common approach to the problem of coverage in literature is dividing the map into cells [15], [16] and path planning [17]. We consider only algorithms which do not take advantage of these methods: the reason being the lack of orientation and location info of the indoor-moving robot, mainly because of the accumulating error of odometry and tag detection inaccuracy of simple robots. Notice that obvious solutions for outdoors navigation - GPS and compass - cannot be reliably used indoors.

Contribution of the thesis

The thesis gives two different contributions in robotics: a knowledge engineering contribution and a swarm robotics research contribution.

The knowledge engineering oriented contribution of the thesis consists in designing the knowledge representation, reasoning and communication principles and software of the RFID-equipped intelligent cleaning robot on top of the stock iRobot Roomba cleaning robot, developed by a team with the participation of the author. The large set of experiments performed by the real swarm of robots with this knowledge architecture proved that the designed principles and representations are feasible in practice.

The research contribution of the thesis is a design and comparative investigation of different simple and robust coverage algorithms for non-communicating swarms of realistic cleaning and lawn mowing robots with limited and error-prone sensors in the environment enriched with landmarks detectable at close range. We note that we have measured the behaviour of the algorithms only on a simulated swarm, not a physical swarm.

Most of the previous work in coverage algorithms has been conducted with different assumptions, like exact sensors and a thorough knowledge of the environment.

We show that a specific parameter of robot behaviour - the default turning angle - makes a significant difference for the performance of the investigated algorithms.

We show that the algorithms employing landmarks are almost consistently better than the parameter-optimized random algorithm and are, on the average, close enough to the ideal behaviour to be considered as practically sufficient. As expected, the differences between the performance of the algorithms are more significant in case the environment consists of several rooms or areas with narrow, hard-to find doors or connections. In particular, the capability to recognize doors has been shown to be a key aspect for multi-room areas.

Most importantly, we show that as the swarm size and density increases, the performance improvements gained by better algorithms and more knowledge decrease quickly: in the other words, the size of the swarm trumps sensors and intelligent behaviour.

Thesis organization

Thesis starts with the general introduction and overview of the relevant related work in the area of swarm robotics.

Next we give a detailed overview of the entire robot platform: how the processes work inside the robot. The particular focus is on knowledge representation and usage for the robot and the swarm. The author was a part of the team designing the algorithms and the data representation for the robot. These chapters also constitute an introduction to the main research contribution of the author given in the following chapters.

The chapters “Area coverage with a swarm” and “Experimental results” contain the main research contribution of the author. First we introduce the simulator and the background details of all the algorithms that have been developed and tested for the robot. Then we focus on actual tests with the simulator. Finally the test results are reviewed and explained and main conclusions drawn.

1 RELATED WORK

This chapter gives an overview of the work related to the main contribution of the paper: robust coverage algorithms for a swarm of robots.

The problem of robot navigation and path planning is a classic question of robotics with a long history.

The widely used practices in literature for robot coverage problem involve dividing the area into the cells [23] [24] and path planning [25].

The spanning tree covering algorithm [26] works by dividing an area into cells and creating a spanning tree. There are two main approaches: off-line and on-line. In the off-line approach we assume full knowledge of the entire area and create the entire spanning tree at the start. The on-line approach uses sensors to determinate the surrounding cells and creates the spanning tree incrementally. In both cases the robot has position and orientation sensors which our robot lacks.

The path planning algorithms [27] also divide an area into cells; then they try to move to cells which are considered to be obstacle free. Being in a cell means positioning the robot in the center of the cell. Moving between the cells means moving from one center point to another center point. This kind of navigation requires precise positioning and knowledge of the robot orientation.

Our work, however, focuses on scenarios with little or no previous knowledge about the environment and simple robots with limited and error-prone sensors where the classic questions of path planning are not applicable. We analyze the behaviour of swarms of robots and the effect of swarm size to the efficiency of solving the task. Hence we will concentrate on the related work with platforms, goals or assumed limitations similar to ours.

1.1. Robot platforms

First we have a look at the developments in related robot platforms.

To start with, the main platform used in our thesis – iRobot Roomba extended with rfid readers during the Roboswarm project – has seen similar developments from the iRobot company itself. As reported in [28], iRobot has developed the Roomba 980 model equipped with a wifi and a camera along with intelligent navigation capability based on VSLAM (Vision Simultaneous Localization and Mapping). The robot provides also remote control with a smartphone app. VSLAM is a way of dynamically building a map while keeping track of your own position at the same time. To recognize places, the camera takes a picture, and then looks for distinctive patterns of pixels in that picture. The Roomba 980 also performs what is known as “sensor fusion,” meaning it combines data from various proximity sensors with imagery from its camera. The robot will then remember what the seen features look like and will keep track of them as it moves.

The paper [29] presents technology and experiments converting a large 200kg non-autonomous floor scrubber into an autonomous one. The robot uses LIDAR, camera and UWB for navigation. The authors say that Lidar+AMCL is a good choice as primary localization tool for the robot module. However, they recognize that during normal operation, automatized robot scrubber will operate in open spaces (e.g. warehouses and/or squares) where maximum rage of current Lidar (6 m) will not be adequate and robot might not get sufficient data for such an accurate localization. Then UWB based localization should be tested for such environments.

The kilobot project [30] [31] [32] focuses on building a robot swarm with a size of up to 1000 robots, positioning themselves into preprogrammed 2D shapes. They use infrared reflection of the surface for communication and distance sensing between each other. Robots are very small, with a diameter of 33 mm and a height of 34 mm. The communication distance between two tiny robots can be up to 70 mm. Forming a single preprogrammed shape takes about 12 hours for the entire swarm. Kilobots do not have any location sensors and are similar to our robots in the sense that they do not use path planning. Kilobot is a popular and a relatively cheap way to perform real world tests with a relatively large robot swarm.

Mobile Agricultural Robot Swarms (MARS) [33] have a set of objectives similar to our thesis: use robots with a minimal set of sensors to achieve low cost and energy efficiency. Differently from our swarm, the members of MARS swarm are centrally guided and each member has precise knowledge of its location.

1.2. Robot swarm behaviour

Next we will take a look at interesting papers concentrating specifically on the robot swarm behaviour, navigation and coverage strategies suitable for robot swarms. We have found no papers dealing with the assumptions and goals exactly like ours, although several papers focus on very similar tasks and questions, with somewhat different assumptions on robots/environment. We will start with the older papers and end up with the newest ones.

The paper [34] develops a swarming navigation algorithm in order to find the odor sources in an unknown environment, based on the ability of each swarm member to sense the odor. This task is similar to the task presented in the current thesis. Each robot in the swarm has a cooperative localization system which uses wireless network as a mean of measuring the distance from the other robots. At least three robots act as stationary measurement beacons while the other robots of the swarm navigate in the environment towards the odor source. The novel approach of the paper is the usage of the wireless network to estimate the distances. Our paper assumes simpler robots without such capabilities.

The paper [35] applies automated probabilistic formal verification techniques to robot swarms, in order to assess whether swarms will indeed behave as required. The example presented in the paper is a foraging robot scenario, which is similar

to the task handled in the current thesis, although the methods applied are widely different.

The paper [36] proposes a novel motion control method: magnitude-dependent motion control (MDMC). Similarly to the current thesis the authors focus on simple robots that lack the capability to detect the orientation of their neighbors. However, the task of the robots in the paper – flocking together by keeping a certain distance from its neighbors - is very different from the task we consider in the thesis.

The paper [37] proposes a sweep coverage formulation for a multi-agent system to cover a region with uncertain workload density, and provides a decentralized coverage algorithm based on the formulation. To achieve the coverage, the covered region is divided into a finite number of stripes, and an algorithm is proposed by incorporating two operations on stripes: workload partition and sweeping. The paper presents a theoretical analysis of the upper bound of coverage time spent more than the optimal time. In our thesis we gain a similar comparative benchmark by simulating a near-ideal coverage run.

The paper [38] presents a distributed control strategy, enabling agents to converge onto and travel along a consensually selected curve among a class of closed planar curves. Individual agents identify the number of neighbors within a finite circular sensing range and obtain information from their neighbors through local communication. Again, the work differs from our thesis by having a different goal: while in our work it is advantageous for robots to spread out, the focus of the paper is to follow a common path.

The paper [39] presents and investigates Darwinian Particle Swarm Optimization (DPSO): an evolutionary algorithm using natural selection to enhance the ability to escape from local optima. The goal of the paper is similar to one of the assumptions or subgoals of our thesis: decreasing the amount of required information exchange among robots. The paper presents a stability analysis of the RDPSO.

The paper [40] investigates a swarm of robots with similar capabilities to ours and with similar fundamental problems: inexact odometry, both in the sense of the travel distance and turning angle. However, while one of our main methods is to spread the robots, the task considered in the paper is the opposite: gathering robots together.

The paper [41] investigates coordination principles inspired by the behaviour of honeybees and ants for coordination purposes in multi-robot systems. While the swarm robotics approach with limited resources is similar to ours, the paper does not give concrete simulation or real-life experiments, but rather proposes possible approaches using stigmergy, somewhat similar to the approaches used in our thesis.

In the paper [42] the authors generalize the control law based on minimization of the coverage functional to such non-Euclidean spaces punctured by obstacles.

They propose a practical discrete implementation based on standard graph search-based algorithms and demonstrate the applicability of the proposed algorithm by solving efficient coverage problems on a sphere and a torus with obstacles, and exploration problems in non-convex indoor environments. Concretely, they consider exploration and coverage of an office environment by a team of four robots. An important focus is flexibility of the framework with respect to incorporating human inputs to guide exploration. No comparisons with other approaches or investigations of swarms with different sizes are presented. Differently from the assumptions in our thesis, the robots are equipped with onboard range sensors and can localize themselves in a global coordinate frame.

The paper [43] investigates decision-making strategy to solve the best-of-n decision problem in a swarm of robots. This problem requires the swarm to establish a collective agreement on the highest-valued option among a finite set of alternatives. A certain similarity to our thesis could be seen in a question of which marker to reach or which room to enter next. However, in the case of our thesis the main complexity lies in actual navigation, not so much in choosing the next target.

The papers [44] and [45] analyze methods for patrolling and surveillance in an environment with a distributed swarm of robots with limited capabilities. The focus is on structured exploration of unknown spaces with multi-robot systems, using triangulation that is constructed in a distributed fashion and guarantees good local navigation properties. Similarly to our thesis, the sensors and robots have very limited capabilities. However, the papers assume the use of large set of infra-red beacons and a beacon sensing capability from the robots, somewhat similar to our rfid tags, but a certain ability to sense distance and direction, differently from ours. The authors of the papers claim that experimental results with real robots are very similar to the results obtained by simulation. It is interesting to note that the effects of increasing the swarm size indicated in the papers are similar to our results.

The paper [46] considers a complicated superset of the tasks considered in our paper: creating an integrated 3D-view of the environment using camera-equipped robots. Irobots equipped with a camera, Kinect and a Raspberry Pi are used as a test platform.

The paper [47] presents a neural dynamics for complete area coverage navigation by multiple robots. A bioinspired neural network is designed to model the workspace and guide a swarm of robots for the coverage mission. The same platform as used in the thesis – Irobot Roomba – is used for simulation and testing, with a complex added sensor system including ultrasonic sensors and a Microsoft Kinect sensor. Unexpectedly, the authors report that two robots cover the area significantly faster than a single robot; no extensive experiments with multi-room environments or a large swarm are reported.

2 ROBOT PLATFORM

The current chapter presents an overview of the robot platform. The author was a part of the team designing and programming the robot and contributed significantly to the knowledge representation and communication tasks. The main pure research contribution of the author will be presented in the following chapters “Area coverage with a swarm” and “Testing results”.

The results presented in the current chapter have been published in the papers A and B in the appendix.

The concept of the robot architecture has stayed the same through all the published articles: it is based on the layered multi-agent system. Agents are implemented as continuously running processes. The entire platform is divided into three layers:

- The sensor-actuator layer responsible for communication robots control hardware.
- The control layer is a constantly running dispatcher process responsible for executing behavioral tasks. Behaviors in our context are small binaries fulfilling several smaller tasks.
- The knowledge layer conducts reasoning – derives new information from gathered data also communicating with other robots via RFID tags (passive communication) or optionally via central server (active communication).

The centre of the architecture is a shared memory RDF-inspired database. Similar approaches can be found in low latency robot control architectures performing well without a real time operating system. Agent communication is implemented by using the memory database. Each agent can access all the data inserted to the database. This kind of approach is known as a classical blackboard model [48].

The in-memory database serves three basic purposes:

- A postbox between different process agents including external world communication.
- A fast and relatively simple in memory database implemented as circular buffer.
- A deductive database, generating new facts based on the rule language.

2.1. Memory database

The entire robot platform runs on the Gumstix computer which is planted to the Roomba robot. The memory database can be used by all agents running on a robot. The database itself is not a process: it is implemented as a C library which can be used through the public API, providing read, write and search functionality. Data is read and stored in the shared memory.

Data model consists of one single public table with an RDFm structure. Inside the database there is an additional table for storing unique strings, pointed to from

the public table. Strings in the memory database are immutable. Whenever a string is changed in the main table, a new item is created to the string table and pointed to it. However, if such a row already exists, then the pointer will be addressed to already existing value.

All rows are organized as a circular list. The last element will be removed from the list if the size limit is reached. Exceptions are made for the critical data items – so called flagged rows – which are kept in the database until they are released by the agent.

The locking system for the memory database is row based and implemented by using semaphores. When a row is being written it is hidden for all the concurrent threads. Reading and search functionality does not lock anything.

The hardware setup with the Gumstix computer takes about 0.14 milliseconds to write one row. Looping over 2000 rows needs approximately 4.8 milliseconds of time: these performance numbers are acceptable for our needs.

The memory database is used as a postbox between different agents running inside the robot. Rows created by one agent can be addressed to another by using name as addressee. During runtime an agent will look for rows with its name, process them and then remove or reassign them.

2.2. Data model and languages

The common data model for communication between different counterparties is based on a RDF triplets. The current architecture has extended RDF triplets with metadata fields which we call RDFm.

The common data model and the data model in RFID tags has been published in the paper A in the appendix.

The main parts which have most effect to the robot behavior can be divided to the following groups:

- Sensors and control software.
- Internal memory database contents.
- RFID tags read.
- Binary executables together with data / rule files from server.

The server collects data from all the robots and affects their behavior by updating the knowledge of robots. Data sent back to the robots can consist of some new informational units to improve the behavior or uploading new binaries and rule files to modify the objective. Robots can acquire new instructions or information by reading encountered RFID tags during their movement. Data from sensors generates yet another set of information for the control software. All these layers must present their data in a unified way to understand each other, resulting in a use of the following set of languages for the robot platform:

- RDFm encoding in RFID tags.

- Our specialized rule language for deriving new information based on the data in memory database.
- Both CSV-based syntax and an XML-based RDF syntax for data exchange between robots and the central server or external systems.

While choosing the languages we had to take into account the different key factors of the mediums: mostly storage space and communication speed. For example, RFID chips contain very little memory, which requires space efficient encoding. Also, the transfer rate cannot be compared with WIFI. On the other hand, communication between the servers does not necessarily require space efficiency, resulting in the use of self-descriptive XML-based standards for better understanding.

The RDFm data model itself is inspired by RDF triplets which we have extended resulting in so-called RDFm consisting of three different groups of data fields:

- RDF triple data fields
- Contextual metadata fields
- Automatically generated fields

Standard RDF triples have the following data fields:

- Subject: id of whatever has the property.
- Property: name of the property of the subject.
- Value: value of the property

The value field has an associated type, indicating the way to interpret the value. Observe that the property field may determinate the suitable or expected type, but this not always the case.

The second set of fields in the RDFm data model contains contextual metadata fields:

- Date/time: when this fact held: in the most cases same as the time of storing the field data.
- Source: identifies the origin of the data: RFID code, person id who added the data, other robot id, agent name, etc.
- Context: in most cases identifies the addressee or data group, but can also indicate the succession of robot commands.

Program units aka agents can enter their own contextual values to the memory database. Otherwise the default values - current time, current robot id, empty context - are used.

The third set of fields in the RDFm data model contains metadata fields which are generated automatically by the memory database during data insertion:

- Id: unique number for data row.
- Timestamp: date / time of the storage.

Automatically generated fields are the strictest set of fields: they cannot be manipulated by the agents and are not available in the other data languages. Their purpose is to guarantee efficient and convenient data management and they are used by the reasoner and dispatcher processes.

Instead of using contextual and metadata fields we could have created the data model using standard RDF triples to store the same information. However, this would have been memory consuming and inefficient, especially for the RFID chips, but also for the memory database.

2.3. Data format in RFID tags

Our setup uses RFID tags for external object/location recognition, asynchronous messaging between robots and for location specific messages/instructions from humans. Tags can be divided into two main groups – cheaper ones with only an id value on the chip and more expensive tags with a small internal memory.

While analysing the coverage task, simple tags with only the id values on the chip can be used as beacons for detecting visited points. The tags with internal memory can be used for marking objects, giving warnings about dangers on the field or instructions where to drive next. Such tags behave like information carrying graffiti distributed all around the working environment. Both human operators and robots are allowed to write information to the tags with internal memory.

Different sources tend to write different types of data. Humans will usually provide static or passive types of information, while robots store often-changing data. Within the environment setup phase a human users are expected to write information for:

- coordinating the robots – “this tag has coordinates X and Y”
- giving idea about surroundings – “this tag is located on a chair”
- guiding the robot – “there is a tag at direction R at 5 meters”
- warning about dangers – “keep away from here”

The robots are expected to write information about the current situation on the field:

- Informing what has happened in current location, like “robot N brushed here for 10 minutes on 10.06.2007 at 15:10”.
- Leaving information about its further plans, like “robot N left this place and moved towards the living room”

Data on the RFID tags must be compact and easily understandable for all the interested counterparties. External applications, robot software and agents outside the roboswarm must have a unified understanding of the encoding to read and write data to tags. Examples provided beforehand are human readable representations of the RFID contents and are not in any way memory efficient. All data units written to the tags are essentially data rows with several predefined fields which can be of a type string, integer or float. The RFID database resembles the memory database used inside the robot computer – older values are pushed

out by the newer items. Rows with the context value of "static" on the RFID memory database are considered as flagged items and are preserved during removal process. Data from the tag is read, written or deleted one row at a time. Updating is allowed only for specific fields – on the value field, the timestamp field and the source field.

Table 2.1 Graphical representation of the RFID memory

subject	property	Value	source	context
me	inRoom	Kitchen	human	static
kitchen	hasPriority	7	human	static
kitchen	dutyStatus	cleaingInProgess	robot3	work
robot2	wentInDirection	270	robot2	work

Data rows with the static context are inserted by humans during the setup of the environment. Messages with the "work" context value are written during problem solving by the swarm members to improve the fulfillment of the tasks at hand.

Similarly to the robot memory, a global table for strings is kept separately and only the codes are used in place of the strings. Encoding the values is performed by using the string table on a robots memory database. During startup the global string information is propagated throughout the roboswarm, providing same data to each robot to have common understanding of the encoded values. This approach divides data field contents into the direct/primitive values like integers, RFID chip ids or indirect values like strings. Direct values are being written to the tag as-is. Long string values are replaced with the corresponding code from the global string table. We are allocating 2 bytes of memory for the string table identifier.

RFID tags always have a built-in unique id. The size of the id may vary, depending of the manufacturer and chip type. However there are number of widely known standards for id encoding and majority of the tags are expected to follow these standards. Our solution uses direct 96-bit EPC[3]'s for identifying the RFIDs. It is very common for the tag to know information about the location or the object that it is being glued on. Keeping in mind the small amount of memory available we will be using value "me" to refer itself in the data row instead of the real EPC value.

2.4. Rule based control system

The robot control in our architecture has been divided between several agents getting the instructions from the memory database. The entire control mechanism can be divided into two major parts – the main framework with built in agents and user defined applications which can be executed via the rules. All data – sensor readings, decisions, reports – will be available to each party via the memory database.

[3] An Electronic Product Code (EPC) is a universal identifier that gives a unique identity to a specific physical object.

The rule based control system has been built around the memory database and is tightly coupled with the main data processing mechanism: the prover. Based on the existing user defined rule files and data located in the memory database, the prover will regularly derive new facts. Newly generated instructions will become inputs for the other agents and user defined applications controlling the robot.

Based on the process lifecycle the components can be divided into two groups – ones that work as an endless loop and others that are being executed only on demand. The prover together with the memory database, the communication process, the low level hardware access software (sensor agents, actuator process) and the dispatcher run all the time. The built-in and user-defined applications which do not have to be running continuously, are executed via the dispatcher process.

Problem solving algorithms can be divided into modules and rule sets. Reasonable balance should be kept between the logic implanted into the modules and rule sets size. Trying to create simpler rule files, the series of low level commands are gathered together into modules which are meant to perform atomic tasks. For example, a simple module (binary executable) can play a sound, calculate an average or achieve a complex goal, like performing a localization procedure when encountering an RFID tag.

Rules have the role of linking together binary sequences and making decisions during the runtime. For example: agent A stores the fact B into the memory database. The prover derives (according to the current rule set) the new fact C from the fact B, where C is a command for dispatcher to start the agent D. When the dispatcher process sees the newly derived fact C in the memory database, it executes the demanded agent D binary, which in turn can change the contents of the database.

The rule based control system and the rule language has been published in the paper B in the appendix.

2.5. Rule engine and rule language

The entire robot system is controlled via the memory database where all the obtained facts, derived facts and commands to execute are inserted. Rules are written in a prolog-like syntax and stored in the local file system. Rule files can be stored to the specific robot manually or propagated throughout the swarm via the central server. New facts are generated by the rule engine integrated into the robot architecture as a core process. The rule engine work cycle starts with reading the rule file from the file system. The second step is applying the rules to the up-to-date facts in the memory database and inserting newly derived facts into the memory database. The rule engine cannot be used directly by any agent in the robot architecture: entire communication is performed by inserting facts to the memory database and reading the output afterwards.

In other words, the goal of the rule engine is not to answer queries, but to automatically derive new facts based on the data inserted to the memory database by other processes. The rules set must be consistent and should not contain too many or too complex rules. As a rule engine we are using a special modification of the Gandalf [49] first order resolution-based theorem prover. The rule engine process is being executed automatically after each pre-determined interval of time, typically one second. We call the execution of the prover and one derivation cycle a "derivation session". Using a relatively simple set of rules we are able to keep the derivation session duration under one second: during this time the rule engine will load the rule set from file system and perform all possible derivations stemming from the facts added to the memory database after the last run.

The rule system has two main goals:

- Derive generalizations (like chair is furniture) from rules
- Derive commands depending of the situation

For example, if we have a rule

attachedTo (X, furniture) :- attachedTo (X, chair).

and the memory database contains the facts shown in Table 2.2 Initial data in memory database

Table 2.2 Initial data in memory database

subject	property	value	source	context
tag4	attachedTo	chair	RFID	null

then the rule body *attachedTo (X,chair)* will match the row in the memory database and during the derivation session rule engine will generate the new fact and insert it to the memory database as shown in Table 2.3 Fact generated by the derivation process

Table 2.3 Fact generated by the derivation process

subject	property	value	source	context
tag4	attachedTo	furniture	wGandalf	null

All the words in the rules starting with the uppercase letter are variables. In our example here X is a variable.

The following example demonstrates a simple session of rule set usage.

handleTask (me, Task) :-

state (me, stateIdle), receivedTask (N, Task), myNameIs (me, N).

state (me, stateWorking) :- handleTask (me, T).

startMode (me, cleaningMode) :- handleTask (me, clean).

startMode (me, patrollingMode) :- handleTask (me, patrol).

state (me, stateIdle) :- state (me, stateWorking), status (currentTask, finished).

Table 2.4 The rule system startup dataset in memory database

subject	property	value	source	Context
me	state	stateIdle	init	wGandalf
me	myNameIs	robot3	init	wGandalf
robot3	receivedTask	clean	init	wGandalf

Based on the rule set the engine will automatically derive and add the facts in Table 2.5 Facts generated based on the startup dataset into the memory database during the derivation session.

Table 2.5 Facts generated based on the startup dataset

subject	property	value	source	context
me	handleTask	clean	wGandalf	null
me	startMode	cleaningMode	wGandalf	null

When a cleaning action is finished and the process adds the fact described in Table 2.6 Fact generated after finishing cleaning process to the database.

Table 2.6 Fact generated after finishing cleaning process

subject	property	value	source	context
currentTask	status	finished	cleaningAgent	wGandalf

The rule system will derive and insert the fact described in Table 2.7 Fact generated during derivation session after cleaning is finished into the memory database during the next derivation session.

Table 2.7 Fact generated during derivation session after cleaning is finished

subject	property	value	source	context
me	state	stateIdle	wGandalf	null

The rule engine uses both the publicly available main memory database and a temporary storage. Temporary storage holds non-final facts and is being cleaned up after each derivation session. During the derivation process a large set of new facts and clauses (temporary rules) are derived. Final facts without variables (ground unit clauses), not containing nested terms and having a suitable number of arguments are stored in the shared memory database and are available to all other processes in the robot.

A derivation session starts with reading and parsing the rule file from the local file system and adding parsed rules and facts into the temporary space. Continuous re-loading makes it possible to update the contents of the rule file on the fly. We employ the widely used discrimination tree index for the unit subsumption and unit deletion. Only the temporary area, not the facts in the shared memory database are kept in the index. The engine uses a version of a set-of-support binary resolution with common optimizations like subsumption and tautology elimination. See Robinson and Voronkov (2001) [50] for the common algorithms employed in first-order automated reasoners.

Re-derivation of facts which have been already derived in the last session has to be avoided, otherwise the reasoner would produce facts causing the robot to do the same things repeatedly. Hence we have developed a timestamp-oriented

special version of the set of support algorithm for rule engine, which avoids redoing the same derivations in the next session. This has an added effect of keeping the amount of derived facts during one derivation session down even for relatively large rule sets.

2.6. Behaviors and rule language

Behaviors are a collection of commands gathered into one group which can be executed with a single command from the rules side. Each behavior is implemented as a small binary program written in C language. It is built from atomic commands or calls to other binaries in order to perform complex operation. During implementation of the binary we should remember that several instances of binaries could run at same time.

For example, let us have the following rule set:

behavior (me, "monitorObstacles") :- state (me, stateInitial).

behavior (me, "goAhead 200") :-

state (me, stateCanMove), obstacle (me, nothing).

behavior (me, "handleFailState") :-

result (solveObstacle, fail), state (me, stateDriveAround).

Where

- behavior is a special name, indicating that the fact is the command to launch the given binary.
- monitorObstacles is a binary program for monitoring whether any obstacles are getting in the robots path. If there should be an obstacle in front of the robot, the obstacle (me, front) fact will be added into the memory database.
- goAhead is a binary program that makes the robot to move forward or backward with the speed stated as an argument to the command. For the current example the translational velocity is 200 mm/s and angular velocity is 0.
- handleFailState is a binary program that is executed in critical situations: it will stop robot movement and sensors to save power and will communicate the information of the failure situation to other robots or to the central server.
- solveObstacle – a binary program that tries to find a way to get past the obstacle that has gotten in way of the robot.

Binary execution is being handled by the process we call dispatcher. In order to execute a binary at the desired time the proper command has to be inserted into the memory database.

Table 2.8 Dispatcher execution command fact describes an example of the command row which makes the dispatcher to execute a binary.

Table 2.8 Dispatcher execution command fact

subject	property	value	source	context
me	behavior	command	wGandalf	dispatcher

where command is a string "behaviorName arg1 … argN".

Timing will become critical when implementing the robot control application on top of the prover and a relatively large set of behaviors. The time elapsed between the point of giving the command and the actual execution varies greatly depending of the current contents of the memory database, the size and complexity of the rule file, the number of processes running in the system, the length of the reaction chain and other factors. However, in our test cases the response times have proved to be acceptable.

Based on our testing, let us consider the following example where by cooperation between the prover, the dispatcher and two behaviors the robot has to avoid colliding into the obstacle. Typically it takes about 400ms from the moment when one behavior (currently monitorObstacles) discovers an obstacle to the point where the prover inserts a command into the memory database to launch another behavior. After about 20ms has passed the dispatcher has received the command and is ready to start the given behavior. After additional 100ms the second behavior (solveObstacle) takes over the robot control and tries to maneuver the robot past the obstacle. High-level decision making can safely rely on the given architectural scheme. However life-critical emergency responses like avoiding robot falling down the stairs after the cliff sensor detects danger should be implemented into the hardware or handled by the low-level software agents.

2.7. Robot communication with the server

Robots having the wireless network capability can use the robot-server centralized communication and robot-robot ad hoc communication when no WIFI access-points are available. A single transfer session consists of two steps: first the robot sends its data to the server and then downloads new information addressed to it from the server. Communication is being handled by the separate process on the robot: it sends all the new data items from the memory database to the server. Robots have been configured to execute data transfer with a one second interval. On the server side the data received from each robot is stored inside the postgresql [51] database for further processing. For example, suppose data received from robot4 contains information addressed to the robot9. When the robot9 initiates data transfer, the server has to gather all the facts for robot9, including the ones arrived from robot4 and send them out.

The server replies to each uploading act with the dataset gathered for this particular robot, accumulated since the last transfer session. Software agents on the server side cannot directly send any data or commands to robots: everything is communicated via the postgresql database. The special communication agent

handles the data transfer between the server and robot. Items received from the server will be stored directly to the robot memory database.

Humans can monitor and send commands to the swarm or a single robot through the dedicated user interface built on the server. Data will flow through several processes on the server side and will finally be transferred to the robots via the communication agent.

The server has an additional swarm coordinating role used in some applications. For example, the robot swarm can be used to find a certain object in an environment. After a user gives the task, the server chooses the suitable group of robots in a swarm and communicates the id of RFID to the selected robots. Fulfillment of the task begins, robots spread out in the environment and start looking for the specified tag. As soon as the necessary RFID is being found the robot who discovered it, reports to the server. The task is now considered completed, the user is being notified and all the other robots selected for the job are notified that tag has been found and told to stop.

Communication between the robot and the server is being handled using the CSV[4] data format and sent via the http POST. First row of the data bundle is the robot id where the data is sent from. All the following rows are the CSV representation of the memory database contents. The CSV protocol is used for both directions: sending robot data from to the server postgresql database and vice versa, receiving data from the server side.

2.8. Summary

In this chapter we have presented a high level overview of the architecture and the design of the entire system and its components. All the components communicate via the in-memory database described in the section 2.1 “Memory database”. Data structures used for the robot control and guidance are described in the sections 2.2 “Data model and languages” and 2.3 “Data format in RFID tags”. Moving the robot and making it to perform tasks is achieved using the commands derived from the rules. These are described in a rule language and enable the rule engine to infer the specific commands and facts. The rules and the rule language are explained in the sections 2.4 “Rule based control system” and 2.5 “Rule engine and rule language”. We introduce small programs called behaviours, similar to the simple commands but performing more complex tasks: we present these in the section 2.6. “Behaviors and rule language”. Robots can communicate and share their knowledge via the central server which is explained in the section 2.7. “Robot communication with the server”.

[4] comma-separated values

3 AREA COVERAGE WITH A SWARM

This and the following chapter contain the main research contribution of the thesis.

Our task for the robot swarm was to move around the closed two-dimensional map and to cover it as quickly as possible. The algorithms are mostly introduced in the last two papers of the author (papers C and D in the appendix) with an addition of some elements in the current thesis. Task fulfillment is being measured by the number of location markers (tags) found; the goal is to find a certain percentage of all the location markers in the environment. We use the RFID tags as experimented with in the Roboswarm project. However, the same algorithms are applicable to other types of location markers: for example, locations recognized by an on-board camera as used in newer iRobot Roombas. Different map sizes and tag densities were used during the testing. Several algorithms were created, run and compared on the simulated robot.

Our robot cannot use path planning or divide the area into cells due to the very limited knowledge of its position and low odometry precision. Path planning would be hard to achieve due to the underlying robots low precision odometry which makes it hard to navigate to exact location at a longer distance. Dividing a map into small cells and moving between them is not possible due to the fact that we have no knowledge about the map layout where the swarm is operating.

However, orientation of the robot can be approximately determined in a situation when the robot finds two or more RFID tags during a straight line drive. From there, the position could be calculated from odometry values, but while the robot continues driving, precision is quickly lost.

Therefore, our algorithms do not require the knowledge of the robot orientation or the location of the robot. Some of our algorithms assume limited knowledge of the environment: the locations of tags. The assumption that we do not have a map of the area is useful in the situations where the room layouts change often, like a modern office, hospital environments, etc.

We have developed four separate algorithms taking advantage of different ways to find tags inside the environment and requiring various level of input data at the initialization point. In the following we will use the names “random”, “history based”, “map aware” and “extended map aware” for these four algorithms.

First, the “random” algorithm does not require any knowledge of the surrounding environment: the robots just drive around the map, bouncing against the walls, sometimes finding the tags. Importantly, the turning angle after bumping into an obstacle plays noticeable role in the performance and will be covered later.

The second, “history based” algorithm remembers encountered tag sequences and tries to use them to change its course when running into the same tag sequence afterwards. The initiative for this algorithm originates from the idea to find simple and low cost ways to improve the performance of the random walk: this turns out

to be a good starting point (as by [52]) in the cases where sophisticated solutions exerting FastSLAM (e.g. [53]) are not applicable due to the hardware constraints. The random algorithm moving principles are used as the base and are put into use when no help can be gained from the knowledge of travelling history.

Third, the “map aware” algorithm has a higher level of input data at initialization point: it knows the coordinates of all the tags located around the environment. The positions of walls, doors and obstacles are unknown. The main goal is to navigate to the nearest unvisited tag, which can be done only in occasions where the robot has seen two or more tags during a straight line move. Again, the default driving principle is based on the random algorithm and is enabled when no tags have been found and the robot does not know its orientation.

Fourth, the “extended map aware” algorithm knows everything what the regular map aware algorithm does and has additional knowledge and special handling for the door tags. These RFIDs are located exactly at the doorstep and have been specially marked.

The Table 3.1 Summary of data usage by each algorithm summarizes the data usage for algorithms.

Table 3.1 Summary of data usage by each algorithm

algorithm	locations of tags	remembers found tags	remembers found sequences
random		X	
history based		X	X
map aware	X	X	
map aware ext.	XX	X	

Note: the extended map aware algorithm can differentiate between tags on doors and all other tags.

The Figure 3.1 Inheritance of the properties of algorithms. illustrates the re-use of the properties of algorithms and the behaviors of more complex algorithms extending the simpler algorithms.

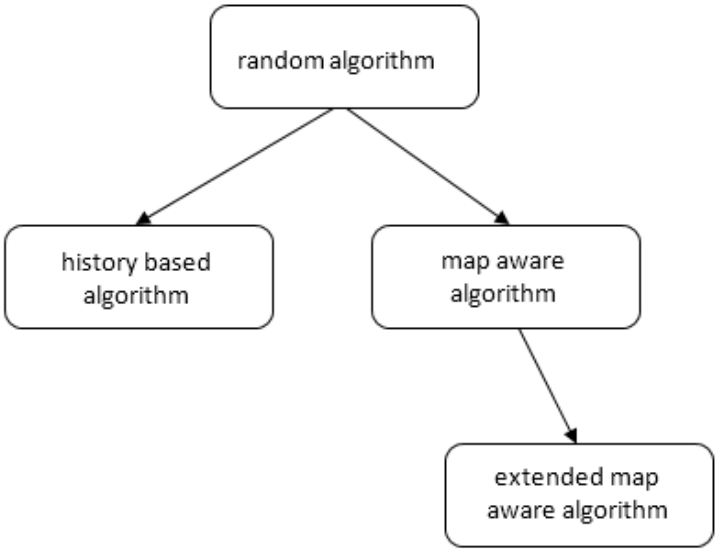

Figure 3.1 Inheritance of the properties of algorithms.

Each tag placed into the environment holds a minimal set of data needed for the robot navigation by the map-aware algorithms:

- x, y coordinates of the RFID tag
- RFID tag id, which has to be unique within the environment
- room id or if it is a doorstep tag, then the special value indicating it

3.1. The simulator

For architectural design and rule engine testing we have been experimenting with real RFID reader equipped Roomba robots [54]. However using real robots for testing and developing control algorithms is time consuming and can be heavily affected by external variables. Physical robots require a person to be located in the development center, batteries need to be recharged regularly and switching the environment would mean reorganizing the entire test area. For these reasons the decision was made to use simulations for control algorithm development. There are various simulators available, but due to the need for the exact and full control over the simulated robots we have developed a custom simulator. Our first simulator was developed using the Panda3D [55] gaming engine using Python programming language which eventually turned out to be too resource demanding. Running multiple instances of the simulator in one machine was almost impossible. Clearly there was a need for lighter solution: hence the simulator was rewritten to Java using the JGame 2D [56] gaming engine. In the same development machine ten or more instances of the simulator can be run without any performance problems.

Testing system setup consists of two applications. One is the graphical engine that visually moves the robots: an actual simulator. The second application is the so called algorithm runner where the logic is implemented. The latter one connects to the simulator using sockets and sends byte commands to the simulator which then starts to move robots accordingly. Commands are based on the instruction set from the Roomba robot manual. The reason for using original commands is to be able to connect the algorithm runner directly to our physical robot system in the future. All the protocol logic and conversion from integer values to the actual byte messages is performed by the driver class which also does a small amount of optimizations. For example, if the robot is told to turn 370 degrees, then the actual robot will never turn 370 degrees, but 10 degrees instead, as it would waste energy and time to make a pointless full turn.

Several configuration files exist for managing the environment, tag locations and robots. A map is defined in the environment configuration using plain text where one letter states the wall and another empty space. Tags with locations are listed in a separate configuration file where each line describes one RFID: an x coordinate, y coordinate, tag id and a room name which can have a special value when the tag located at the doorstep. Each robot has its own configuration file, which states its turning angles, trace color, log file locations, ports etc. Additionally there exists a global robot configuration file which describes the robot placements in the environment as well as the starting coordinates and robot headings. The default configuration tells robots to drive with a constant speed of 500mm/s for straight line movement. When a robot stops and turns standing still, then the movement speed is set 200mm/s making the Roomba to perform a full turn in around 8 seconds. For understanding the approximate room size the empty tile areas are summed. Tile side is equal to the Roomba robots diameter which is 34 centimeters.

The main features of the simulator:

- Movement logging.
- Automatic screenshots at each test run completion.
- Automatic test repetitions for result averages.
- Tag management with room identification.
- The original Roomba robot communication protocol between the simulator and the control algorithm.
- Creating test runs with predefined movement data: we use this functionality to create ideal situations for comparing the results of different algorithms and to replay previous test runs.

While moving around the simulated environment, each robot leaves behind a colored trace. After a test run completion it is convenient to get a firsthand evaluation of the efficiency by looking at the robot traces.

An example of the test run can be seen on the Figure 3.2 Example room setup for testing. The brick tiles on the figure represent walls and other impassible terrain. The little antennas around the map represent the RFID tags that are not yet found.

The tags are crossed through when a robot has seen them at least once. The dark filled circles represent the robot, the little red dots in front of the robot represent bumpers and the green dots in front of them represent the area where a robot is able to detect RFID tags. The legends can be seen on Table 3.2 Legend of simulator items. and Table 3.3 Legend of robot parts.

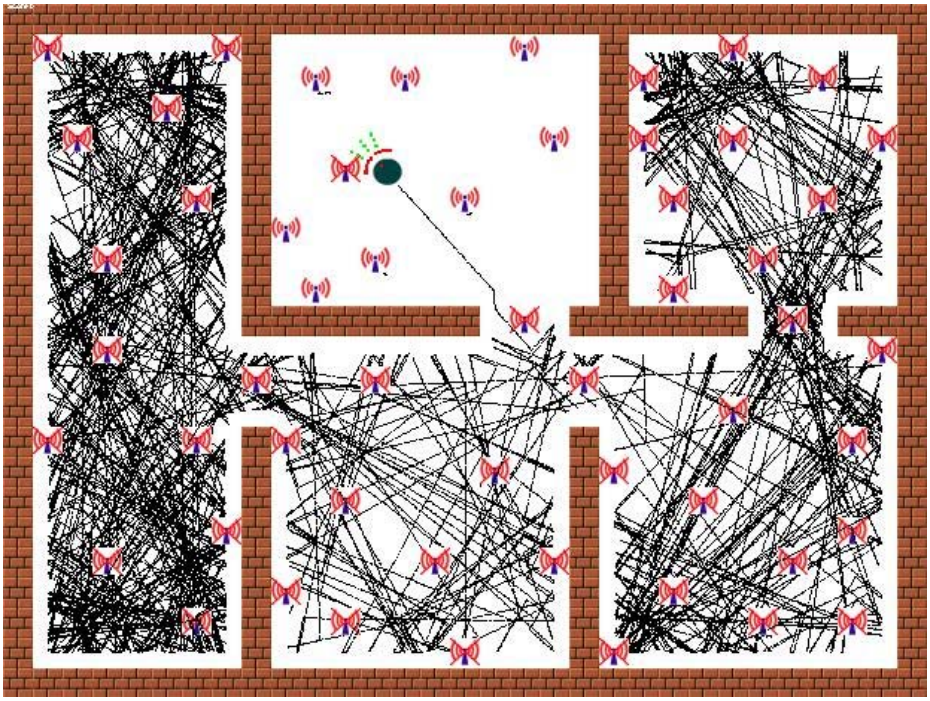

Figure 3.2 Example room setup for testing (published in paper C)

Table 3.2 Legend of simulator items.

	Not found RFID		Found RFID
	Wall		Robot

Table 3.3 Legend of robot parts.

	RFID detection area marked with the circle.
	Robot bumpers marked with the circle.
	Robot body marked with the circle.

3.2. Turning angle

During collision with the obstacle a robot can bump with either the left or right bumper or hit the barrier at ca 90 degrees, switching on both bumpers. After a head on impact a robot cannot continue its current path and must change its direction. Degrees to turn to at this situation are given in the robot configuration.

The turning angle can noticeably affect the robots performance. For finding the optimal value, tests were executed using a single robot with the turning degree values from 30 degrees to 120 degrees with a 15 degree step. Each test was run five times with random, history based and map aware algorithms. Results were compared and 30 degrees was selected as the most suitable turning angle for a single bumper (left or right) collision. If both bumpers hit, a random value from the 75 to 115 degree range is used. At first we experimented with a constant 90 degree turn, but found out that the random value from this range turned out to be more efficient.

During simulation we do not simulate the battery usage, but for the real world cases it should be remembered that the bigger the turning angle is the more energy is consumed to perform the movement.

To illustrate the effect of the turning angle we bring the Table 3.4 Average test runtimes in seconds by the turning angles of average run times for different angles (in the single bumper case) and different algorithms, produced by the simulations, in seconds.

Table 3.4 Average test runtimes in seconds by the turning angles

angle (degrees)	history based	random	map aware
30	1305	1975	1009
45	1188	3976	3101
60	1703	2016	2200
75	1628	5139	3078
90	4247	5800	1974
105	3714	5715	1915
120	2317	1975	2882

Looking at the experimental running times at Table 3.4 Average test runtimes in seconds by the turning angles we see that the 30 degree angle which was chosen for the overall testing is not the best angle for all cases. However, the first criteria for choosing the turning angle is that it must be the same for all algorithms, otherwise the preconditions would differ and the results are not comparable. Different values would result in unequal time spent for turning the robot and – for the real robot case – in a different power consumption during the experiments. For the random and map aware algorithms the 30 degree turning angle is the best angle. The history based algorithm has the best performance with a 45 degree angle and the second best with the 30 degree turning angle. Since the performance difference for the 30 degree angle is less than 10%, this led us to a decision to use the 30 degree turning angle also for the history based algorithm. Notice that in real robots the 30 degree angle would also use less energy for turning and thus conserve battery.

3.3. Success conditions for test runs

Finding each tag in an environment can take a remarkable amount of time. For example, a robot may find 18 tags out of the total 20 tags within a reasonable amount of time and then search for the last 2 RFIDs almost endlessly. Testing has showed that trying to search all the tags makes the test results vary a lot. We have implemented a configuration parameter – the so called cutoff value - which marks the percentage at which point the number of found tags will make the coverage task to be considered completed.

During the implementation of this parameter we made several test runs to find out the optimal cutoff value. The best value proved to be 85%. For example, if during the test run a robot has found 85 unique tags out of the 100 tags located on the environment then we consider the task to be completed. Various test experiments showed that all the algorithms are more or less impacted by the random factor and in some cases finding the last 15% can even take more time than finding the first 85% of the tags.

3.4. The Random algorithm

The random algorithm has the least amount of information known at the startup and during the entire problem solving time. During the initialization it will be given knowledge about the total amount of tags located in the environment. Encountering a tag makes the algorithm to store the tag id. For the random algorithm that is all the data needed for covering the area. For finding RFIDs in the area, the random algorithm employs a simple principle: the robot will drive around the environment and when it bumps into an obstacle the robot will back up a bit and then change its direction and move forward again.

For every test run the robot starts from the same place in the environment and drives at a straight line until it hits an obstacle. A robot can collide into an obstacle under some angle or directly head on. Hitting the wall or another robot under an angle makes the left or right bumper toggle, giving a signal of the collision.

A collision with a single causes the robot to back by some centimeters and then turn the robot to an opposite direction of the toggled bumper. For example, if the left bumper gives a signal, we turn the robot to the right. As described before, the degree of the turn has a significant effect on the room coverage efficiency [57]: hence we turn by 30 degrees which has proved to be the optimal for most cases.

A head on collision toggles both the left and right bumper: in such case the algorithm makes the robot back up and then turn at a random angle between 75 and 115 degrees. After finding the needed amount of tags in the environment, robot will stop and the simulator will start the next run.

3.5. The history based algorithm

This algorithm uses the previously described random algorithm as a base and extends it with additional logic. The main idea of this algorithm is to stop repeating the already passed paths and scatter the robots even more around the environment.

After seeing some tag sequence for the first time, the robot drives straight until it bumps with an obstacle and then makes a turn according to the principles from the random algorithm. The robot will store this information: for example, it has encountered 2 tags and finally turned 30 degrees. Seeing the same sequence of tags for the second time, the robot will immediately turn 30 plus a small delta of 15 degrees. Notice that the algorithm will not wait for the bump to occur: the turn will be executed right after the sequence is detected.

For sequence matching two or more tags have to be found during the single straight line move. There can be situations when the first robot movement line has a small offset from the second line and more tags are found in this case. Our algorithm takes these situations into account: when at the first move two tags A and B were found and during the second move the robot finds tags A, F, T and B, it will still detect a match with the sequence of A and B and will make the turn right after encountering the tag B. Encountering the tag sequence for the third time, the robot will turn 60 degrees based on the example, because the very last turn was 45 degrees.

The following is an example record for a single move:

StraightLineMove {

distanceMoved: 1200mm, turnAtTheEnd: 30, foundTags: [

tag {id: A, foundAtDistance: 350},

tag {id: B, foundAtDistance: 550}]

}

For each straight line move we store one such record, containing distance passed, the list of tags found during the drive and the degree turned at the end. Associated with the tag is the distance passed from the beginning of the current move.

During the moves where only a single tag was found the algorithm tries to scatter the movement by trying to detect if a similar movement has been recorded in the past, right after the robot has collided into an obstacle. To do that, the algorithm uses the foundAtDistance value associated with the tag, telling us the tag distance from the starting point. In case a similar movement has been performed before, the robot will turn 30 degrees plus the 15 degree delta. If the same situation occurs multiple times, the delta will be multiplied by the number of occurrences. The robot will never turn the entire circle because the driver optimizes the turns: for example, if the algorithm tells the robot to turn 750 degrees, the robot will actually turn only 30 degrees.

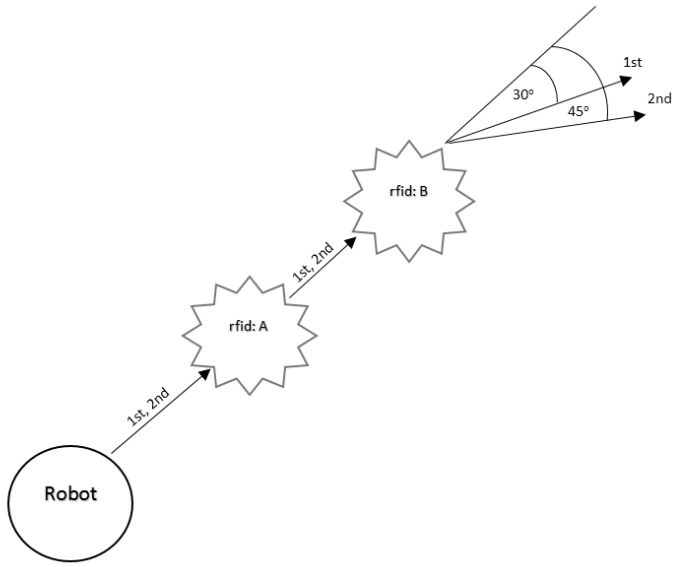

Figure 3.3 Figure indicating how the robot turns after finding the same tag sequence for two times during one test run.

3.6. The map aware algorithm

This algorithm requires the list of all tags present in the environment as input. Such information was not available for the previously described two algorithms. Similarly to the history-based algorithm it will use the random algorithm as the default strategy and when encountering two or more tags during a single straight line move the map aware algorithm will engage. The map aware algorithm knows all the tag locations on the map along with the room name where the tag is located. No additional information like the position of the walls, obstacles, the position of the robot starting points or positions of other swarm robots is provided. However, there are tags that are marked as door tags in order to help the robot to understand that it has left or entered the room.

The room id is detected from the first tag found in a room. The door tags will make the algorithm to reset the room id for the robot. The base configuration values for the turning angles are the same as used for the random algorithm.

In our simulator the tags contain approximate information about their coordinates – the tile id - not the exact coordinates. Actual precision would depend on the tile size, but during our tests the tile size is constant, equal to the size of a square with the side length being the robot body diameter. It can be argued that there is no need for higher precision due to the fact that while driving and turning the robot odometry loses accuracy. Robot estimation of its position is always imprecise.

The knowledge of the tag locations allows the robot to calculate the approximate new heading towards the closest unvisited tag, assuming the robot knows its direction and location. The robot is able to detect its position and heading only when it has found two or more tags in a single straight line move: due to the imprecise odometry it has no other persistent way to determine its location and direction.

The map aware algorithm starts to drive the robot around, using the random movement algorithm until it encounters two or more tags during a single straight line move. The known locations of these two tags are used to find the direction to the nearest not yet found tag in the same room. The robot is immediately stopped, the new heading is calculated and the robot is turned towards the nearest tag, commencing straight driving.

Due to the imprecise odometry the robot can miss the target tag and eventually bump into the obstacle: this causes the algorithm to switch to the random mode until two or more tags are again found during the straight line move.

It takes a lot of time to stop and turn. The map aware algorithm optimizes by calculating the presumed time to reach the next tag, adding the time spent while turning and the time spent while driving the between the current tag and tag to go to. For example, suppose we have two tags: one located 100mm away at a completely different direction and another 120mm away with almost no required direction change. Reaching the physically closest tag might require an algorithm to turn the robot for 150 degrees and then drive 100mm, while the tag located at

120mm would require only a 10 degree turn. Based on the time calculation it would be faster to drive to the tag located 120mm from current location.

Since the robot has no knowledge of its actual location, the distance between two tags is calculated based on the map description the robot has. Distance is calculated from one center point of the tag to another center point. RFID reader does not, in most cases, detect the center point of the tag. Stopping the robot takes some milliseconds as well. All that causes the loss in accuracy.

The map aware algorithm has no knowledge of the walls and obstacles inside the room. However, the nearest not yet found tag can be located at the other side of the wall or obstacle. Failing to reach the needed tag with bumpers giving a signal and seeing that the passed distance is smaller than the distance to the desired tag means that there is probably an obstacle on the way of the robot. For this situation the robot remembers that it encountering a tag sequence XY and wanting to reach the tag Z caused the collision into an obstacle. The algorithm decreases the probability of reaching the tag Z. Encountering the same XY tag sequence next time algorithm looks at the closest not yet found tags, sorted first by the distance and then by the reaching probability, thus moving the tag Z farther in the suggestion list. Tags in the suggested list of suitable tags are selected only from the current room.

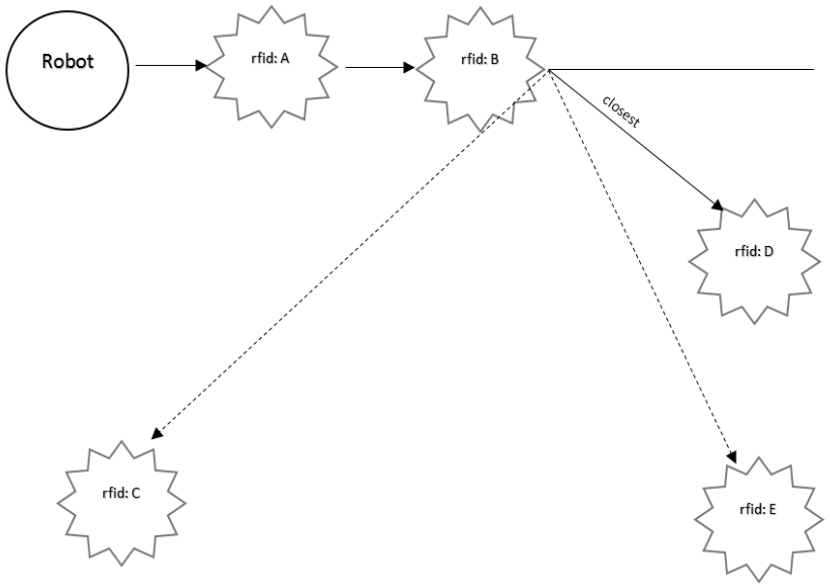

Figure 3.4 Figure indicating how the robot plans the move after encountering two or more tags during one straight line drive.

3.7. An extended map aware algorithm

Our latest published article (paper 4) mentions the use of the extended map aware algorithm as a special variation. In the current thesis we treat it as a separate algorithm, tested apart from the regular algorithm.

The regular map aware algorithm may accidentally move out of the room before finding all the tags. The extended map algorithm improves the regular map aware algorithm by trying to keep the robot inside the room until all the tags for the current room have been found. In order for this feature to work, the room entry points / doors must be marked with special tags.

Once the robot has found all the tags in the current room, it will attempt to exit the room by driving towards the special door marker tags. When encountering two or more tags during a single straight line drive the robot stops and calculates the angle to turn towards the closest door tag. Reaching the door tag and having found all tags in the current room, the robot drives over the door tags and exits. Otherwise it will make an approximately 180 degree turn and drive back to the room to search the not yet found tags in the room.

3.8. Strong and weak points of each algorithm

Before looking at the concrete results, we will give a brief overview of the most important observations for these four algorithms.

The random algorithm gives stable results for each environment, is able to offer competitive performance and is fairly easy to implement.

The history based algorithm is complicated to implement, can lose track in some situations and does not give as stable results as the random algorithm. The performance is similar or slightly better than the random algorithm for some cases. When the robot does not encounter two or more RFID tags in a row on a straight line move then it is not possible to use the history collected by the previous moves.

The performance of the map aware algorithm depends heavily on the setup of the rooms: it may exhibit very good or moderate results. The complexity of implementing the map aware algorithm is between the two beforementioned algorithms. The good results are achieved with more accurate navigation as a robot is aware of all the tag locations on the map. Knowing the coordinates of each tag means that after finding at least two tags in a row the robot is able to calculate the heading for the next closest tag. The results are greatly dependent of the room setup: for complex rooms the performance may turn out to be moderate.

The extended map aware algorithm is optimized for moving between the rooms. It tries to find all the tags in the room before leaving, otherwise the algorithm behaves as the regular map aware algorithm. Being able to move efficiently from one room to another results in a much better performance, but only in a multi-room environment. Basically, the additional ability to move from one room to another easier resolves some of the weaker points from the regular map aware algorithm.

3.9. Summary

The chapter "area coverage with a swarm" gave an overview of algorithms and the test methodologies. A custom built simulator was used to mimic the Roomba communication protocol. It has full control over each part of the testing process, as described in the section 3.1 "The simulator". The key configuration parameter – turning angle – which tells a robot how many degrees it has to turn when bumping into the obstacle is described in the section 3.2 "Turning angle". A short overview of the successful test conditions are given in the section 3.3 "Success conditions for test runs". All the coverage algorithms created are described in the sections 3.4 "The Random algorithm", 3.5 "The history based algorithm", 3.6 "The map aware algorithm" and 3.7 "An extended map aware algorithm". Each algorithm has some features that can either result in a performance gain or loss in some situations. The section 3.8 "Strong and weak points of each algorithm" gives a brief overview of these features.

4 EXPERIMENTAL RESULTS

Experiments with our algorithms can be divided into two parts. In the first part we compute the pseudo-optimal results. In the second part we compare the experimental runs of our four algorithms against the pseudo-optimal results obtained earlier. We will use the word "experiment" for a larger group of activities and test runs for a certain investigated algorithm. We will use the word "test" for a set of simulations with specific parameters, typically run during one experiment.

The experimental results have been published in the paper D in the appendix.

4.1. Configuration and comparison data

The simulator replay function was used to gather the data for more or less optimal runs that we call pseudo optimal test runs. During these test runs a simulated robot follows precisely the user-defined, nearly shortest path between the tags. The outcomes of these tests are used for comparison with the results of the previously described four algorithms to measure their performance.

Pseudo-optimal tests are being run with a single robot and configurations which correspond to the real algorithm tests. For the swarm case a presumption is made that adding the robots to the swarm has a linear effect to the results. For example, if a single robot is able to find all the needed tags in the environment with 60 seconds, then we presume that for the pseudo optimal test run 10 robots will find the tags in 6 seconds.

All the tests have been run with the same robot algorithm configurations – a single bumper turning degree 30 degrees, both bumpers turning degree randomly in a range of 75 to 115 degrees and a mandatory tag finding percentage 85. The swarm sizes for all beforementioned algorithms have been from a single robot to fifteen robots. Each test set for the presented results consists of five test runs. The graphs below indicate the average times of these five test runs. The total number of tests run for algorithm development and testing exceeds the 10 000 test runs. In addition to previously published articles the thesis covers also the testing results for the extended map aware algorithm.

4.2. Testing strategies and simulation areas

There are six groups of experiments, with the results of each group depicted on a graph presented below. The groups stem from three kinds of rooms:

- The whole area is just a single room without any obstacles.
- The whole area is split into three rooms without any internal obstacles in the room.
- The whole area is split into seven rooms with one of them being a corridor connected to all the other rooms.

There are two kinds of initial positions of the robots:

- All the robots start at the (almost) same place.
- The robots start at different places, distributed evenly in the rooms.

Starting from the different places at the beginning of a test run means that robots are spread around the map. For the next test run the starting positions will be the same: there is no random shuffling at the beginning of each test. The starting locations and tag locations are predefined in a configuration file. The following screenshots will give an overview of how the pseudo-optimal solutions look like when the robot has to find 85% of tags.

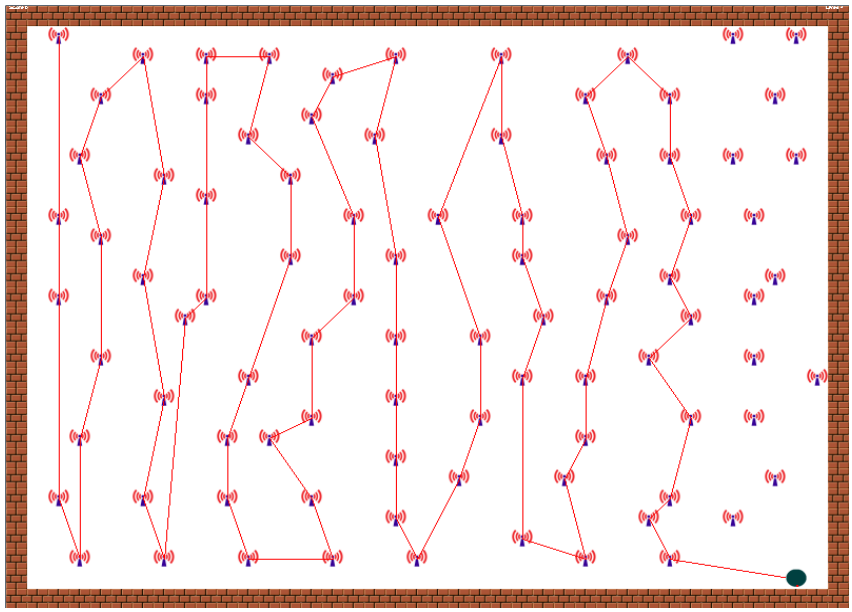

Figure 4.1 A single room pseudo-optimal run, takes 420 seconds.

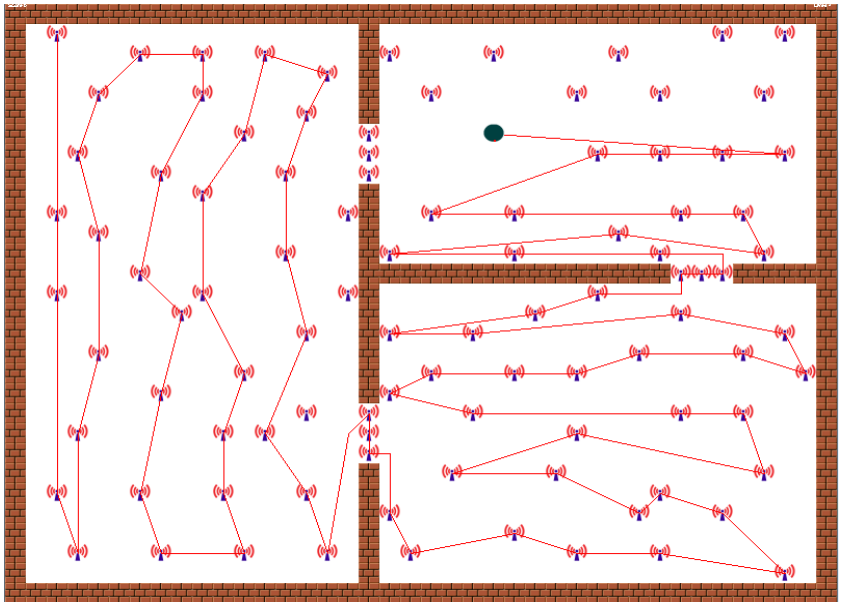

Figure 4.2 3-room pseudo-optimal run, takes 518 seconds.

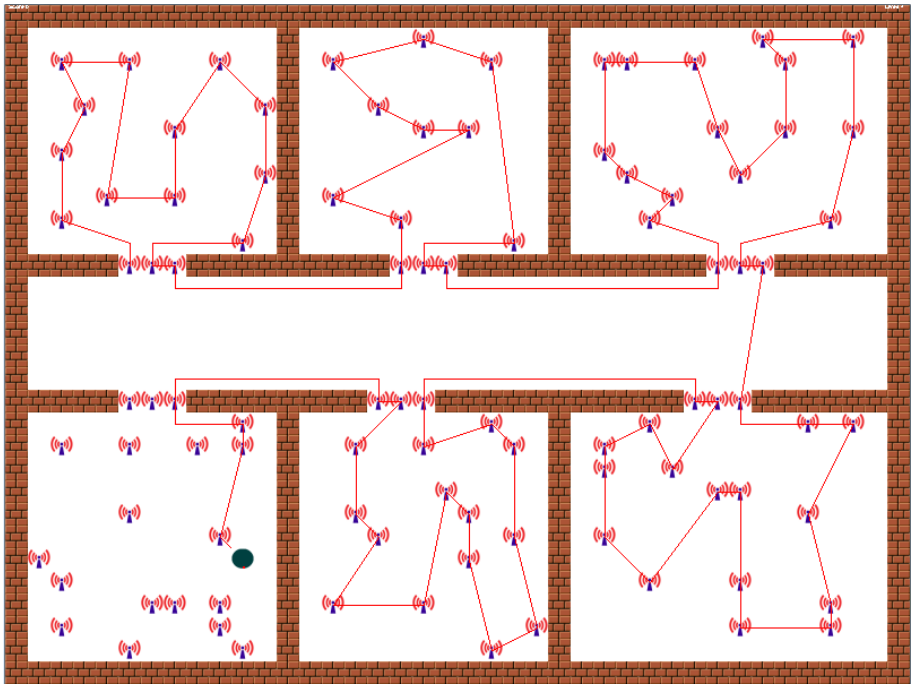

Figure 4.3 Seven-room pseudo-optimal run, takes 512 seconds.

A robot swarm participating in one test run shares the same configuration parameters and follows the same instructions as others. Scenarios where robots in the same swarm use different algorithms to solve the task have not been investigated.

It is important to note that the navigation algorithm of the robot simulates the random fluctuations of both the robot turning angles, turning times and detecting tags as they actually occur in the real Roomba cleaning robots.

The following screenshots show the finished state of problem solving of one test run in a single room with six robots, using the map-aware guidance algorithm and starting from the same location at the top of the room. The room has 90 tags randomly laid out as shown on the screenshot. An approximate environment area where the robot can move in this room is 123 square meters. Observe that some tags on the environment are not crossed through, meaning they have not been found: we use the 85% limit for marking the task completed.

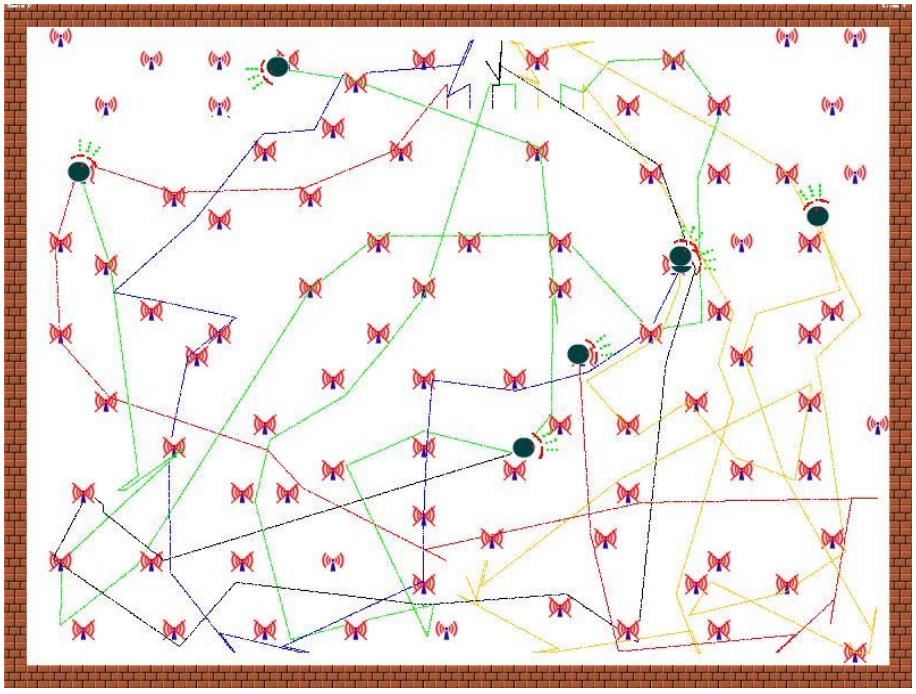

Figure 4.4 End position of a test run in a single room.

The following screenshot depicts the final state of a test run of a three-room environment with a swarm of five robots using the random guidance algorithm. Robots are distributed into various rooms at the startup. This room contains 90 regular RFID tags and nine specially marked tags at the doorsteps. The special door tags are used only by the map-aware and extended map-aware algorithms. An approximate size of the environment is 118 square meters. Observe that two robots have exited the original room where they started from and the other three have been staying in the initial room throughout the entire test run.

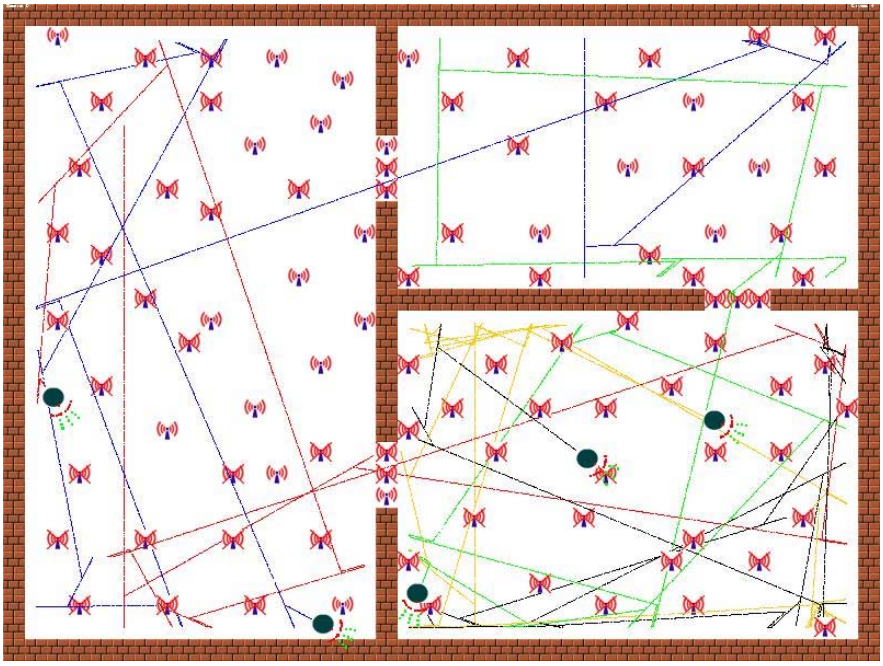

Figure 4.5 End position of a test run in a 3-room space.

4.3. Comparison of the simulation results

The next four graphs depict the results for our algorithms with different swarm sizes. Each graph depicts one group of experiments as described above and shows a calculated pseudo-optimal (ideal) run line for the current setup. The vertical axis indicates the average time of five runs. The horizontal axis indicates the number of robots in a swarm.

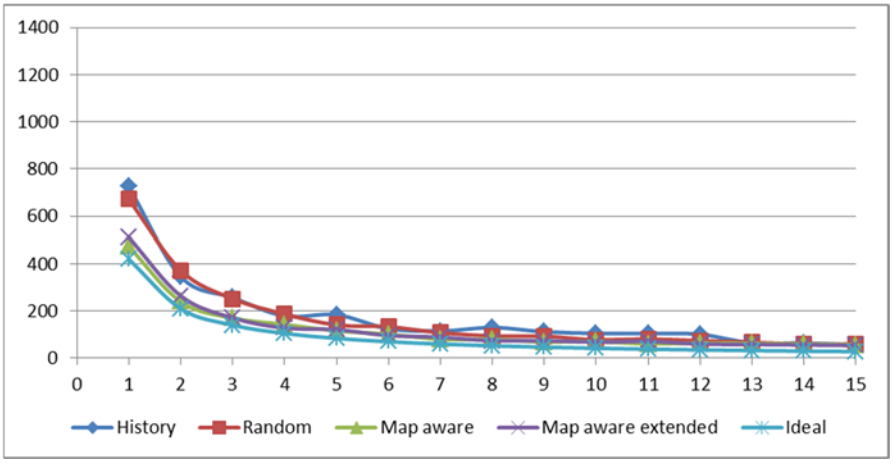

Figure 4.6 A single room with robots starting from one location. Y-axis values are representing experiment run times in seconds and x-axis shows the number of robots participating in the experiment.

The results presented on the Figure 4.6 above show that the history-based and the random algorithm perform almost identically: the timing differences between the runs are random fluctuations. Hence the seemingly useful idea of avoiding paths already travelled does not translate into clearly measureable gains for any swarm size. However, it has to be taken into account that - as mentioned before - the utilization of a history based algorithm is heavily affected by the environment, the amount of tags and a random factor. A robot just might not encounter already found tag sequences and be unable to take advantage of the collected information.

The map-aware algorithm constantly shows better performance than the simpler map-agnostic algorithms. The largest performance gain achieved from the map-aware algorithm occurs with small swarm sizes (58% of the coverage time of the random/history based algorithms for the one-robot case) and it shrinks with greater swarm sizes (68% for a 7-robot swarm). However, it is important to notice that the benefit of increasing the swarm size is very strong for swarm sizes of up to seven robots for our environments. Adding robots to larger swarms does not bring noticeable changes to the results.

The coverage time of the two-robot case is almost twice smaller than for the one-robot case (regardless of the algorithm), and the coverage time of the four-robot case is, again, almost twice smaller than for the two-robot case. However, looking at the graphs last two points 14 and 15, the difference between problem solving times is insignificant (again, regardless of the algorithm). We could pose a hypothesis that one of the reasons for diminishing gains is the need to travel to the far corners and edges from the common starting point: the time it takes is roughly the same regardless of the swarm size. This leads us to the next experiment: the same (single) room with the robots starting from various locations on the environment chosen randomly. The only restriction is that the robots are placed near the walls and at the startup they head towards the center of the room.

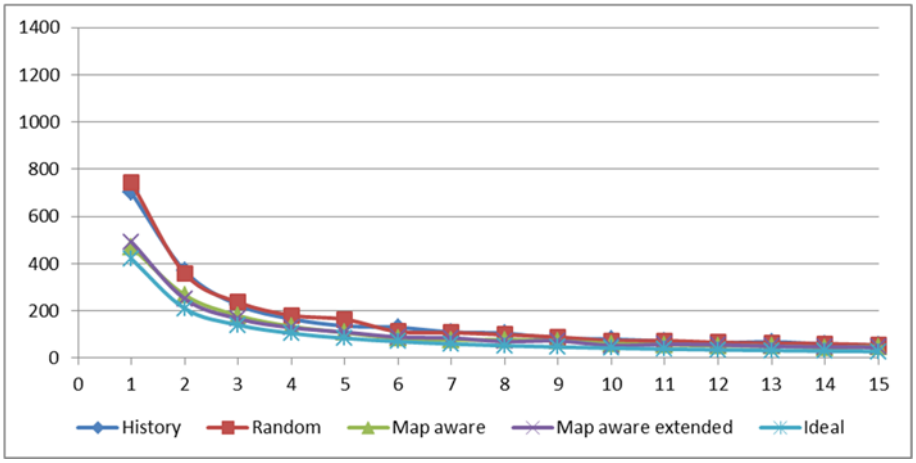

Figure 4.7 A single room with robots starting from different locations. Y-axis values are representing experiment run times in seconds and x-axis shows the number of robots participating in the experiment.

According to our test setup and maps the graphs start to go flat after the 7th or 8th robot joins the swarm. Starting from a single location casuses the robots to consume more time to reach different parts of the map, compared to the situation where robots are placed all around the map. Theoretically, for our tests with eight or more robots the resolving times keep decreasing by tiny steps when adding robots to swarm, until every tile contains a robot. A map fully covered with robots would be solved in a couple of seconds – the robots must start, rfid readers must detect the tags and after merging the info of the tags found the test ends.

Looking at the Figure 4.6 we notice that flattening does not occur as quickly as on the Figure 4.7. For example, consider the swarm size of seven robots. When the robots start from one location, the random algorithm is able to perform the task in 108 seconds, the history based algorithm in 115 seconds, the map aware in 80 seconds and the extended version of the map aware algorithm in 89 seconds. The pseudo-optimal time for a swarm size of seven robots is 60 seconds.

Looking at the next point of the graph – the swarm size being eight robots - the result for the random algorithm is 94 seconds, for the history based it is 130 seconds, for the map aware it is 80 seconds and the extended map aware algorithm manages to solve the task in 74 seconds. The pseudo optimal time is 52 seconds.

We notice that the history based algorithm with 8 robots performed even worse than it did with seven robots. The reason is that all the algorithms follow some percentage of time according to the random movement principles. During that time they are not able to apply their specific logic, basically meaning having not found two or more tags during straight line drive. Increasing the robot swarm size from seven to eight robots has shortened the problem solving times up to 20%. Moving further and looking at the swarm of size 14 robots, the time consumed by the random algorithm is 60 seconds, the history based algorithm is able to complete the task in 64 seconds, the map aware in 60 seconds and the extended map aware in 56 seconds. The pseudo optimal time for a swarm of 14 robots is 30 seconds.

Comparing the results with swarm sizes of seven (Table 4.1 Results represented in seconds for seven robot swarm experiment with same and various start locations.) and fourteen (Table 4.3 Results represented in seconds for fourteen robot swarm experiment with same and various start locations.) tells us that the random algorithm time has decreased by 80%, the history based algorithm has decreased by 79.6%, the map aware by 33.3% and the extended map aware by 59%. The pseudo optimal time has changed 50%.

Looking at the same points for Figure 4.7 with robots starting at various locations in the environment with seven robots, the random algorithm performs the task in 108 seconds, the history based algorithm is able to finish with 110 seconds, the map aware completes in 76 seconds and the extended map aware in 85 seconds.

Moving forward to the results for swarm with the size of eight robots (Table 4.2 Results represented in seconds for eight robot swarm experiment with same and various start locations.) the random algorithm gets the job done in 99 seconds, the history based algorithm completes its task in 105 seconds, the map aware is able to finish in 78 seconds and the extended map aware in 70 seconds. Pseudo optimal time for the task completion is the same - for seven robot swarm 60 seconds and for eight robot swarm 52 seconds - as it was for the comparisons for Figure 4.6. Time difference comparing the swarms with sizes of seven and eight for random algorithm has decreased around 9%. The history based algorithm has performed about 5% better, the map aware has almost the same result, but still the performance has gone down and problem solving time has increased by 2.5%. The extended map aware has been able to solve the problem with eight robots 21% faster than with the seven robot swarm. Pseudo-optimal performance time has decreased by 14%.

Looking at the resolving times for the swarm with fourteen robots, the random algorithm solves the problem with 61 seconds, the history based in 59 seconds, the map aware algorithm in 45 seconds and the extended map aware in 47 seconds.

Looking at the time change percentages against the swarm size change by one robot it can be seen that in case of a common start location the change in values is greater than for the configuration where the robots start from various places around the map.

Importantly, we observe there is no significant time difference for random and history based algorithms between the results with the same location and varying location starting placement strategies illustrated on the Figure 4.6 and Figure 4.7. An exception is a history based algorithm with the swarm size of eight robots, which is presumably a random fluctuation.

Table 4.1 Results represented in seconds for seven robot swarm experiment with same and various start locations.

seven robots	random	history based	map aware	ext. map aware
same start location	108	115	80	89
various start location	108	110	76	85
	0,00%	4,55%	5,26%	4,71%

Table 4.2 Results represented in seconds for eight robot swarm experiment with same and various start locations.

eight robots	random	history based	map aware	ext. map aware
same start location	94	130	80	74
various start location	99	105	78	70
	-5,05%	23,81%	2,56%	5,71%

Table 4.3 Results represented in seconds for fourteen robot swarm experiment with same and various start locations.

fourteen robots	random	history based	map aware	ext. map aware
same start location	60	64	60	54
various start location	61	59	45	47
	-1,64%	8,47%	33,33%	14,89%

Table 4.4 Results represented in seconds for fifteen robot swarm experiment with same and various start locations.

fifteen robots	random	history based	map aware	ext. map aware
same start location	60	56	56	54
various start location	55	54	49	46
	9,09%	3,70%	14,29%	17,39%

We see that robots spread very quickly in the same-location scenario and the initial spreading process has little effect on the overall time for simpler algorithms or smaller swarm sizes. Quick spreading is made possible by the random fluctuations of robot behavior as described above. For map aware and extended map aware the spreading is not that important. Knowing the locations of each tag and finding a sequence of at least two tags during a single straight line move directs the robot towards the nearest not yet found tag: this principle causes spreading.

The next two experiments are conducted in a simple three-room space shown on the Figure 4.5 End position of a test run in a 3-room space. above. The size of the space is approximately same and the number of tags is the same as in the previous one-room experiment, with an exception of nine additional doorstep tags.

The principal complexity of covering a multi-room space stems from the problem of a robot (or several robots) being stuck in a few rooms and not reaching all the rooms necessary to obtain the 85% tag coverage condition. In the robot room coverage literature this problem is typically alleviated by different planning algorithms. Not so in our simple algorithms: the random and history-based algorithms are completely unaware of the geometry of the room the robot is in or the tags located in the room at any given moment.

However, the map-aware algorithm always has the knowledge as to which room the found tag belongs. It first tries to find all the tags in the room and after this it attempts to escape the room by driving toward the closest door marking tags.

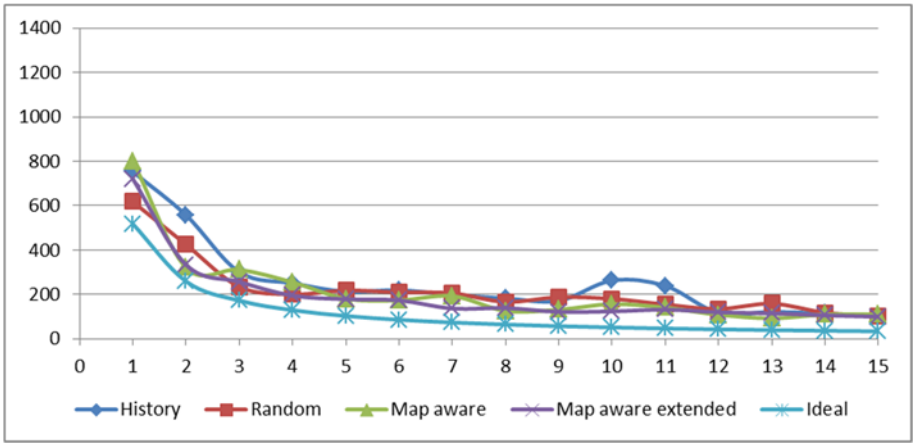

Figure 4.8 A three-room space with robots starting from one location. Y-axis values are representing experiment run times in seconds and x-axis shows the number of robots participating in the experiment.

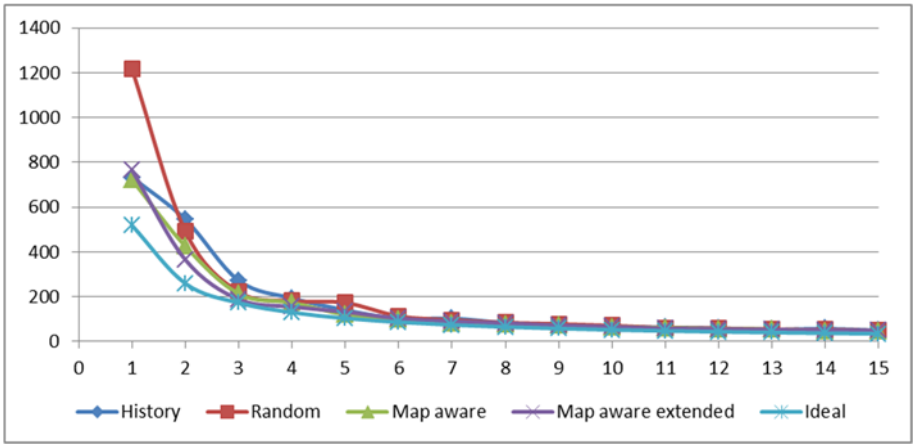

Figure 4.9 A three-room space with robots starting from different locations. Y-axis values are representing experiment run times in seconds and x-axis shows the number of robots participating in the experiment.

A single robot is much more affected by the random factor than the entire swarm, hence it is hard to give a credible comparison for the results of a single robot for the three room space. As it can be seen from the graphs, one robot with a random algorithm is able to solve the task for the three room space even faster at one test run, when compared to the single room testing. However looking at the Figure 4.9 it can be seen that the time for three room space with the random algorithm has almost doubled. On Figure 4.8 and Figure 4.9 the one robot test runs are basically the same: with a single robot the various starting locations do not have an effect. The history based and map aware algorithms have all constantly increased the task solving times for the three room space. Increasing the swarm size causes the fluctuating result time for the random algorithm to diminish. However, for larger swarms the time difference increases: it takes about twice as

long to cover the three-room space for the 7-robot swarm when compared to the one-room space. The reasons for the performance loss are hidden in the spreading, which is much more complex for environments with several rooms when compared to one single open space. However, the overall trend is the same: increasing the size of a robot swarm first results in a rapid drop of the problem solving time, but after a while it will decrease less and less.

The next experiment is conducted in the same environment as the previous one with the difference that all the robots are spread out evenly around the map. At start there is at least one robot in each room, assuming the swarm size has increased to three robots or more. Increasing the swarm size, we see that the form of the graph starts to become similar to the one-room case, which is – again – not unexpected, since there will be at least one robot for each room in the initial situation. Again, the positive effect of increasing the swarm becomes smaller and smaller as the robot swarm grows.

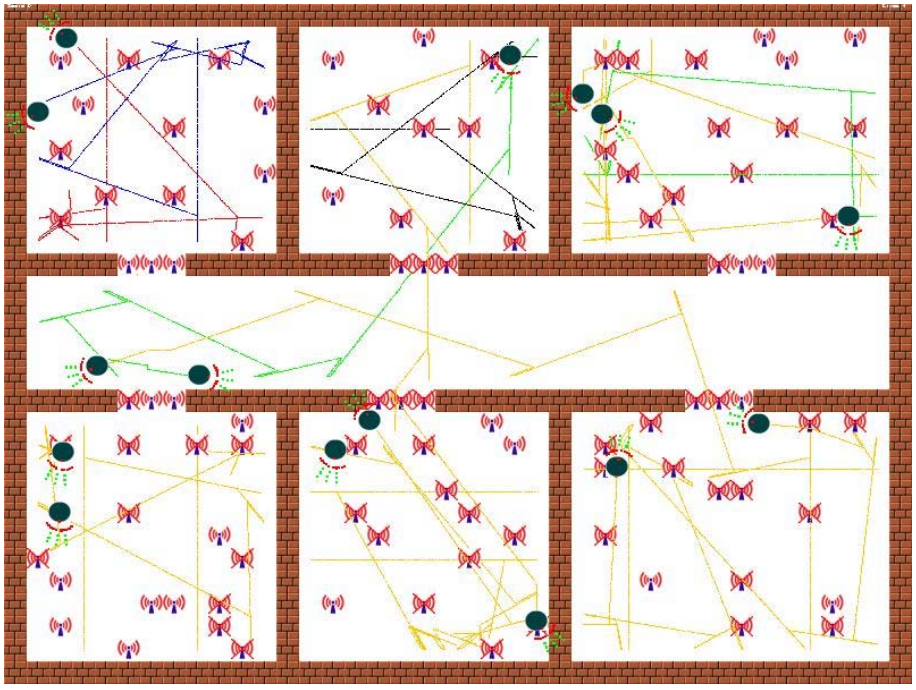

Figure 4.10 Random algorithm with 15 robots, starting from different locations in a seven-room space.

Finally we have chosen the seven-room setup with six small rooms and a corridor. The map has about 111 square meters of space and there are 100 tags located around the environment, including the 18 special tags placed on at the doorways.

For this complex environment the key factor for quick coverage times is spreading robots across all the rooms. Based on the results it is somewhat surprising that starting from different locations (robots evenly distributed in the rooms versus all robots together in a corridor) has a relatively small effect on

coverage time. It is also worth noting that increasing the size of a swarm has a significant effect until the amount of robots is equal or larger than the number of rooms, after which adding new robots has a negligible effect on the coverage time.

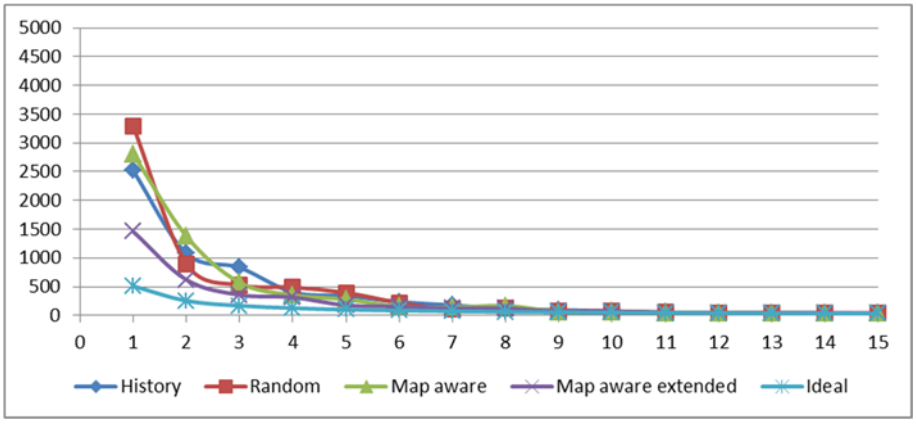

Figure 4.11 Six small rooms with a corridor, robots starting from different locations. Y-axis values are representing experiment run times in seconds and x-axis shows the number of robots participating in the experiment.

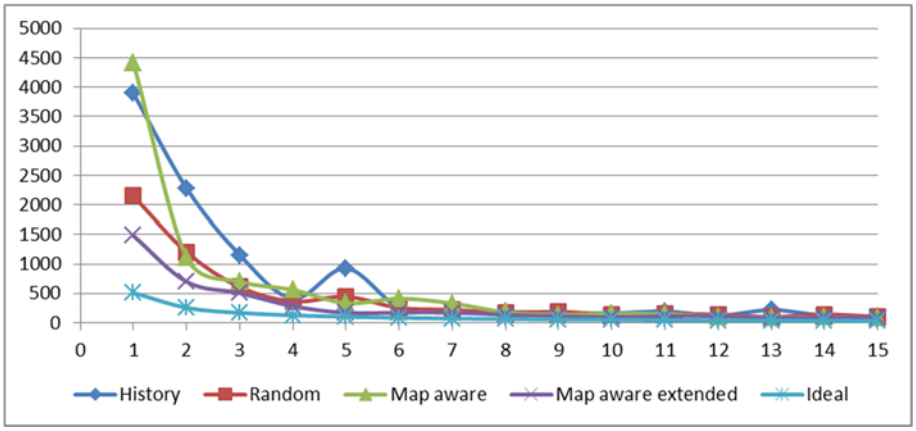

Figure 4.12 Six small rooms with a corridor, robots starting from one location. Y-axis values are representing experiment run times in seconds and x-axis shows the number of robots participating in the experiment.

4.4. Environment versus the swarm size

When we compare all the six graphs presented, it can be clearly seen that the random factor has an effect on the results. It should be expected that a larger amount of test runs would smoothen out the random factor in results, yet it remains a significant aspect for the actual use of the swarm.

While single room graphs look smooth, the graphs for multiple room setups have several “bumps”. The reason for these fluctuations relies on the navigation issues

as robots have to move between rooms and are unable to always pass the doorways. Our hypothesis for the multi-room spaces is that the main speedups can be gained by spreading the robots evenly in the rooms, which should be attainable even by employing the tags (or other landmarks) located at doors only. It is also possible that it would be sufficient to employ a random algorithm for all the situations except when the robot is located at the door and should decide whether to enter the door or not. This hypothesis should be tested in future work.

In order to compare the swarm results for all the three environments described above we present the graph of averages compiled from all the algorithms. Each line on the graph represents average results for all the algorithm results merged together. The graphs representing coverage time as a function from the size of the swarm S are roughly $a*S^{-0.86}$ for a single room, $b*S^{-1.09}$ for the three-room setup and $c*S^{-1.58}$ for the seven-room setup, where a, b and c depend on the room size and robot speed.

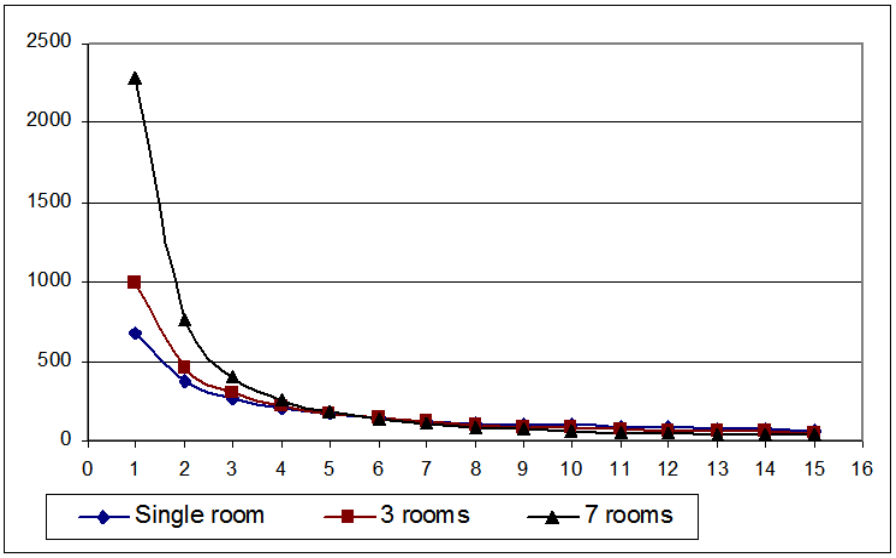

Figure 4.13 Averages for all the three algorithms. Y-axis values are representing experiment run times in seconds and x-axis shows the number of robots participating in the experiment.

4.5. Algorithm performance for very high RFID densities

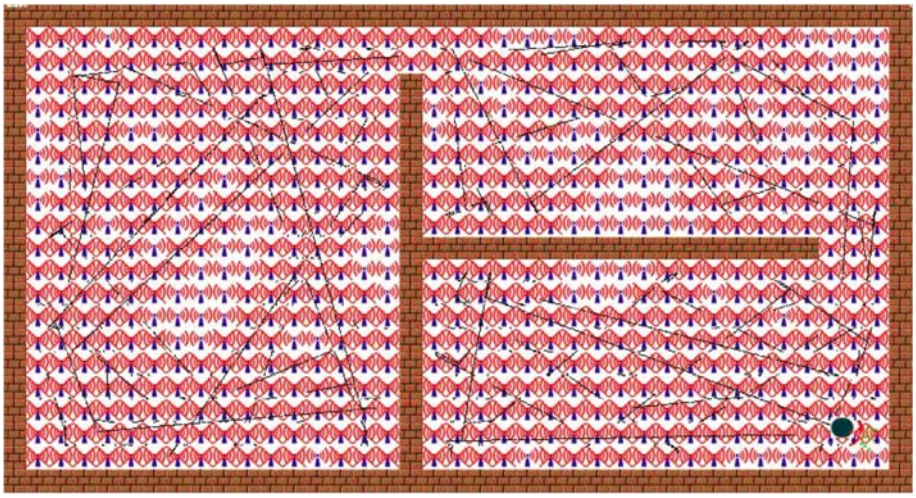

Figure 4.14 Extreme situation in an experimental room setup with all the available space covered with tags (published in paper C).

Finally we present a short overview of the experiments with a three room space which has a very high tag density, up to every single tile being filled with an RFID tag. Such tests give valuable input for tuning the algorithms.

The first group of tests has been ran with various room setups and randomly placed tags with a coverage around 15 – 25%.

The greatest impact of dense tag positioning was observed for the history based and map aware algorithms, as could be predicted. Both of these algorithms make decisions based on the observed tag sequences, while the random algorithm is just driving the robot around. We ran all the beforementioned room setups with multiple test runs. The results indicated rising solving times. One of the reasons is that a tag will not always be registered when it is in the radio coverage. Also, for the map aware algorithms the high density causes the robots to turn very often, thus losing a lot of time turning, not driving. This is caused by the fact that a robot almost never discovers the tag at its center point and is not able to navigate precisely due to its weak odometry.

Running our algorithms in rooms completely filled with RFID tags resulted in an average run time increase up to 50%. The worst cases actually ended with the time loss of 200% or more, although the proportion of so bad cases was lower than 5% of all runs.

4.6. Summary

In this chapter we have presented an overview of the results of tests and the environments where they were tested in. The section 4.1 “Configuration and comparison data” briefly describes the robots turning angle configuration and the base data on which the results are compared. Algorithms were tested in various room setups and different swarm distribution strategies. Each test strategy is described in the section 4.2 “Testing strategies and simulation areas”. Comparisons of the results of testing are described in the section 4.3 “Comparison of the simulation results”. Different room setups have a distinct effect on the performance of our coverage algorithms. The section 4.4 “Environment versus the swarm size” gives an overview of these effects. Regular tag coverage of the room is around 20% and all the algorithms have been tested in these conditions. Since there could also occur irregular situations, the section 4.5 “Algorithm performance for very high RFID densities” gives an overview of how do the extreme tag densities affect the performance of our algorithms.

CONCLUSIONS

Our main goal was to investigate and develop algorithms using navigational tags for enhancing the performance of a swarm of robots when precise navigation is hard to achieve or not feasible.

First, we have designed and presented a knowledge architecture for intelligent robots operating as a swarm, able to use RFID tags both as landmarks and communication channels. The architecture is based on using extended RDF triples for knowledge represenation on all levels: tags, robot knowledge base and the swarm knowledge base on the server. We have also designed a rule system for robots, providing reactive control while a robot is in action.

This architecture was implemented for iRobot Roombas extended with RFID readers and new control software. The demonstrated ability of the real swarm of physical robots to solve given tasks indicates the feasibility of the architecture.

Second, we have developed and investigated four different, robust coverage algorithms for swarms of simple robots tasked with cleaning, search or similar activities inside buildings. In order to run realistic tests we have developed a simulator closely matching the actual capabilities and behaviour of the real cleaning robots. The key findings of the experiments are as follows:

- A specific parameter of the robot behaviour - the default turning angle - makes a significant difference for the performance of all the investigated algorithms. One of the main reasons is that bigger turns take more time than smaller turns, thus wasting time which should be spent covering the room. As shown in the table 3.4, the 30 degree turn is the best or next best choice for the algorithms experimented with.
- The algorithms knowing the locations of landmarks are consistently better than the parameter-optimized random algorithm: roughly estimated 20% faster for small swarms, with significant variations stemming from room setups and smaller improvements for larger swarms: see tables 4.1 – 4.4. They are also, on the average, close enough to the ideal behaviour to be considered as practically sufficient: running time is roughly 1.2 times of the ideal, although this varies depending on the room setup and swarm size, see figures 4.6 – 4.12.
- Assuming the tag location reader is inexact, then for rooms with a very high density of landmark locations it is important to avoid using all the landmarks for potential optimizations of the search path. The main reason is misreading the location of closely positioned tags and the time spent (incorrectly) turning at too many landmarks, thus wasting more time spent on turning than the optimizations gain. As presented in section 4.5, in our experiments the time spent in a room with a very high-density tag cover was roughly 1.5 times the time spent in a “normal” tag density room.
- As the swarm size and density increases, the performance improvements gained by better algorithms and more knowledge decrease quickly: in the

other words, increasing the size of the swarm dominates the effect of having better sensors and more intelligent behaviour. This effect can be seen best on figures 4.6 – 4.12 and as a summary on the figure 4.13. For example, for our three-room setup increasing the size of the swarm from one robot to two robots decreases the running time ca two times, increasing from two robots to seven decreases the time ca four times and from seven to fourteen ca two times. The coverage time as a function from the size of the swarm S is roughly $b*S^{-1.09}$ for the three-room setup, where b depends on the room size and robot speed. In contrast, the improvements gained from better algorithms and more knowledge are ca 20%.

For the future work, it would be interesting to consider the swarm algorithms tuned to the use of the camera module included in the newer cleaning robots, in contrast to the algorithms focused on finding RFIDs in the environment. While there are significant similarities between these approaches and we believe that our key findings still hold, there are also important differences and potential improvements to be made.

REFERENCES

[1] Y. Altshuler Y, A.M. Bruckstein, I.A. Wagner: Swarm Robotics for a Dynamic Cleaning Problem. In "IEEE Swarm Intelligence Symposium", pages 209–216, 2005.

[2] T. Tammet, J. Vain, A. Kuusik: "Using RFID tags for robot swarm cooperation". WSEAS Transactions on Systems, 5(5), pages 1121–1128, 2006.

[3] H. Endres et al., "Field test of a navigation system: autonomous cleaning in supermarkets," in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 1998, pp. 1779–1781.

[4] R. Murphy, "Human-robot interaction in rescue robotics," IEEE Syst., Man, Cybern., C, Appl. Rev., vol. 34, no. 2, pp. 138–153, May 2004.

[5] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, S. Thrun. The interactive museum tour-guide robot. Proc. AAAI-98, Madison, WI (1998)

[6] Y. Huang et al., "Automatic operation for a robot lawn mower," in SPIE Conf. Mobile Robots, vol. 727, 1986, pp. 344–354

[7] D. Hougen et al., "A miniature robotic system for reconnaissance and surveillance," in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 2000, pp. 501–507.

[8] S. Srinivasa, D. Ferguson, C. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, M. V. Weghe. Herb: A Home Exploring Robotic Butler. Autonomous Robots, 2009

[9] Space mining at http://spectrum.ieee.org/automaton/robotics/military-robots/nasa-training-swarmie-robots-for-space-mining (25.04.2017)

[10] J. Haverinen and A. Kemppainen, "A global self-localization technique utilizing local anomalies of the ambient magnetic field.", International Conference on Robotics and Automation, pp 3142 – 3147, 2009.

[11] M.A. Batalin, and G.S. Sukhatme, "Coverage, Exploration and Deployment by a Mobile Robot and Communication Network", in Proc. International Workshop on Information Processing in Sensor Networks, 2003, pp. 376 – 391

[12] N. Karlsson, E.D. Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian and M.E. Munich, "The vSLAM Algorithm for Robust Localization and Mapping," Proc. IEEE Int',l Conf. Robotics and Automation, 2005.

[13] Dorigo, M., Bonabeau, E., & Theraulaz, G. (2000a). Ant algorithms and stigmergy. Future Generation Computer Systems, 16(8), 851–871.

[14] Ziparo, V.A., Kleiner, A., Nebel, B., and Nardi, D. (2007). Rfid-based exploration for large robot teams. In IEEE International Conference on Robotics and Automation, 4606-4613.

[15] N. Agmon, N. Hazon, and G. A. Kaminka, "Constructing spanning trees for efficient multi-robot coverage," in Proceedings of the 2006 IEEE International conference on robotics and Automation, vol. 1-10, (Orlando, FL, USA), pp. 1698-1703, 2006.

[16] Burgard, W.; Moors, M.; Stachniss, C.; Schneider, F.E., "Coordinated multi-robot exploration," Robotics, IEEE Transactions on , vol.21, no.3, pp.376,386, June 2005

[17] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon decomposition", in: Proc. of Int. Conf. on Field and Service Robotics, Canberra, Australia, December 1997.

[18] S. Schneegans, P. Vorst and A. Zell, "Using RFID Snapshots for Mobile Robot Self-Localization.", European Conference on Mobile Robots, pp. 1 – 6, 2007.

[19] D. Hahnel, W. Burgard, D. Fox, K. Fishkin and M. Philipose, "Mapping and localization with RFID technology.", International Conference on Robotics and Automation, pp. 1015 – 1020, 2004.

[20] J. Bohn and F. Mattern, "Super-Distributed RFID Tag Infrastructures." Lecture Notes in Computer Science, vol 3295, pp. 1 – 12, 2004.

[21] S. Park and S. Hashimoto, "Indoor localization for autonomous mobile robot based on passive RFID.", International Conference on Robotics and Biomimetics, pp 1856 – 1861, 2009.

[22] M. Baglietto, G. Cannata, F. Capezio, A. Grosso, A. Sgorbissa and R. Zaccaria, "PatrolGRAPH: a Distributed Algorithm for Multi-Robot Patrolling", IAS10 - The 10th International Conference on Intelligent Autonomous Systems, Baden Baden, Germany, pp. 415 – 424, July 2008.

[23] N. Agmon, N. Hazon, and G. A. Kaminka, "Constructing spanning trees for efficient multi-robot coverage," in Proceedings of the 2006 IEEE International conference on robotics and Automation, vol. 1-10, (Orlando, FL, USA), pp. 1698-1703, 2006.

[24] Burgard, W.; Moors, M.; Stachniss, C.; Schneider, F.E., "Coordinated multi-robot exploration," Robotics, IEEE Transactions on , vol.21, no.3, pp.376,386, June 2005

[25] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon decomposition", in: Proc. of Int. Conf. on Field and Service Robotics, Canberra, Australia, December 1997.

[26] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. Annals of Mathematics and Artificial Intelligence, 31, pp 77-98, 2001

[27] O.Hachour. Path planning of Autonomous Mobile robot. International Journal of Systems Applications, Engineering & Development, issue 4, vol 2, pp 178-190, 2008

[28] Ackerman, E. & Guizzo, E. (2015). iRobot Brings Visual Mapping and Navigation to the Roomba 980 http://spectrum.ieee.org/automaton/robotics/home-robots/irobot-brings-visual-mapping-and-navigation-to-the-roomba-980

[29] Čelan, V., Stančić, I., & Musić, J. (2016, July). Cleaning up smart cities—Localization of semi-autonomous floor scrubber. In Computer and Energy Science (SpliTech), International Multidisciplinary Conference on (pp. 1-6). IEEE.

[30] Rubenstein, Michael, Alejandro Cornejo, and Radhika Nagpal. "Programmable self-assembly in a thousand-robot swarm." Science 345.6198 (2014): 795-799.

[31] Kilobot specifications at http://www.k-team.com/mobile-robotics-products/kilobot/specifications (25.04.2017)

[32] Rubenstein, Michael, Christian Ahler, and Radhika Nagpal. "Kilobot: A low cost scalable robot system for collective behaviors." Robotics and Automation (ICRA), 2012 IEEE International Conference on. IEEE, 2012.

[33] Blender, T., Buchner, T., Fernandez, B., Pichlmaier, B., & Schlegel, C. (2016, October). Managing a Mobile Agricultural Robot Swarm for a seeding task. In Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE (pp. 6879-6886). IEEE.

[34] Marjovi, A., Nunes, J., Sousa, P., Faria, R., & Marques, L. (2010, May). An olfactory-based robot swarm navigation method. In Robotics and Automation (ICRA), 2010 IEEE International Conference on (pp. 4958-4963). IEEE.

[35] Konur, S., Dixon, C., & Fisher, M. (2012). Analysing robot swarm behaviour via probabilistic model checking. Robotics and Autonomous Systems, 60(2), 199-213.

[36] Ferrante, E., Turgut, A. E., Huepe, C., Stranieri, A., Pinciroli, C., & Dorigo, M. (2012). Self-organized flocking with a mobile robot swarm: a novel motion control method. Adaptive Behavior, 20(6), 460-477.

[37] Zhai, C., & Hong, Y. (2013). Decentralized sweep coverage algorithm for multi-agent systems with workload uncertainties. Automatica, 49(7), 2154-2159.

[38] Sartoretti, G., Hongler, M. O., de Oliveira, M. E., & Mondada, F. (2014). Decentralized self-selection of swarm trajectories: from dynamical systems theory to robotic implementation. Swarm Intelligence, 8(4), 329-351.

[39] Couceiro, M. S., Martins, F. M., Rocha, R. P., & Ferreira, N. M. (2014). Mechanism and convergence analysis of a multi-robot swarm approach based on natural selection. Journal of Intelligent & Robotic Systems, 76(2), 353-381.

[40] Pásztor, A. (2014). Gathering simulation of real robot swarm. Tehnicki Vjesnik-Technical Gazette, 21(5).

[41] Alers, S., Tuyls, K., Ranjbar-Sahraei, B., Claes, D., & Weiss, G. (2014). Insect-inspired robot coordination: foraging and coverage. Artificial life, 14, 761-768.

[42] Bhattacharya, S., Ghrist, R., & Kumar, V. (2014). Multi-robot coverage and exploration on Riemannian manifolds with boundaries. The International Journal of Robotics Research, 33(1), 113-137.

[43] Valentini, G., Hamann, H., & Dorigo, M. (2015, May). Efficient decision-making in a self-organizing robot swarm: On the speed versus accuracy trade-off. In Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (pp. 1305-1314). International Foundation for Autonomous Agents and Multiagent Systems.

[44] Maftuleac, D., Lee, S. K., Fekete, S. P., Akash, A. K., López-Ortiz, A., & McLurkin, J. (2015, May). Local policies for efficiently patrolling a triangulated region by a robot swarm. In Robotics and Automation (ICRA), 2015 IEEE International Conference on (pp. 1809-1815). IEEE.

[45] Becker, A., Fekete, S. P., Kröller, A., Lee, S. K., McLurkin, J., & Schmidt, C. (2013, June). Triangulating unknown environments using robot swarms. In Proceedings of the twenty-ninth annual symposium on Computational geometry (pp. 345-346). ACM.

[46] Zheng, S., Hong, J., Zhang, K., Li, B., & Li, X. (2016). A multi-frame graph matching algorithm for low-bandwidth RGB-D SLAM. Computer-Aided Design, 78, 107-117.

[47] Luo, C., Yang, S. X., Li, X., & Meng, M. Q. H. (2017). Neural-Dynamics-Driven Complete Area Coverage Navigation Through Cooperation of Multiple Mobile Robots. IEEE Transactions on Industrial Electronics, 64(1), 750-760.

[48] Hayes-Roth, B. (1985). A blackboard architecture for control. Artifcial Intelligence, 26(3), 251-321.

[49] Tammet, T. (1997). Gandalf. Automated Reasoning, 18(2), 199-204.

[50] Robinson, J.A. and Voronkov, A. (eds.) (2001). Handbook of Automated Reasoning. MIT press.

[51] PostgreSQL at https://www.postgresql.org/ *(25.04.2017)*

[52] R. Morlok and M. Gini, "Dispersing robots in an unknown environment", in 7th International Symposium on Distributed Autonomous Robotic Systems (DARS), June 2004.

[53] H.S. Jeon, M.-C. Ko, R. Oh and H.K. Kang, "A practical robot coverage algorithm for unknown environments", in Proceedings of the 9th Mexican international conference on Advances in artificial intelligence: Part I Pages 129-140, 2010.

[54] Tammet, T., Reilent, E., Puju, M., Puusepp, A., & Kuusik, A. (2010). Knowledge centric architecture for a robot swarm. IFAC Proceedings Volumes, 43(16), 294-299.

[55] Panda3D at https://www.panda3d.org/ *(25.04.2017)*

[56] Jgame 2D at https://sourceforge.net/projects/jgame-engine/ *(25.04.2017)*

[57] A. Tanoto, U. Rückert, "Local Navigation Strategies for Multi-Robot Exploration: From Simulation to Experimentation with Mini-Robots", Procedia Engineering, vol. 41, 2012, pp. 1197-1203

KOKKUVÕTE

Dissertatsioon tegeleb robotitega, mis suudavad kasutada RFID märgiseid nii orientiirpunktide kui omavahelise suhtluskanalina. Töö annab kõigepealt detailse ülevaate intelligentsete robotite jaoks loodud teadmus-arhitektuurist. Sellele järgneb ülevaade erinevatest töö käigus loodud katvusalgoritmidest selliste robotiparvede jaoks, mille eesmärgiks on koristamine, otsing jms. ülesanded siseruumides. Töö põhieesmärgiks ongi leida efektiivseid katvusalgoritme ja analüüsida parves osalevate robotite arvu mõju ülesande lahendamiseks kuluvale ajale. Ühe olulise tulemusena näitame, et parves osalevate robotite arvu tõstmine on ülesande lahendamise ajale oluliselt suurema mõjuga, kui roboti teadmiste täiendamine ja algoritmi optimeerimine.

Väitekirjas keskendume robotitele, millel puudub sidevõime, mis on varustatud vaid väheste, seejuures vearohkete sensoritega ning mil ülesande lahendamise alguses praktiliselt puudub teadmine ümbritsevast keskkonnast. Tüüpilised koristus- ja muruniitmis-robotid on taoliste robotite näiteks.

Konkreetne arendatav ja testitav robotitüüp on Roboswarm EU FP6 projekti käigus (kus osales ka töö autor) iRobot Roomba täiustusena arendatud RFID lugejaga robot. Algoritmide testimiseks ja võrdlemiseks kasutame töös simulatsioone. Spetsiaalselt antud ülesande jaoks ehitatud simulaatoriga on kõigepealt genereeritud peaagu ideaaltulemustele vastavad baasandmed, mida on seejärel kõrvutatud robotiparvede poolt erinevate algoritmide rakendamisel saadud tulemustega.

Appendix A

Paper A

T. Tammet, J. Vain, A. Puusepp, E. Reilent, A. Kuusik. RFID-based communications for a self-organizing robot swarm. In: Proceedings Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2008: 20-24 October 2008, Venice, Italy: (Toim.) Brueckner, Sven; Robertson, Paul; Bellur, Umesh. Los Alamitos, Calif.: IEEE Computer Society, 2008, 45 - 54.

Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems

RFID-based Communications for a Self-Organising Robot Swarm

Tanel Tammet, Jüri Vain,
Andres Puusepp, Enar Reilent
Department of Computer Science,
Tallinn University of Technology
Ehitajate tee 5, 19086 Tallinn, Estonia
tammet@staff.ttu.ee, vain@ioc.ee,
e.reilent@gmail.com, anduoma@hot.ee

Alar Kuusik
Department of Electronics,
Tallinn University of Technology
Ehitajate tee 5, 19086 Tallinn, Estonia
kalar@va.ttu.ee

Abstract

We investigate the practical questions of building a self-organising robot swarm, using the iRobot Roomba cleaning robot as an experimental platform. Our goal is to employ self-organisation for enhancing the cleaning efficiency of a Roomba swarm. The implementation uses RFID tags both for object and location-based task recognition as well as graffiti- or stigmata-style communication between robots. Easily modifiable rule systems are used for object ontologies and automatic task generation. Long-term planning and central coordination are avoided.

1 Introduction

The concept of a robot swarm denotes a large number of relatively simple physically embodied agents designed in a way that the desired collective behaviour emerges from the local interactions of agents and the interactions between the agents and the environment. The swarms are meant to perform a wide range of tasks which are infeasible to accomplish by a single robot. Their application ranges from simple cleaning tasks to exploration of large unknown areas, surveillance, rescue, coordinated weight lifting, minesweeping etc. where intervention from human operators is minimized.

The goal of a swarm mission can be considered generally as an integrated service provided by the swarm members collectively over a given period of time. Since swarms typically act in a dynamic and partially observable environment, the service requires repetitive and coordinated action by the swarm members throughout the mission.

The overall goal of our project is to develop simple and low-cost technologies for making both single robots and swarms of robots more intelligent. We use the dynamic cleaning problem [1], [7] as a testbed for the developed knowledge architecture, focusing on making swarm cleaning more efficient.

The crucial part of the project is to achieve the efficient cooperative behaviour of robots without any central coordination and planning.

That, again, requires propagation of understandable and reusable information among the robots which may be different in hardware and software. The target goal can be called "knowledge centric" architecture approach focusing on uniformal (or easily convertible) on-robot and inter-robot data management.

We use ordinary passive RFID chips for marking objects like chairs, walls, doors. This is significantly cheaper and more flexible than using cameras on robots for object recognition. The same RFID chips on objects are also used by the robots to leave messages to other robots. The solution is inspired by ants' communication using pheromone trace known as stigmery. The usage of RFID tags reduces the communication overhead related with coordination significantly [9].

We use a popular iRobot Roomba cleaning robot and attach a tiny ARM-based Gumstix computer (500 MIPS computing power) using a BusyBox 2.6 Linux distribution (without real-time capabilities) and a stock RFID reader/writer on the Roomba. The attached computer takes over control of the Roomba. While a standard Roomba is fairly simple-minded, will clean places recently cleaned and does not understand that some places should be avoided - or vice versa, cleaned often - our system adds necessary intelligence.

First, Roombas understand object descriptions and simple messages written on RFID chips by humans: like "go away", "fragile", "clean here", "this is a chair" etc. When the robot notices an RFID chip ahead, it will read its content and behave accordingly, following the configurable rules on board. The rule engine uses ontologies and allows the robot

978-0-7695-3404-6/08 $25.00 © 2008 IEEE
DOI 10.1109/SASO.2008.62

to understand, for example, that chair is a furniture and you can probably go around furniture.

Second, Roombas write their own messages on RFID chips. For example, when the robot notices an RFID tag while cleaning, it will write on the tag that it was cleaning there at that particular time. Next time when it comes near the same tag, it will not clean the place, unless enough time has passed. What is more important, when we use a whole swarm of Roombas for cleaning, all the other Roombas will also avoid cleaning on this marked up place for some time, avoiding wasted work. Similar optimizations are achievable for spreading our swarm members to different rooms, mapping the area, etc.

2 Robot control architecture

The architecture for the robot control is based on a layered multi-agent system, with agents implemented as continuosly running processes, and contains three layers (figure 1):

- The sensor-actuator access layer dedicated to communication with robot control hardware. The lowest part of robots sensor-actuator layer is executed by the iCreate onboard microcontroller. The external service time of this microcontroller was set to 20ms as shortest allowed period. Tests showed that the core agent communication solution did not add any additional mentionable response delays.

- The control layer that includes usual short-term planning and behavioural layer tasks. Merging two traditionally separate layers is reasonable due to the fact that the swarm robots (e.g. cleaning devices, room patrols) are relatively simple and the number of different behaviours is rather limited.

- The knowledge layer that targets reasoning (deriving new information from acquired data), communicating with other robots (using RFID tags) and the optional central server (using WIFI, if available).

The layered architecture is built around a fast and transparent RDF database implemented in shared memory. The RDF database realizes a core for interprocess communication of several on-board agents (processes), in particular the Main Control Agent (Central Control Process, control layer), Sensor Agent, Actuator Agent (sensor-actuator access layer), Reasoner Agent (knowledge layer). Moreover, the RDF database can manage the inter-robot communication using different functions/technologies for sending data to other robots (knowledge layer as well).

The internal knowledge architecture follows the classical blackboard model [4] In short, the agents communicate by

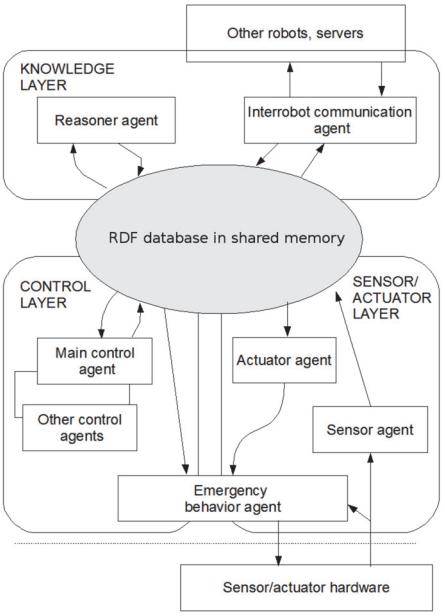

Figure 1. Robot architecture

writing data to the RDF database. The data on the RDF database is available to all agents.

The RDF database serves three roles:

- A postbox between different process agents (including external world communication).
- A fast and simple in-memory data store (circular buffer).
- A deductive database, using a rule language for rule-based generation of new facts.

Technologically the RDF database is built as a simple data store operating in shared memory. Shared memory based database approach is frequently used for low latency robot control architectures performing sufficiently well without real time OS. By our benchmarking tests performed with 500MHz 32 bit embedded ARM processor running non-real-time BusyBox 2.6 Linux distribution, the realized shared memory data store access time was in tens of microseconds.

3 Languages, common data model and the RDF database

The behaviour of the robot is primarily influenced by four players: sensors and control software, internal RDF database contents, RFID tags read, data and rule files read from the swarm server.

The swarm server collects data from the robots and influences them by sending new data and modifications to rule files in the robots.

The different players above use specialised language representations, all based on RDF. Different syntaxes stem from practical needs: for example, since RFID chips contain very little memory, we have to use a space-efficient encoding for information on RFID-s. On the other hand, communication between different servers does not require space efficiency: rather, it is preferable to use common, verbose XML-based standards.

We use the following RDF-based languages in the robot swarm system:

- Our specialised RDF encoding in RFID tags.
- Our specialised rule language for deriving and adding new data from/to the RDF database.
- Standard XML-based RDF syntax for data exchange between robots and the central server (using WIFI if available) and the central server and external systems.

All these languages/use cases share a common data model and the concrete strings for sensor and task representation (robot sensor/task language).

3.1 Common data model

The common data model is based on RDF triplets (proper data fields) to which we add two additional groups of data fields: contextual data fields and automatically generated metadata.

Proper data fields:

- Subject: id of whatever has the property.
- Property: name of the property of the subject.
- Object: value of the property.

The value field has an associated type, indicating the proper way of understanding the value. Observe that the property field typically - but not always - already determines the suitable or expected type.

In addition to basic RDF, we will always add three contextual fields to the beforementioned proper data fields of the triplet.

Contextual data fields:

- Date/time: when this fact held (in most cases same as the time of storing the data).
- Source: identifies the origin of the data (RFID nr, person id, other robot id, etc).
- Context: identifies a data group or addressee or indicates the succession of robot commands, often left empty.

Agents can enter their own contextual values to the RDF database. If no values are given by the agent, the default values (current date/time, robot id, empty context) are entered automatically.

Automatically generated metadata:

- Id: robot-unique id of the data row, auto-increased.
- Timestamp: date/time of storage.

Automatically generated metadata is present only in the RDF database, and not in the other data formats/languages. Agents cannot enter their own values at will. These two fields are important for efficient and convenient management of the data, and are used for example, by the reasoner.

Instead of using additional contextual and metadata fields we could have chosen to use reification of RDF triples to store the same information. However, this would have cumbersome and inefficient both in the internal RDF database used by agents inside the robot, and even more so in the data representation inside RFID chips, as described in the following chapters.

For data exchange between different swarms and external applications we will use the reified form of the contextual data fields, represented in the common XML syntax of RDF.

3.2 RDF database

The RDF database is implemented in the Gumstix computer on the robot as a library for storing and reading information to/from shared memory. Agents in the robot use a simple C API for writing, reading and searching data from the RDF database. Special RDF query languages are not used.

Strings in the RDF database are pointed to from the data fields: they are kept in a separate table, guaranteeing uniqueness: there is always only one copy of each string.

The data rows are organised as a circular list. The last data element will disappear when a new one is added. However, there are exceptions to this order: data items deemed critical are kept longer.

Although the data store should be normally seen as a mid-term memory, containing tens of thousands of rows (old data is thrown away), it is easy to use it as a post-box between different agents onboard: just put the name of the addressee agent in the context field and program the addressee agent to look for the rows with her name, process them and then delete them.

An agent X may also read "messages" intended to an agent Y, but should under normal circumstances ignore these "messages": it should look for data rows with either no addressee at all or an addressee with the name X.

4 Data encoding on RFID tags

The roboswarm architecture requires recognising external objects/locations, reading location-specific messages/instructions from humans and reading/writing location-specific messages from/for other robots.

All these three tasks use RFID tags at different locations. The simplest types of RFID tags contain only the RFID id. However, we have been using RFID tags with a small internal memory: both human operators and robots can write information to the tags. We use the tags as information-carrying graffiti.

A human user is expected to write to a tag information like "this tag is located on a chair", "this tag has coordinates X and Y", "there is a tag at direction R at distance 5 meters", "keep away from here" etc.

A robot N is expected to write to a tag information like "N was here at 10.06.2007 at 15.10", "N did brush the surroundings at 17.10" etc.

4.1 Kinds of data on RFID chips: conceptual example

The following categorization gives a clearer picture of kinds of data to be written on RFID:

- Present by default on all RFID tags: built-in RFID id number, up to 12 bytes.
- Control information written to the tag by a human user:
  - Stop immediately.
  - Keep away from here.
  - Turn to direction X and move N meters.
  - Do not clean here.
- General information about objects:
  - What kind of object: wall, bed, chair, robot nr X.
  - This place needs cleaning very often.
  - Danger in direction X distance Y.
  - Some object (lift,docking,...) in direction X distance Y.
  - Robot nr X was here at time T and cleaned / could not clean / did not want to clean.
- Localisation of a tag, either:
  - Global, for example gps coordinates.
  - Local, relative to a given base vector: direction and distance.
  - Information about other tags in the neighbourhood: direction and distance.
  - Additional useful information: path to door, path to charger.
- Information about a robot: tag glued on a robot
  - I am a robot.
  - I am a robot nr X.
  - Kind/capabilities of a robot: simple cleaner / complex control robot.

We are using local coordinate vectors and special methods and algorithms for coordinate vector markup and finding. These methods are not covered in this paper.

4.2 Data encoding principles for RFID tags

RFID tags contain relatively little memory: we are currently using tags with 256 bytes. Reading of RFID data over wireless may be prone to error. Hence:

- The data format has to be extremely compact.
- Old data has to be regularly overwritten.
- There must be a way to indicate that some parts of data should not be overwritten by robots.
- We need a control sum for data blocks.

We use 32 bytes for encoding one data block, hence we can put 8 data blocks on our RFID tags.

Data encoded on tags must be easily understood both by robot software and external applications: software used by humans to read/write data to tags, agents different from the roboswarm components.

Hence we provide a simple mapping from RFID data to both robot internal data format and the generic RDF format for data.

All data items written to the RFID tags are essentially data rows with several predefined fields. All fields may contain different data items: strings, integers, floats. Hence the RFID data store is similar to a single database table.

Data is written, read and deleted one full row at a time: while changing stored rows is technically possible, we do not recommend doing that: it is better to add a new full row, and if necessary, delete the old row(s).

Conceptually, each data row corresponds to several RDF triplets. Standard RDF triplets contain the subject, property name and value fields, like this:

```
[12, performingaction, cleaning]
[12, notperformingaction, cleaning]
[15, lookingfortag , 244]
[15, notfoundtag , 244]
```

The subject field is normally filled with an id of an object which has the property with the value indicated. The value field may be filled either with a direct value or an id of some object (for example, a tag).

Using triplets will inevitably mean that recording one data item may require several triplets to be written. Suppose that a robot wants to write the message "Robot nr 15 has been here at 14.20 on 28. January looking for tag nr 244 and did not find a tag while here." on the RFID tag.

Using standard triplet format this would translate to the following triplet set:

```
[15, washereattime,
    14.20 on 28 January]
[15, waslookingfortag, 244]
[15, didnotfindtag, 244]
```

For RFID tags we always add timestamp, context and source (agent) id contextual fields. In our sextet data model the information would contain the following fields:

```
[15, 14.20 on 28 Jan, general,
  15,washereattime,14.20 on 28 Jan]
[15, 14.20 on 28 Jan, general,
  15, waslookingfortag, 244]
[15, 14.20 on 28 Jan, general,
  15, didnotfindtag, 244]
```

Data field contents are either direct (integers, RFID chip id-s) or indirect (identifiers of long strings):

- Direct values are put on the tag as-is.
- Long strings are not kept on the RFID, since we do not have enough space: we use a string number in a global string table instead.

The direct values are either 4 or 12 bytes long, depending on the type of a data block (see later sections). A direct value may either indicate one concrete measure (say, distance or time), contain a short string (up to 4 or 12 characters) or encode several short values, for example, a coordinate.

As the standard RDF format requires, the robot RDF database uses strings for identifying subjects and property names. RFID tags do not have enough space for long strings.

Hence we assume that a roboswarm has a common string table of predefined strings, where each string has a concrete number, common for all robots and RFID chips. Robots can certainly use more and dynamically created strings, but these strings will not be encodable on RFID chips.

Property names (but normally not property values) have a namespace prefix. We will commonly use http://www.roboswarm.eu/lang as a namespace for property names in the roboswarm. However, other namespaces may be used as well. For example, a full name string of a "washereattime" property would be http://www.roboswarm.eu/lang#washereattime and this could be encoded as, say, number 135 in a common string table.

Identifiers for robots and humans are swarm-specific. We will use namespaces for these as well, however. The default swarm namespace for our experiments is http://www.roboswarm.eu/swarm. Concrete swarms may use different namespaces.

The roboswarm environment may potentially contain a huge number of different RFID tags. Old tags may be replaced, new ones may be glued on objects at any time. It would be impractical to assume that the robot software

has predefined knowledge of all tags in the environment. Hence the RFID id on the tags is used "as is", without encoding it via a separate string table (see the next section). The robot software components will identify RFID tags by strings with the http://www.roboswarm.eu/RFID namespace followed by the hexadecimal encoding of the RFID id number.

As said before, the RFID data blocks contain both direct values (date/time, measures, coordinates, RFID id numbers, short strings) and numbers of strings in a common string table (external to chips)

The numbers in a string table start from number 0 and continue with numbers 1, 2 etc. We use 2 bytes for string table numbers, even if the field containg the string number is longer.

The global string table is loaded into the robot and has to be the same for the whole swarm. It is necessary only for coding and decoding data for the RFID tags.

4.3 RFID tag id numbers and special strings

RFID tags carry an id. The id size may vary. However, there are several widely used standards for product encoding, and most RFID tags are expected to conform to these standards:

- UPC (universal product code): 12 digit numbers identifiying product type, commonly used on bar codes.
- 64-bit EPC (electronic product code): 64 bit code identifying concrete items, forward compatible with a 96-bit version.
- 96-bit EPC (electronic product code): 96 bit code capable of identifying concrete items.

We use direct 96-bit EPC-s to identify RFID tags. Inside the robot software the RFID tags numbers are not used directly (as-is). Instead, they are encoded to identifier strings (uris) with the following algorithm: the initial part of the string is always a namespace prefix http://www.roboswarm.eu/rfid# and the following part of the string is formed from the RFID id number (of whichever length) by converting the number to a lower-case hex string in a conventional manner.

It is very common for a tag to contain information about its own location or the object it is glued to. In order to avoid putting the full 96-bit EPC into the subject id field, our string table contains contain a special string: http://www.roboswarm.eu/lang#me stands for the EPC of the RFID chip containing this data item.

4.4 Encoding details: data fields

We use 32 bytes for one data block (a sextet in our data model). We have two types of blocks. First type contains a short, 4-byte subject field and a long, 12-byte (96 bit) object field. Second type contains a long, 12-byte (96 bit) subject id field containing EPC and a short 4-byte object field.

Otherwise the structure and meaning of the data blocks is identical for both types:

- Blocktype 1 byte: contains block type nr, either 1 or 2.
- Agent 2 bytes: number of the agent string (robot, human, ...) writing data.
- Datetime 4 bytes: datetime of writing, according to the robot clock (up to one second), unix format.
- Context 2 bytes: number of the context string (adressee, data group, etc: often ignored)
- Subject (blocktype 1) or object (blocktype 2) 4 bytes: numeric, datetime, short string or string number in the string table.
- Property 2 bytes: number of the property name string in the name string table.
- Object (blocktype 1) or subject (blocktype 2) 12 bytes: epc, numeric, datetime, short string or string number in the string table.
- Reserved 2 bytes.
- Object type 2 bytes: number in the string table indicating type of value (int, short string, some structure etc).
- Checksum: 1 byte.

We use xml schema datatype names as value type indicator strings, extended by our own specific datatype names.

We use the simplest checksum algorithm: adding bytes 0...31 one after another and keeping the lowest byte of the sum after each addition.

Multibyte integers and floats have to follow the high-endian (intel standard) byte order. Direct short strings start from the leftmost byte and should be terminated with a zero byte. In case there is no zero byte, the data reader has to append the zero byte to the direct string (4 or 12 bytes) read.

In normal cases it is recommended to use the first type of data blocks with a long value field. The second type is suited for cases where we want to write information about a specific RFID chip, different from the current chip (in the latter case we should use the special 'me' string http://www.roboswarm.eu/lang#me).

4.5 Reading and writing data

In case a robot writes data to an RFID tag, it will normally have to delete some old data to make room for new data to be written. It will also have to take care that important data is not deleted. The robot follows these principles:

It will always delete the oldest data block which is allowed to be deleted. By default all data blocks written by humans have to be preserved. The internal datastore of a robot contains information about kinds of writers (block contains the writer id).

5 Antennas and other practical aspects of RFID reading and writing

Before writing or reading data, the robot will have to understand that an RFID tag is in a reading or writing distance. It will then start reading and - sometimes - also writing the RFID.

We have conducted a number of RFID reading and writing experiments with an iRobot Create equipped with a Gumstix Verdex microcomputer and the Skyetek M9 OEM RFID reader card, operating frequency was 865MHz, output power 27dBm.

Achieved dependable access ranges for ISO 18000-6B and 6C tags have been between 0.7 and 1.2 meters, depending on tag orientation and various other factors.

The practical issue of detecting a tag depends on many factors, quite significantly also on the shape of the tag's antenna and the orientation of tag in the robot's RF field. Therefore one antenna should be omnidirectional (e.g. circular polarization antenna).

However, two switched linearly polarized reader antennas may be used giving additional direction information. That idea will be evaluated further.

On the figure 2 we have an iRobot Create equipped with the 6 dBi Yagi antenna. This antenna appeared to be too sensitive directionally: it was hard to notice tags not directly in front of the robot.

The figure 3 demonstrates a 13dBi spiral antenna designed during the project. While detecting tags from a somewhat longer distance than the Yagi antenna, it had analogous problems with directionality.

The best choice so far has been the patch antenna on the figure 4. The small loss in tag detection distance is compensated by the significantly wider area of coverage, enabling the robot to detect tags not directly in front of it.

In our experiments it has been somewhat easier to detect the RFID and read its id number than to read full RFID memory. Hence, when a tag is detected somehwere in front of the robot, we keep driving for a short while to get to the practical reading/writing distance.

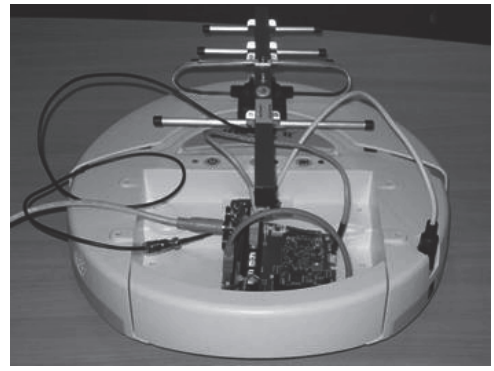

Figure 2. iRobot Create with a 6 dBi Yagi antenna

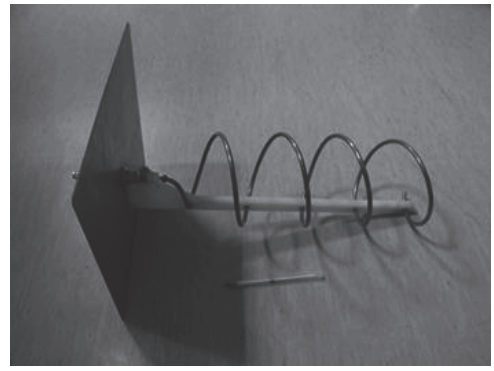

Figure 3. 13dBi spiral antenna

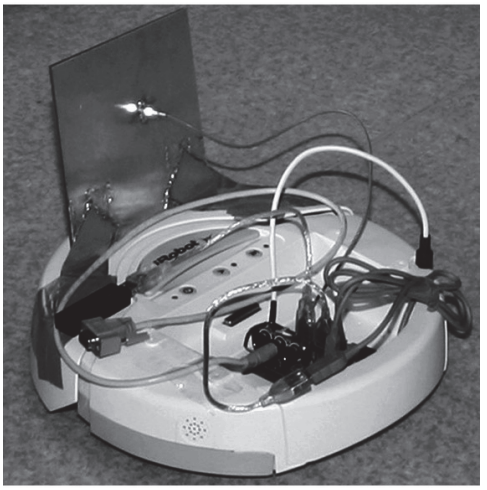

Figure 4. iRobot Create with a 6.5dBi patch antenna

Another important aspect is the frequency of scanning for the RFID tags: since tag reading and scanning draws significant amount of power, high frequency of scanning drains the Roomba internal battery faster than would be practically feasible.

6 Rule engine and the rule language

The central command agent uses the RDF database contents as grounds for deciding whether the robot is doing ok, is in trouble or what to do next.

Programming the robot to act correctly for each case is hard. We are using a rule engine to perform specific checks on data and make decisions based on the given set of rules. Rules are written in a prolog-like syntax and stored initially as a plain-text rule file in the robots file system. The rule engine takes all the input data from RDF database and stores derived facts again into the RDF database. Other agents do not use the rule engine directly, they just read the output from the in-memory database.

In other words, the rule engine is not used for answering queries, but for automatically deriving new facts added to the RDF database. Obviously, the set of rules has to be consistent and should not contain too many or too complex rules. We are using the special modification of the Gandalf first order resolution-based theorem prover [8] as a rule engine.

The rule engine is fired automatically by the rule engine process after each pre-determined interval. Using a relatively simple set of rules we manage to keep the interval under one second: during this time the rule engine performs all possible derivations stemming from the facts added to the RDF database after the last iteration.

The rule system is used for two main kinds of tasks:

- Deriving generalisations (chair is furniture) from ontology rules.
- Deriving commands and subcommands, depending on the situation.

We are not using OWL directly as an ontology language. Instead, the central server contains a component for converting given OWL files to the rule language syntax. These rule files are then preloaded to robots and updated over WIFI, if available.

The # mark in the following examples stands for the full default namespace http://www.roboswarm.eu/lang. Although we use the syntax based on Prolog, the derivation algorithm is a specialized version of the bottom-up resolution algorithm as often used in automated theorem provers, starting from the facts and deriving new facts/lemmas. The derivation process does not attempt to solve a posed "query", just to derive new facts. Hence the language does not contain extralogical predicates like cut and closed-world not.

Two simple ontology rules, indicating that anything attached to a glass object is attached to a fragile object, and anything attached to a fragile object is attached to a dangerous object:

```
#attachedTo(X,fragile) :-
    #attachedTo(X,glass).
#attachedTo(X,dangerous) :-
    #attachedTo(X,fragile).
```

Sample rules for firing executable commands with argument 0 and high priority 1 ("me" is a special macro constant indicating robot itself):

```
command(escape,0,1) :-
   #attachedTo(X,dangerous).
command(clean,0,1) :-
   "found-tag"(me,tag2).
```

The next rule derives information about a need to keep away for 10 minutes from the given location. "now" is a special macro constant indicating current time. There should be further rules given to make the robot actually use this information:

```
#keepaway(Loc,600) :-
    #roombusy(Loc,Time),
    lesstime(now,Time).
```

7 Robot sensor/task language

The sensor/task language does not have a separate syntax. Rather, it is a collection of strings with predetermined meanings, designed for two goals: storing robot sensor data and giving commands to the robot (clean here, drive away, find a certain item).

The sensor/task language uses the RDF database for storing both tasks (commands) and sensor information. In other words, all the commands to the robot and sensor information items are stored in the RDF database as ordinary data objects with a special meaning to the robot control process.

The sensor/task language contains several different categories of object strings:

- Task data items: used for giving general kinds of commands to the robot (clean here, drive away, look for object, exit room etc).
- Sensor/status values: information added by sensor processes or derived using rules.
- Generally useful special values (me, now etc).

A typical task data item contains the following fields:

- Subject: a string indicating actual command, like "escape" or "clean".
- Property: special predicate "command".
- Object: used for tasks or commands requiring extra information (like how far to drive). In case the command requires several information fields, these are encoded into a single value.
- Context: used to indicate both the succession of commands and nesting of commands.

Say we have two main commands c_1 and c_2 which should be performed in succession. Command c_1 has two subcommands s_1 and s_2. The command c_1 will be automatically replaced by s_1 and s_2 by the corresponding rule. These three commands will then have the following context sequences:

- s_1: [1,1]
- s_2: [1,2]
- c_2: [2]

Tasks are represented as data items the RDF database. The robot control process starts fullfilling the task as soon as it is seen in the database. The same process should mark this task as being currently fullfilled. In case of conflict or impossibility the robot control process should choose the action itself.

There can be complex tasks that consist of number of smaller subtasks. These kinds of tasks are presented using rules. Suppose somebody adds a task into the in-memory database. After a while the rule engine will take this task, find the matching rules and add derived subtasks into the database. The derived subtasks could again match some rules, in which case they will also be derived and added to database.

The sequence of tasks is encoded in the context field of data item. Task and subtasks should be seen as an ordered forest of trees with branches corresponding to subtasks.

Tasks are loosely grouped into four levels starting from high-level down to low-level tasks. A high-level task is an abstract representation of what should the robot do: for example, clean a room for whole day. A typical low-level task would be turning the robot 50 degrees.

Long-term activity - normally defined by human.

While fullfilling this type of task, the robot can also fill subtasks like recharging, exiting room etc. These tasks do not restrict robot from doing subtasks.

For example: property "shalldocleaning", object time in seconds until which activity holds. Robot should be in the general cleaning mode: driving around and cleaning.

Short-term activity - normally given by rules or control process, but can also be defined by human.

Fullfilling this kind of task may consist of several small tasks like turning and moving some distances.

The associated rules are expected to generate atomic commands, which are put into in-memory database waiting to be fullfilled one after another. Only one task is fullfilled at time. Here the time information is not relevant, rather, the succession should be followed.

Atomic activities - basically procedural, normally generated by rules or the control process.

For example:

- `command(turn,Degrees,Context)`: robot should turn the indicated amount of degrees. "turn" here is a constant string indicating the actual preprogrammed procedure the robot should follow. Context should contain a task order/priority indicator as explained before.
- `command(move,Centimeters,Context)`: robot should move (with 'normal' speed) the given amount of centimeters.

Direct activities - these kinds of tasks can be directly delegated to the robot API for execution. The Roomba robot has very few such direct commands available: the most imporant command is "drive with speed X and radius N".

8 Related work

The SHAGE/AlchemistJ framework [5] can be mentioned as one solution for robot knowledge exchange using data repositories and component brokers.

Using high level data representation is a trend of modern robotics. For example, XML based data coding has been used on robots [5]. However, besides the benefits of universal high level representation the XML encoding requires additional conversions between exchange and machine control domains.

Conventional XML based RDF format is used for time uncritical inter-robot or server communication, the description can be found in [2]. A special, compact RDF format is used for storing real-time algorithms of robot operation.

The RFID technologies with goals similar to our experiments have been investigated in [9]. The authors use RFIDs to allow an autonomous mobile robot to acquire a target and approach it for task execution. The robot is equipped with a dual directional antenna that communicates with controllable RF transponders.

See also [3], [6], [10].

9 Conclusions and future work

We have designed the architecture for the robot swarm and started actual implementation and testing with real tags and robots. So far we have successfully implemented both the robot hardware and software, including the RDF database, RFID and rule engine usage as described in the paper. The experiments have been encouraging when we consider processing power and reaction times: the tiny onboard Gumstix computer manages to run the described agents, use the RFID chips, RDF database and the rule engine without slowing down the robot reactions. On the other hand, detecting, reading and writing RFID tags requires considerable care when selecting tag types, antennas and the scanning frequency. A usable solution has been worked out, but further optimisations and improvements are needed.

We have also started to implement components of the central server and open connections to other robot swarms and external software. However, this work is still ongoing, specific details are being filled in and it is too early to report experiments from the high-level perspective.

Acknowledgements. The work was supported by FP6 ICT "ROBOSWARM" project, see http://www.roboswarm.eu.

References

[1] Y. Altshuler Y, A.M. Bruckstein, I.A. Wagner: Swarm Robotics for a Dynamic Cleaning Problem. In "IEEE Swarm Intelligence Symposium", pages 209–216, 2005.

[2] E. Ardizzone, A. Chella, I. Macaluco, D. Peri: A Lightweight software architecture for robot navigation and visual logging through environmental landmarks recognition, in Proc of International Conference on Parallel Processing Workshops, ICPPW 2006.

[3] A. Elci, B. Rahnama: Human-Robot Interactive Communication Using Semantic Web Tech. in Design and Implementation of Collaboratively Working Robots, RO-MAN 2007. The 16th IEEE International Symposium, pages 273–278 (2007).

[4] B. Hayes-Roth: A blackboard architecture for control. Artificial Intelligence, 26(3): pages 251–321, July 1985.

[5] S. Lee, I.H. Suh and M.S. Kim (Eds): Recent Progress in Robotics, LNCIS 370, Springer, pages 385–397, 2008.

[6] C. Stanton, M.-A. Williams: Grounding Robot Sensory and Symbolic Information Using the Semantic Web in RoboCup 2003: Robot Soccer World Cup VII, Springer LNCS 3020/2004, pages 757-764, 2004.

[7] T. Tammet, J. Vain, A. Kuusik: "Using RFID tags for robot swarm cooperation". WSEAS Transactions on Systems, 5(5), pages 1121–1128, 2006.

[8] T. Tammet: Gandalf. Journal of Automated Reasoning vol 18 No 2, pages 199–204, 1997.

[9] V. A. Ziparo, A. Kleiner, B. Nebel, D. Nardi: RFID-Based Exploration for Large Robot Teams. In Proc. IEEE International Conference on Robotics and Automation, pages 4606–4613, 2007.

[10] L. Vasiliu, B. Sakpota, K. Hong-Gee: A semantic Web services driven application on humanoid robots. in the Second International Workshop on Collaborative Computing, Integration, and Assurance. SEUS 2006/WCCIA 2006. The Fourth IEEE Workshop, 2006.

Paper B

T. Tammet, E. Reilent, M.Puju, A. Puusepp, A. Kuusik, A. Knowledge centric architecture for a robot swarm. In: 7th IFAC Symposium on Intelligent Autonomous Vehicles (2010). IFAC-PapersOnLine, 2010, (Intelligent Autonomous Vehicles; 7/1). 2010.

Knowledge Centric Architecture for a Robot Swarm

Tanel Tammet, Enar Reilent, Madis Puju, Andres Puusepp [*]
Alar Kuusik [**]

[*] *Department of Computer Science, Tallinn University of Technology Ehitajate tee 5, 19086 Tallinn, Estonia (e-mail: tammet@staff.ttu.ee, e.reilent@gmail.com, pudismaju@gmail.com, anduoma@hot.ee).*
[**] *Department of Electronics, Tallinn University of Technology Ehitajate tee 5, 19086 Tallinn, Estonia (e-mail: kalar@va.ttu.ee).*

Abstract: We have built and tested a knowledge centric system for a robot swarm. Our implementation enhances iRobot Roomba cleaning robots with a tiny linux computer, RFID tag reader/writer and optionally a WIFI card. Robots use the RFID tags for object recognition and message passing. The knowledge architecture of the system is inspired by semantic web principles, spanning over several layers: RFID tags on objects, process interaction in a single robot via a main memory datastore and a rule system, central database for a swarm. The communication components of the system have been already ported to the larger Pioneer and Mugiro robots via the Player middleware. The paper presents our solutions to the knowledge management and communication problems stemming from the robotics issues and demonstrates feasibility of using the semantic web principles in the robotics domain.

1. INTRODUCTION

The overall goal of the project is to develop simple and low-cost technologies for making both single robots and swarms of robots more intelligent. We use the dynamic cleaning problem Altshuler et al. (2005), Tammet et al. (2006) as a testbed for the developed knowledge architecture, focusing on making swarm cleaning more efficient.

Our goal requires propagation of understandable and reusable information among the robots which may be different in hardware and software. There are several well-known frameworks (like Player and Orca) addressing mostly low level data management in robots, however, robotics industry expresses need for an environment supporting hardware-independent data presentation and exchange among robots.

The target goal of this paper can be called a "knowledge centric" architecture, focusing on uniform (or easily convertible) on-robot and inter-robot data management Tammet et al. (2008), which is achieved by using semantic web principles, prolog-like rules, and first-order logic. The modular system of independent asynchronous software components for reactive control of cleaning robots was created as a proof of concept.

We use the popular iRobot Roomba cleaning robot and attach a tiny ARM-based Gumstix computer (500 MIPS) using a BusyBox 2.6 Linux distribution (without real-time capabilities) and a stock RFID reader/writer on the Roomba. The attached computer takes over control of the Roomba. While the standard Roomba is fairly simple-minded, will clean places recently cleaned and does not understand that some places should be avoided - or vice versa, cleaned often - our system adds necessary intelligence.

We use ordinary passive RFID chips for marking objects like chairs, walls, doors, as well as locations in the environment. This is significantly cheaper and more flexible than using cameras on robots for object recognition. The same RFID chips on objects are also used by the robots to leave messages to other robots. The solution is inspired by ants' communication using pheromone trace known as stigmery. The usage of RFID tags reduces the communication overhead related with coordination Ziparo et al. (2007).

The main components of the architecture have been already ported by the industrial project partner Fatronik to two different robots (Pioneer and Mugiro) via the Player middleware.

2. ROBOT KNOWLEDGE ARCHITECTURE

The architecture for the robot control is based on a layered multi-agent system, with agents implemented as continuosly running processes. Three layers can be brought out:

- The sensor-actuator access layer dedicated to communication with the robot control hardware. The lowest part of robots sensor-actuator layer is executed by the Roomba onboard microcontroller.
- The control layer consists of dispatcher process which executes behavioral tasks in our context called binaries.
- The knowledge layer that targets reasoning (deriving new information from acquired data), communicating with other robots (using RFID tags) and the optional central server (using WIFI, if available).

The layered architecture is built around a fast and transparent RDF inspired datastore implemented in shared memory. This kind of approach is frequently used for low

latency robot control architectures performing sufficiently well without using a real time OS. The internal knowledge architecture follows the classical blackboard model Hayes-Roth (1985). In short, the agents communicate by writing data to the memory datastore and every agent can access all data inserted to datastore.
The memory datastore serves three roles:

- A postbox between different process agents (including external world communication).
- A fast and simple in-memory data store (circular buffer).
- A deductive database, using a rule language for rule-based generation of new facts.

3. COMMON DATA MODEL AND LANGUAGES

The behavior of the robot is primarily influenced by four players:

- sensors and control software
- internal memory datastore contents
- RFID tags read
- binary executables plus data and rule files read from the swarm server.

The swarm server collects data from the robots and influences them by sending new data back to the robot datastore, updating rule files in the robots and sending new binary executables to the robots.

The different players above use specialised language representations, all based on RDF triples plus metadata: the combination which we will call RDFm. Different syntaxes stem from practical needs. For example, since RFID chips contain very little memory, we have to use a space-efficient encoding for information on RFID-s. On the other hand, communication between different servers does not require space efficiency: rather, it is preferrable to use common, verbose XML-based standards.

We use the following languages in the robot swarm system:

- RDFm encoding in RFID tags.
- Our specialised rule language for deriving new information based on data in memory datastore.
- Both a CSV-based syntax and an XML-based RDF syntax for data exchange between robots and the central server (using WIFI if available) and the central server and external systems.

All these languages share a common data model and the concrete predefined strings for adressing data to agents.

3.1 Common data model

The common data model is inspired by RDF triples to which we add two additional groups of data fields (metadata): contextual data fields and automatically generated metadata.

Data fields taken from RDF triple:

- Subject: id of whatever has the property.
- Property: name of the property of the subject.
- Value: value of the property.

The value field has an associated type, indicating the proper way of understanding the value. Observe that the property field typically - but not always - already determines the suitable or expected type.

In addition to basic RDF, we will always add three contextual metadata fields to the beforementioned proper data fields of the triplet.

Contextual metadata fields:

- Date/time: when this fact held (in most cases same as the time of storing the data).
- Source: identifies the origin of the data (RFID nr, person id, other robot id, agents, etc).
- Context: usually identifies addressee or data group, can also indicate the succession of robot commands.

Agents can enter their own contextual values to the memory datastore. If no values are given by the agent, the default values (current date/time, robot id, empty context) are entered automatically.

Automatically generated metadata:

- Id: unique data row nr for a robot, auto-increased.
- Timestamp: date/time of storage.

Automatically generated metadata is present only in the memory datastore, and not in the other data languages. Agents cannot enter their own values at will. These two fields are important for efficient and convenient management of the data and are used for example, by the reasoner and dispatcher processes.

Instead of using additional contextual and metadata fields we could have chosen to use reification of RDF triples to store the same information. However, this would have been cumbersome and inefficient both for the internal memory datastore used by the agents inside the robot, and even more so for the data representation inside RFID chips, as described in the following chapters.

4. MEMORY DATASTORE

The memory datastore is implemented in the Gumstix computer on the robot as a library for storing and reading information to/from shared memory. Agents in the robot use only a simple C API for writing, reading and searching data from the memory datastore. Agents see datastore as one table based on RDFm format. Using shared memory instead of classical socket-based data exchange increases flexibility as potential consumers of data do not have to be known in advance.

Strings in the memory datastore are pointed to from the data fields: they are kept in a separate table, guaranteeing uniqueness: there is always only one copy of each string.

The data rows are organised as a circular list. The last data element will disappear when a new one is added. However, there are exceptions to this order: data items deemed critical are kept longer.

Locking is implemented using semaphores and is row-based. Reading operations do not lock anything. When a row is being written it is invisible for all the concurrent reads.

Writing one row on Gumstix platform takes about 0.14 ms while looping over 2000 rows takes approximately 4.8 ms, which is acceptable for our needs.

Although the data store should be normally seen as a mid-term memory, containing thousands of rows (old data is thrown away), it is easy to use it as a postbox between different agents onboard: just put the name of the addressee agent in the context field and program the addressee agent to look for the rows with its name, process them and then delete them.

5. DATA ENCODING ON THE RFID TAGS

The roboswarm architecture requires recognising external objects/locations, reading location-specific messages/instructions from humans and reading/writing location specific messages from/for other robots. All these three tasks use RFID tags at different locations. Exerting RFID tags for coordination purposes increases swarm size scalability.

The simplest types of RFID tags contain only the RFID id. However, we have been using RFID tags with a small internal memory: both human operators and robots can write information to the tags. We use the tags as information-carrying graffiti, in other words, tiny data stores distributed all over the environment.

A human user is expected to write to a tag information like "this tag is located on a chair", "this tag has coordinates X and Y", "there is a tag at direction R at distance 5 meters", "keep away from here" etc.

A robot N is expected to write to a tag information like "N brushed here for 10 minutes on 10.06.2007 at 15.10", "N left this place for the living room" etc.

Data encoded on tags must be easily understood both by the robot software and the external applications: software used by humans to read and write data to tags as well as agents outside the roboswarm.

All data items written to the RFID tags are essentially data rows with several predefined fields which may contain strings, integers or floats. The RFID data store is very similar to the robot's memory datastore. Data is read, written and deleted one full row at a time, updating is allowed only on the value field, the timestamp field and the source field are updated at the same time automatically.

For example, one tag might carry the following data:

subject	property	value	source	context
me	inRoom	kitchen	human	static
kitchen	hasPriority	7	human	static
kitchen	dutyStatus	cIP	robot3	work
robot2	wentInDirection	270	robot2	work

* cIP - cleaningInProgress

where the "static" context is used for data describing the surrounding enviroment and the "work" messages are written to the tag by swarm members to improve cooperation while performing tasks.

Data field contents are either direct (integers, RFID chip ids) or indirect (long strings). Direct values are put on the chip as-is. Long strings are not kept on the RFID, since we do not have enough space: we use a string number in a global string table instead. This global string table is loaded into the robot and has to be the same for the whole swarm. It is necessary only for coding and decoding data for the RFID tags. We use 2 bytes for the string table numbers.

5.1 Reading and writing RFID tags

RFID tags carry a built-in id. The id size may vary. However, there are several widely used standards for product encoding, and most RFID tags are expected to conform to these standards.

We use direct 96-bit EPC-s to identify RFID tags. It is very common for a tag to contain information about its own location or the object it is glued to. While referring to itself we use the string "me" in the data row instead of the real EPC value.

In case a robot writes data to an RFID tag, it will normally have to delete some old data to make room for new data to be written. It will also have to take care that important data is not deleted. The robot follows these principles: It will always delete the oldest data block which is allowed to be deleted. By default all data blocks written by humans and the blocks with the context "static" have to be preserved.

6. KNOWLEDGE BASED CONTROL SYSTEM

The robot control and decision making responsibilities in our system are divided between several different agents.

The control system architecture has two layers: the supporting framework and the user applications built upon the framework.

The crucial element in our system is the memory datastore. All the other subsystems are meant to be built around the datastore and interact with each other only via the datastore. As a consequence, all data - sensor readings, decisions, commands, reports, etc - ever created by some agent will be available to all the agents.

Gathering all kinds of knowledge into one place and representing it in the same format encourages us to attach a general data processing mechanism - the prover - to the memory datastore. The prover is used to derive new data items based on the existing data in the memory datastore and predefined logic rules.

The supporting components like the prover, the communication process and the low level hardware access software (sensor process, actuator process) run all the time as separate never-ending processes. However, the control-specific modules are not required to run all the time. Therefore, in addition to the prover the control support framework uses a special dispatcher process with the task to launch other agent processes during runtime.

The implementation of "the real" control software is very flexible. The algorithm can be divided to various modules and rules. For several subtasks we have created dedicated modules (binary executables) which are relatively small and simple. A binary executable can perform an atomic task, for example play a sound or calculate an average, or comprise a set of actions to achieve a complex goal, like

performing a localization procedure at the reference point (RFID tag).

Rules have the role of linking binaries together and making decisions during runtime. For example: the agent A stores the fact B into the datastore. The prover derives (according to the given rule files) the new fact C from the fact B, where C is a command to start the agent D. When the dispatcher sees the derived fact C in the datastore, it launches the demanded agent D, which in turn can change the contents of the datastore.

7. RULE ENGINE AND THE RULE LANGUAGE

The control system uses the memory datastore contents as grounds for deciding whether the robot is doing ok, is in trouble or what to do next.

Programming the robot to act correctly for each case is hard. We are using a rule engine to perform specific checks on data and make decisions based on the given set of rules. Rules are written in a prolog-like syntax and stored initially as a plain-text rule file in the file system of the robot. The rule engine takes all the input data from memory datastore and inserts derived facts into the memory datastore. Other agents do not use the rule engine directly, they just read the output from the datastore.

In other words, the rule engine is not used for answering queries, but for automatically deriving new facts added to the memory datastore. Obviously, the set of rules has to be consistent and should not contain too many or too complex rules. We are using the special modification of the Gandalf first order resolution-based theorem prover Tammet (1997) as a rule engine.

The rule engine is fired automatically by the rule engine process after each pre-determined interval. In the following we will call this "firing" process the *derivation session*. Using a relatively simple set of rules we manage to keep the derivation session interval under one second: during this time the rule engine performs all possible derivations stemming from the facts added to the memory datastore after the last iteration.

The rule system is used for two main kinds of tasks:

- Deriving generalisations (chair is furniture) from rules.
- Deriving commands depending on the situation.

While the rule system is working, it uses memory datastore as the main source of facts.

For example, if we have a rule

```
attachedTo(X, furniture) :-
  attachedTo(X, chair).
```

and the following facts in the memory datastore

subject	property	value	source	context
tag4	attachedTo	chair	RFID	null

then the rule body $attachedTo(X, chair)$ will match the datastore row and the rule will generate the new fact and add it to the memory datastore as follows:

subject	property	value	source	context
tag4	attachedTo	furniture	wGandalf	null

All the words in the rules starting with uppercase are variables. In our example X is a variable.

The following example demonstrates a simple session of robot rule usage.

```
handleTask(me, Task) :-
  state(me, stateIdle),
  receivedTask(N, Task),
  myNameIs(me, N).

state(me, stateWorking) :-
  handleTask(me, T).

startMode(me, cleaningMode) :-
  handleTask(me, clean).

startMode(me, patrollingMode) :-
  handleTask(me, patrol).

state(me, stateIdle) :-
  state(me, stateWorking),
  status(currentTask,finished).
```

We start the rule system and then add the following fact to the datastore:

subject	property	value	source	context
me	state	stateIdle	init	wGandalf
me	myNameIs	robot3	init	wGandalf
robot3	receivedTask	clean	init	wGandalf

The rule system will automatically derive and add these facts to the datastore:

subject	property	value	source	context
me	handleTask	clean	wGandalf	null
me	startMode	cleaningMode	wGandalf	null

When we later add the fact

subject	property	value	source	context
currentTask	status	finished	cleaningAgent	wGandalf

the rule system will automatically derive and add this fact to the datastore:

subject	property	value	source	context
me	state	stateIdle	wGandalf	null

The rule engine uses both the main memory database and a temporary storage area which is cleaned up after each derivation session, typically after every second.

During the derivation process a large set of new facts and clauses (temporary rules) is derived. Most of them are stored in the temporary area and are not accessible to other processes in the robot. Only positive singleton facts without variables (ground unit clauses), not containing nested terms and having a suitable number of arguments are stored in the shared database available to all the processes.

Each rule engine derivation session starts with reading and parsing the rule file and adding all the read rules and

facts into the temporary space. Hence the rule file can be changed on the fly.

We employ the widely used discrimination tree index for unit subsumption and unit deletion. Only the temporary area, not the facts in the shared memory database are kept in the index.

The engine uses a version of a set-of-support binary resolution with common optimisations like subsumption and tautology elimination. See Robinson and Voronkov (2001) for the common algorithms employed in first-order automated reasoners.

We have to avoid re-derivation of facts which were already derived during the last session. We cannot rely solely on the subsumption algorithm for this. For example, the robot should not get the derived command facts again each time the derivation session finishes.

Hence we developed a timestamp-oriented special version of the set of support algorithm. The initial facts in the derivation are only those which have been added (or modified) in the database after the previous derivation session. This is possible, since all the facts in the database have the automatically stored timestamp field.

We cannot use, for example, hyperresolution, since this derivation algorithm is not complete in combination with set of support. Hence the use of binary resolution.

The new facts and (partial) rules derived using a binary resolution step can then be used for deriving new facts and rules, guaranteeing that in each derivation chain at least one of the sumption facts has been added/modified after the previous derivation session.

Using the timestamp-oriented set of support algorithm is also crucial for efficiency. The number of new facts added in one second is normally not very big, and most of them typically do not match any or most rules. This keeps the amount of derived facts during one derivation session down even for relatively large rulesets.

7.1 Behaviors

Behaviors are collections of operations that the robot will perform and which are called with one command. Implementation of a behavior is a little binary executable written in the C language. It contains a sequence of commands and conditions to perform a relatively complex operation by the robot.

For example, let us consider the following ruleset:

```
behavior(me, "monitorObstacles"):-
state(me, stateInitial).

behavior(me, "goAhead 200"):-
state(me, stateCanmove),
obstacle(me, nothing).

behavior(me, "handleFailState"):-
result(solveObstacle, fail),
state(me, stateDriveAround).
```

- behavior - a special name, indicates that the fact is the command to launch the given binary.
- monitorObstacles - a binary monitoring whether any obstacles are getting in the robots way. If there is an obstacle in front of the robot, the $obstacle(me, front)$ row will be added to the datastore.
- goAhead - a binary that makes the robot to start moving forward with the given speed (in the current case with the translational velocity 200 mm/s and angular velocity 0)
- handleFailState - a binary that stops the motors and sensor equipement to save power, then tries to communicate the information about the failure situation to other robots or the central server. Used when the robot is stuck and unable to move or trapped in the place where it cannot find the way out.
- solveObstacle - a binary that tries to drive the robot away or around the obstacle which has gotten in the way.

All the behaviours are handled by the process we call dispatcher. The dispatcher executes binaries: small executable programs implementing the behaviours. In order to make the dispatcher to execute one binary, it must be copied to a predefined folder on the robot and at the desired time the proper command must be inserted to the memory datastore.

An example of a command row which forces the dispatcher to execute a binary:

subject	property	value	source	context
me	behavior	command	wGandalf	dispatcher

* command - "behaviorName arg1 .. argN"

While implementing the robot control application on top of the prover and a relatively large set of behaviours, timing becomes a critical issue. The elapsed time between a stimulus and its reaction varies greatly depending on the current contents of the memory datastore, the length and the complexity of the rule file, the number of processes running in the system, the length of the reaction chain and other factors. However, in our case study the response times have proved to be acceptable.

For example, let us consider a cooperation between the prover, the dispatcher and two behaviours to avoid the robot colliding with an obstacle. Typically it takes about 400 ms from the moment when one behaviour (monitorObstacles) discovers an obstacle to the moment where the prover inserts a command into the memory datastore to launch another behaviour. After about 20 ms the dispatcher has received the command and is ready to start the given behaviour. After additional 100 ms the second behaviour (solveObstacle) takes over the control of the robots movement.

High-level decision making can safely rely on the given architectural scheme. However, critical emergency responses like avoiding the robot falling down the stairs after the cliff sensor detects descent should be implemented in hardware or low-level software agents.

8. ROBOT DATA STORAGE ON THE SERVER

Robots using WIFI can use the robot-server centralised communication and robot-robot ad hoc communication in case no WIFI access-points are available. A separate

process on the robot sends new data items from the memory datastore to the server. On the server side the data of the whole swarm is stored in a postgresql database for further processing.

The server replies each uploading act with the new data items intended for this particular robot, accumulated since the last communication session. Software agents on the server cannot directly send any data or commands to robots, in lieu of that they will write the data into the same postgresql database. The special communication agent then passes it to the selected robot as soon as the robot contacts the server. The selected robot adds the data items received from the server to its own memory datastore.

The CSV protocol version sends data over http POST. The first row in the data block contains a sender id, the following rows contain the memory datastore rows in the standard CSV format. The protocol is used to both send data from the robot datastore to the swarm postgresql database on the server, and vice versa: from the swarm database to the single robot datastore.

The CSV protocol data format:

```
robot id
subject,propery,...,usecstamp
...
subject,property,...,usecstamp
```

with the rows containing the same fields as the memory datastore: subject, property, value, valuetype, source, context and two timestamps both comprising seconds and microseconds.

Human users can control and monitor swarm or single robots via dedicated user interfaces built on the server database. It is technically possible to assign a direct task to the robot, even though the data flow normally passes several intermediate agents on the server. Data produced by the user interface is sent to the task decomposition module which specifies and assigns proper subtasks for the robots.

The server has an additional swarm coordination role in some applications. For example, if we consider the task where a group of robots must search for an RFID-tagged object, it is reasonable to use the server. After the user has given the task, the server distributes the task information down to the suitable set of robots which then spread out in the environment and start performing the search. As soon as one of the robots has found the demanded object, it will communicate the knowledge to the server which then informs the user and notifies the other robots to stop searching.

9. RELATED WORK

The SHAGE/AlchemistJ framework Lee et al. (2008) should be mentioned as a solution for robot knowledge exchange using data repositories and component brokers.

High level data representation is a trend of modern robotics. For example, XML based data coding has been used on robots Lee et al. (2008). However, besides the benefits of universal high level representation the XML encoding requires additional conversions between exchange and machine control domains.

Conventional XML based RDF format is used for time uncritical inter-robot or server communication, the description can be found in Ardizzone et al. (2006). A special, compact RDF format is used for storing real-time algorithms of robot operation.

10. CONCLUSIONS AND FUTURE WORK

We have designed the robot swarm system and are conducting actual testing with real tags and robots. So far we have successfully implemented both the robot hardware and software, including the memory datastore, RFID and rule engine usage, communications with the server and the server database as described in the paper. The experiments have been encouraging, especially when we consider the weak processing power and high reaction times: the tiny onboard Gumstix computer manages to run the described agents, use the RFID chips, memory datastore and the rule engine without slowing down the robot reactions. Porting the system to the larger Pioneer and Mugiro robots via the Player interface was relatively easy. As a consequence, we now have three very different robots able to communicate through the same infrastructure.

On the other hand, detecting, reading and writing RFID tags requires considerable care when selecting tag types, antennas and the scanning frequency. A usable solution has been worked out, but further optimisations and improvements are needed. Optimization of different aspects of collective cleaning task is ongoing work and to be described in forthcoming papers.

REFERENCES

Altshuler, Y., Bruckstein, A., and Wagner, I. (2005). Swarm robotics for a dynamic cleaning problem. *IEEE Swarm Intelligence Symposium 2005 (SIS05)*, 209–216.

Ardizzone, E., Chella, A., Macaluco, I., and Peri, D. (2006). A lightweight software architecture for robot navigation and visual logging through environmental landmarks recognition. In *International Conference on Parallel Processing Workshops*.

Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence*, 26(3), 251–321.

Lee, S., Suh, I., and (Eds), M.K. (2008). Recent progress in robotics. In *Lecture Notes in Control and Information Sciences 370*, 385–397. Springer.

Robinson, J.A. and Voronkov, A. (eds.) (2001). *Handbook of Automated Reasoning*. MIT press.

Tammet, T. (1997). Gandalf. *Automated Reasoning*, 18(2), 199–204.

Tammet, T., Vain, J., and Kuusik, A. (2006). Using rfid tags for robot swarm cooperation. *WSEAS Transactions on Systems*, 5(5), 1121–1128.

Tammet, T., Vain, J., Kuusik, A., Puusepp, A., and Reilent, E. (2008). Rfid-based communications for a self-organising robot swarm. In *Self-Adaptive and Self-Organizing Systems*.

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

Paper C

A. Puusepp, T. Tammet, M. Puju, E. Reilent. Robot movement strategies in the environment enriched with RFID tags. 16th International Conference on System Theory, Control and Computing, Sinaia, Romania, 12-14 October 2012.

Robot movement strategies in the environment enriched with RFID tags

Andres Puusepp, Tanel Tammet, Madis Puju, Enar Reilent

Abstract— **The robot navigation in the environment with RFID tags was investigated during a series of simulated tests. Passive RFID tags were placed on the floor to irregular positions not forming a grid. Environments with different room configurations were considered. A robot with bumper sensors, RFID reader, and odometry is given the task to drive around the environment and search for the specified number of tags. Minimizing time needed for accomplishing the task is taken for the target. Three different strategies were investigated and corresponding algorithms evaluated on the simulator. One approach is minimalistic and uses only random movement and bouncing back from walls and obstacles. The second strategy tries to remember the sequence of encountered tags and adjust its movements when finds itself on the same path that was passed before to diversify its routes. The last algorithm uses predefined map of locations of tags and the robot has to visit the tags using navigation, provided that it is able to determine its own location and heading beforehand. The performance of different algorithms is compared based on the results of multiple test runs on the simulator.**

I. Introduction

Autonomously moving robots tend to become gradually more and more common in our everyday lives for performing tasks like vacuum cleaning of floors, guiding visitors in buildings, doing some environment surveillance or amusing people. Independently from the specific purposes or use cases they all rely on localization and navigation capabilities, it makes no difference if indoor or outdoor, (however, chaotic motion proves to be usable for coverage [12]). While focusing on the indoor navigation there exist several approaches and sensor equipment, for example orienteering methods based on laser range finder, magnetometer [5], visual image processing, etc. Another possibility is to place special landmarks into the environment and use them for guiding robot's movements. The landmarks could be active or passive, active radio beacons (e.g. [2]) seen from long distance enable triangulation; passive landmarks could be less expensive and computationally not so demanding.

One option is to make use of inexpensive RFID (Radio-frequency identification) tags as passive landmarks which depending on the details of the set-up might not provide the best accuracy but can suit well in the cases of less sophisticated robots. RFID tags have many possible exerting strategies: they could be mounted either on the walls ([11], [4]) or on the floor ([3]), positioned into a regular grid (e.g. [9]) or randomly. When a small number of tags are used in a rather large and complex set of rooms then the tags serve mostly as a navigational graph for robots which drive from tag to tag whereas trying to reach their destination area as described by [1].

Supposing the robots are equipped only with odometers and RFID readers (plus collision detection) they could not navigate on the graph for a long time without regular updates of position and direction because of the accumulating odometry error and tag detection inaccuracy (antenna is not underneath the robot). However, resetting location is a slow process and if done frequently it consumes considerable amount of time. Nevertheless, the robot gets lost occasionally and has to wander around until able to position itself again.

If the task of the robot rests on coverage, like cleaning for example, and the robot itself lacks additional means for localization then in some cases it might be preferable to spend more time on moving and abandon the idea of precise navigation. However, the robot could still make use of tags to adjust its behavior to perform better compared to the environment with no tags.

Our goal is to investigate potential uses of RFID tags for enhancing robot's performance in the situations where precise navigation is not feasible or desired. Based on the same setup as described by [13] – using the Roomba vacuuming robot (diameter 30 cm) with mounted embedded computer, Skyetek RFID reader (European UHF 865,7 MHz – 867,9 MHz, 24 dBm) and patch antenna, passive RFID tags (ISO18000-6b, effective reading distance is comparable to the robot's dimensions) on the floor) – simulated experiments were conducted to evaluate robot's performance under various different conditions.

A custom made simulator was used in the experiments to have fine-grained control over the test setup and make the robot and RFID as similar as possible to the real world case. The task was specified for the robot to move around in the closed two-dimensional environment until it has found the given number of different tags – in other words, it is coverage in the sense of visiting RFID tags with focus on minimizing the time spent on the task. Different room configurations and densities of tags were used while testing. Several control algorithms were created, run, and compared on the simulated robot.

We considered three different approaches for robot's control algorithm based on the usage of tags to guide the robot. The "random" algorithm (lower bounder) has no initial knowledge about the environment and does not adjust its

A. Puusepp, T. Tammet and M. Puju are with the Department of Computer Science, Tallinn University of Technology, Ehitajate tee 5, 19086, Tallinn, Estonia (corresponding author e-mails: anduoma@hot.ee, tammet@staff.ttu.ee, pudismaju@gmail.com)

E. Reilent is with the ELIKO Competence Centre, Teaduspargi 6/2, 12618 Tallinn, Estonia (corresponding author e-mail: enar.reilent@eliko.ee)

movement according to the encountered tags. The opposite "map aware" solution (upper bounder) knows coordinates of all tags and tries to navigate to the nearest unvisited tag. The third algorithm lies between those two extremities and is called "history based" as it remembers and uses the sequence of seen tags to change its course. The motivation for this algorithm is to find simple and low-cost ways to improve the performance of random walk which turns out to be a quite good starting point (as by [8]) in the cases where sophisticated solutions exerting FastSLAM (e.g. [6]) are not applicable due to the hardware constraints.

The paper is organized as follows. The next section describes the background details of all algorithms that were used in testing. Section 3 focuses on the issues of experimenting itself and discussion of results.

II. ALGORITHMS

Currently there have been three different algorithms surveyed – random, history based and map aware algorithm. Each test of an algorithm has the set of configuration properties which tell how many times the test should be executed with the current configuration, what percentage of tags must be found to consider a test run successfully finished, connection parameters to the simulator and some values concerning the algorithm itself (e.g. angles, velocities). Percentage calculation of found tags is the same for all algorithms and is based on the number of encountered unique tags and the total number of tags in the current room. Total number of RFID tags in a room is taken from configuration (room definition), all tags have unique identifiers in the scope of the test envirenment.

A. Random algorithm

Random algorithm is the simplest, basically doing two things – in the case of seeing a tag registering it and in the case of bouncing into an obstacle changing robot's direction by a degree taken from the test run's configuration (performing minimalistic random walk). When a test starts then the robot is always placed on to the same coordinates in the given room. Robot starts moving and drives straight until it bumps into an obstacle (including walls), depending on the direction of the collision heading will be changed. There are two types of collisions, firstly, when both robot's bumpers give the signal, and secondly, when just one (left or right) of the bumpers gives the signal.

Configuration holds four different values of angles for this occasion:

- Degree to turn when the left bumper is set
- Degree to turn when the right bumper is set
- Degree value to turn left when both bumpers are set
- Degree value to turn right when both bumpers are set

Bumping with the left or the right bumper makes the algorithm to use the specific degree value from the configuration. When the robot hits an obstacle directly in front of it and both bumpers are set then one of the both bumpers' degree values is taken from the configuration randomly. Using two values makes it possible to turn robot left and right randomly in direct collisions. Turning direction is defined by the degree value – positive value turns right and negative value turns left.

Random algorithm has been tested with different rooms having seven test sets with different configuration per room. In each set the robot must find 85 percent of tags in the room and test is repeated multiple times to find statistical averages. All seven sets have the same configuration values (90 and -90 degrees) for both bumper collisions. Left and right bumper collisions will make robot turn to opposite direction according to our algorithm configuration. For example if the robot hits an obstacle with the left bumper then its direction will change by given degrees to the right. Those values differ in each test set by 15 degrees, starting from 30, -30 degrees for the first set and ending with 120, -120 degrees for the last set.

B. History based algorithm

History based algorithm is a bit more complex than the random algorithm, basically sharing random behavior's logic for situations where history is not able to offer any help for decision making. The general idea is not to go again by the same path that the robot has passed before during that test run. To be able to recognize these paths the robot must keep the chronological history of seen tags.

Algorithm collects and uses the history of seen RFID tags to adjust the robot's movement. In general it works similarly to the random algorithm, especially when concerning task completion, moving principles and default collision handling. But in this case all tags that the robot sees during its maneuvering are carefully stored into the so called moving history together with odometry information.

The moving history consists of straight moving episodes where each of them records all tags seen during that drive, the length of the drive and the angle turned at the end. For individual tags there are also distances from the beginning point of the drive. That structure is used for changing the robot's course when needed which can happen near to the obstacle / wall as a part of collision handling as well as in the open ground in the middle of one moving episode.

Main goal is to compare the current move to the previous moves, suppose that the robot has found at least two tags during its ongoing drive and the similar episode is found from the moving history - there is an episode with at least two tags matching, then the current move is finished right away and the robot turns to another direction which is also saved into the moving history. The exact number of degrees to be turned is calculated with the angle from the last similar move and a value specified in the configuration.

If the robot happens to visit the very same area again and recognizes the situation, it changes its course once again and turns by some number of degrees which is calculated from the degrees turned at the same place last time plus parameter "delta" value – so that every next time the new course change is delta degrees larger than the previous one.

However, if the calculation is starting to give values once already used, the robot is driven further by given small distance and only then turned to different heading. The

general collision handling procedure is also affected by the moving history and the robot checks history at the end of every straight moving episode. If the move can be identified by the tags seen and their distances from the endpoints of the moving episode, the direction of the next move is tried to be diversified as much as possible according to the delta mechanism. This history check is applied also when only one tag was seen during the moving episode.

Configuration file contains the same parameters as they are for the random algorithm. Additionally there is the delta parameter which tells how many degrees to increase turning value if the same situation has already been met during the current test run.

Similarly to the previous algorithm, this one is also tested in different rooms and with seven different configuration sets according to the principles used with the random algorithm. Shared configuration values are the same and the additional delta value is set to 15 degrees, however, the very first turn made by the robot at the given spot for changing the course is 30 degrees (then 45, 60, 75, etc).

C. *Map aware algorithm*

Similarly to the history based algorithm we use the same random movement principles for this algorithm's base. Additionally, here we know the location of all tags in the room. There is no knowledge of other objects like walls, furniture etc. Most configuration parameters are the same as they are for the random algorithm, additionally the parameter for location of tag coordinates database file is needed. Tag information is provided from the same source as it is for the simulator instance, no conversions needed. The file holds three values for each tag, tile's x and y coordinate and id of the tag.

Tags placement in the room is described loosely. Objects in the simulator's world are described tile by tile and so are the RFID tags. The robot has approximate knowledge of tag's locations and using that information it tries to locate the tag. There is no need to use precise location markers due to the fact that while driving and turning the robot's odometry loses accuracy and the robot's estimation of its position is always imprecise. It could be also noted that the effective reading area for the RFID tag is comparable to the dimensions of the robot but considerably larger than a tag. Knowing the tags locations is more like a guiding advice not a situation where the robot should be able to drive exactly to the center of a tag.

Algorithm moves the robot around the room based on the random movement's principals until able to localize itself, then starts to navigate. General principles of the robot navigation are presented in [7], however, we try to avoid planning ahead complex paths. Finding more than one tag during a single straight moving episode makes the algorithm stop the current move, choose the closest not yet visited tag to the current position as the next target location, and then continue moving.

Even if the robot misses the target tag it has moved closer to the area where the tag is and there might be also other unvisited tags close by. When the robot finds the tag it continues navigation, otherwise it simply switches to random movement until accidentally passes two tags and regains its location and direction. The scenario is used mainly for reference to evaluate other algorithms. It relies on the predefined map which is not available in the real case if not made by the robot itself beforehand. If the robot has to gather the map information during the test run there is no significant advantage of using the (incomplete real-time) map as the robot cannot know the positions of tags which it has not yet discovered.

Algorithm has been tested like others with different room setups and seven separate configuration sets. Angle turning values and finishing conditions are also the same as with the random algorithm, no delta parameter used here.

III. RESULTS

A. *About the testing process*

Due to the time consuming process of developing algorithms on the real robots and running considerable amount of tests in different room configurations the decision was made to use custom virtual environment for better control over interfaces and automated test execution with logging images, statistics etc. First simulator was built in Python using Panda3D framework which turned out to be too resource consuming. Next version was built on Java JGame framework and is used currently. Lower resource requirements allow running multiple instances of simulator at once so we are able to run more tests in less time. Starting from the beginning of research and development of algorithms total count of test runs is approaching to 5000.

Environment contains of two programs – simulator and the algorithm program, both are written entirely in Java. Communication between them is done by using sockets interface and commands from Roomba's specification. During the testing process of an algorithm one simulator instance and one algorithm program instance with test set specific configuration is used. The same test is being run number of times specified in configuration. While a test set is running the simulator generates easily readable HTML reports with the image of the simulator window, test duration and tag finding statistics for each test. The image represents the last state of the simulator when the test successfully ended. Statistics contain data for each tag: identifier, at which point of time the tag was found, what percentage of time was spent to find the tag, the sequence number of the tag in the chronological list of discovered unique tags, also the count of how many times that tag was seen during the test run.

All algorithms have been tested in the same environments and under the same conditions. Maps presented below are approximately the same size being equivalent to the real world's size of 60 square meters each except for the room 4 (see Figure 4, all referred room numbers match with the numbers of the figures) which has 28 square meters of size. Rooms 2 and 3 have been covered with 68 tags, room 1 with 55 tags, room 4 with 17 tags and room 5 with 669 tags. Each test has been considered successfully completed when the robot has found 85% of tags on the map.

The idea to search for specified percentage of tags came from the earliest tests. Spending approximately 1/3 of each test run time to find the last 15% of tags did not seem to be healthy. The very first test sets showed that the first 1/3 of the tags could be found around 10-15% of entire test run time, finding 2/3 of tags took about 35-50% of the time. Looking at multiple test results and statistics of tag finding the decision was made to set the limit to finding 85% of tags during each run.

In all these mentioned scenarios the robot moves only by straight lines with a constant translational velocity or turns on the spot with a constant angular velocity, all velocities are the same for all tests. Turning by given number of degrees is done by odometers reading only. No wall following is present in the current case. The random algorithm has also been tested with non-straight movement. Straight movement episodes were replaced with circular movements with radius constantly growing until upper limit. Obstacle resolving logic remained the same. Unfortunately this approach ended up with much worst problem resolving times and therefore not discussed further.

On the following figures brick tiles represent walls and other impassible terrain. Little antennas around the map represent RFID tags, when they are crossed through then the robot has seen them at least once. The black circle is the robot, little red dots in front of the robot represent bumpers and green dots in front of them represent the area where the robot is able to detect RFID tags.

B. Outcomes

The tests give best results when turning robot 30 to 60 degrees after finding similar sequence of tags from history or by bumping with one of the bumpers. The map aware algorithm performs extremely well in one large room setup, but with multiple rooms setup the robot can get stuck. For multiple rooms setup with this algorithm the robot can get stuck for some time because of knowing only about the tags and nothing of other surrounding objects. Removing this flaw could make this algorithm give far better results than two others. Map aware algorithm has the best performance in multiple rooms' setup if it is able to resolve the test without getting stuck.

While moving around in the simulated environment the robot always leaves a trace of black line. After tests have finished it is convenient to give firsthand evaluation how well each test went by the coverage of black line on the map. An example of quite unsuccessful run can be seen in "Fig. 1" where the random algorithm spends lots time on visiting the same places over and over again.

Average test duration values point out of which are the best turning degrees (in the case of one bumper hit) to be used for achieving good results. An initial thought could be that with bigger turning angles the covering process of rooms will also be faster as the robot is more likely to bounce back towards the center of the room instead of staying near the walls, but in multiple room setups it actually makes harder for the robot to go from one room to another through a door (opening).

The "table 1" describes results from seven test episodes. Each episode has thirty runs and results are average values of these runs in seconds. (Smaller number indicates better result.) Episodes configuration parameters differ by the turning angle after one of the bumpers gets hit. These values are obtained from the test runs in room setup number 2 illustrated on the "Fig 2".

Based on the results it became clear that 30, 45 and 60 degrees are the best configuration values to be used. There might be a chance to get good results also with 75 degree turning angle, other values start taking almost double the time which is too much. Decision is made by looking at the history based and random algorithms because they have comparable amount of knowledge at the start. Results also indicate that bigger angles make robot to spend more time on turning.

The history based algorithm depends less on the bouncing angle than the random algorithm, but still more than the map aware algorithm. Even as it depends on the room configuration the history based approach tends to be better than the random, according to the intuition, however, outperforming the map aware algorithm is only apparent. For getting better understanding of these three algorithms it is reasonable to look at the peak times spent at each episode. Minimum and maximum values will give an idea of potential of an algorithm but also how stable the algorithm is.

TABLE I. AVERAGE TEST RUN TIMES IN SECONDS

Angle (degrees)	History based (seconds)	Random (seconds)	Map aware (seconds)
30	1305	1975	1009
45	1188	1975	1915
60	1628	2016	2200
75	1703	5139	3078
90	4247	5800	1974
105	3714	5715	3101
120	3317	3976	2882

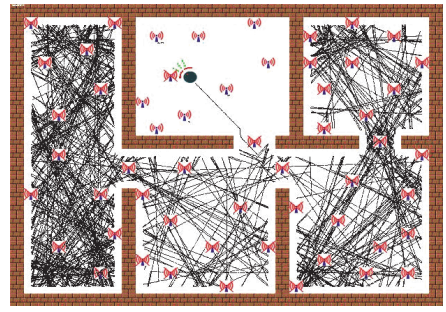

Figure 1. Room setup no. 1 – random algorithm having a really awful run and getting stuck in rooms.

Each column shows minimum or maximum test run value for all three algorithms in room setup number 2 illustrated on "Fig. 2" and are in the same order as in "table 1" – the first line with 30 degrees bumping angle, the second with 45 degrees, etc.

Peak values seen in “table 2” indicate that the random algorithm’s minimum problem solving times can be slightly better if compared to the history based algorithm. Due to the fact that the random algorithm does not remember anything it can also get badly stuck or visit the same areas too many times, which increases the maximum resolving times really high. Based on these test results, it can be concluded that the history based algorithm is able to give more persistent results even if the random bouncing seems to give some really good performance examples.

The history based algorithm is worse than the map aware as expected but not very much in the cases of better (smaller) turning angles, if minimum values are considered. On the other hand, if minimums of the map aware are compared to averages of the history based, it is clear that the current history based algorithm has room for improvement. Currently the weak point of the history based algorithm is when the robot enters into the same state multiple times which ends up with suggesting angles that have been already covered. These situations occur mostly when the last tag, seen before changing direction, is near the wall and instead of diversifying the route the robot spends time on bumping the wall. Still, the history based algorithm has some advantages over the random movement.

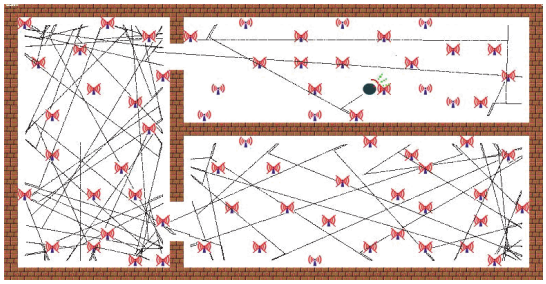

Figure 2. Room setup no. 2 – an example of random algorithm having a good run.

TABLE II. MINIMUM AND MAXIMUM TEST RUN TIMES IN SECONDS

Min history	Max history	Min random	Max random	Min map	Max map
864	1656	582	2450	574	2153
786	1965	1136	2836	622	9109
1033	2412	1055	10838	840	5351
591	3700	984	5577	523	10331
1643	8402	1610	12132	589	3670
1185	8237	2980	10714	526	3516
1406	4418	1320	12967	878	5806

The map aware algorithm also has large maximum problem resolving times, which are consequences of getting stuck in some cases, e.g. as in “Fig 3”. Potential of this algorithm can be seen from minimum values, where it outruns every other algorithms execution time and indicates the best possible solution where the robot smoothly passes all tags. The lack of knowledge about the objects for this algorithm could possibly be solved with remembering the obstacles and avoiding repeating unsuccessful moves. This kind of approach might trim down map aware algorithm’s maximum and also average running times drastically (but would not improve minimum times).

As said before, the map aware algorithm is applicable only in the case where the robot knows the locations of all tags in advance. Fast performance is the result of using navigation. In the case where no map exist the robot must still drive around by the history based algorithm and gradually gather information for composing the map which can be used later for speeding up next tasks in the room. For making map the robot first measures distances between tags using odometry and then calculates coordinates of tags from incomplete and inaccurate data [10].

The “Fig. 4” gives an example of a map which contains a single room. Looking at test results (illustrated in “table 3”) of one single room which has some small randomly placed obstacles we get proportionally more or less the same results compared to other room setups. The map aware algorithm has especially good results, because it has the best results almost for every configuration and still having the flaw of not being able to avoid the walls and obstacles.

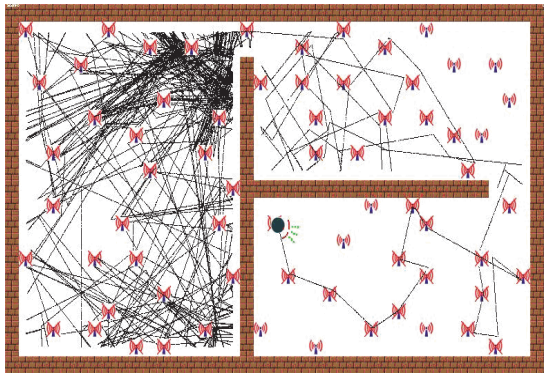

Figure 3. Room setup no. 3 – map aware algorithm is getting stuck at the door but finding last tags quickly.

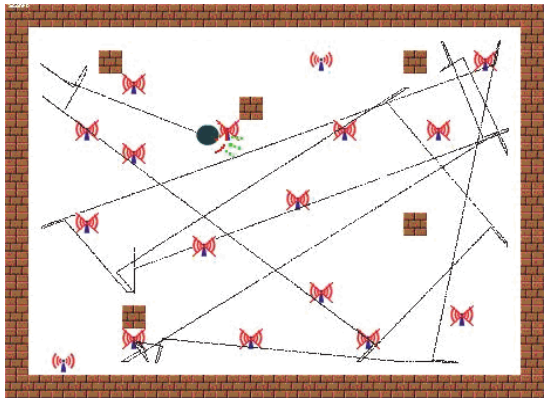

Figure 4. Room setup no. 4 – history based algorithm having a good run.

C. *Outcomes of extreme tag coverage*

Tests have been running with different room setups and randomly placed tags with coverage about 15 – 25%. One idea was to try extreme situation where all movable space in room is covered with tags as illustrated on "Fig. 5". The goal was to see how it affects the performance and will it make the algorithms to start doing stupid moves.

This approach was tested with all algorithms, but the greatest impact was to history based and map aware algorithms. Comparison was done with three different room setups with multiple test episodes (each executed multiple times). Results indicated slight rise in test run times. One of the reasons is that tag will not always be registered as soon as it is in the radio coverage. For map aware algorithms case this means that sometimes robot thinks that it is better to turn, than to drive straight forward. Unfortunately in real world this means additional time spent to turning, which will result in time loss and also means worse performance. Running algorithms against room setups with full of RFID tags resulted with average run time increase up to 50%. Worst cases ended with time loss of 200% or more, but the proportion of these cases was minimal – lower than 5% of all runs.

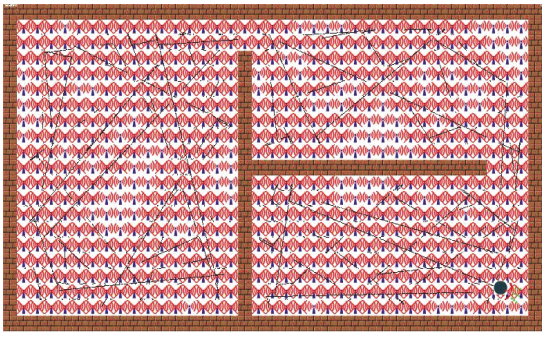

Figure 5. Room setup no. 5 (the same layout as room no. 3) - extreme situation where all movable space is covered with tags

TABLE III. AVERAGE TEST RESULT TIMES FOR A SINGLE ROOM IN SECONDS

Angle (degrees)	History based	Random	Map aware
30	244	340	184
45	337	632	227
60	296	284	267
75	256	291	220
90	287	291	290
105	359	351	211
120	390	367	173

IV. CONCLUSION

Three different algorithms were implemented and tested in the custom-made simulated environment to investigate different possibilities of using RFID tags for guiding the robot which has no reliable odometry, abundant processing power or other sensor equipment (laser range finders, sonars, cameras, GPS, etc.). The history based algorithm (an unpretentious alternative to random walk and preceding phase to map aware algorithm) is able to find tags with good persistent times, but still remains behind the random algorithm's best result times. The map aware approach has the shortest problem resolving times but average can be quite low due to the shortcomings of the present implementation. It has relevance only when the map is constructed beforehand. Using loose tag data as input for algorithm has great potential to produce the best average times, so it should be considered a way to go.

Making map aware algorithm smarter by recognizing "stupid" moves from the past would be one solution for better performance. When the robot has some understanding of walls and obstacles it would be possible to make use of more sophisticated navigation and drive to the distant target without line of sight by applying multi-hop. Also map making should be added to the history based algorithm in the future.

REFERENCES

[1] M. Baglietto, G. Cannata, F. Capezio, A. Grosso, A. Sgorbissa and R. Zaccaria, "PatrolGRAPH: a Distributed Algorithm for Multi-Robot Patrolling", IAS10 - The 10th International Conference on Intelligent Autonomous Systems, Baden Baden, Germany, pp. 415 – 424, July 2008.

[2] M.A. Batalin, and G.S. Sukhatme, "Coverage, Exploration and Deployment by a Mobile Robot and Communication Network", in Proc. International Workshop on Information Processing in Sensor Networks, 2003, pp. 376 – 391

[3] J. Bohn and F. Mattern, "Super-Distributed RFID Tag Infrastructures." Lecture Notes in Computer Science, vol 3295, pp. 1 – 12, 2004.

[4] D. Hahnel, W. Burgard, D. Fox, K. Fishkin and M. Philipose, "Mapping and localization with RFID technology.", International Conference on Robotics and Automation, pp. 1015 – 1020, 2004.

[5] J. Haverinen and A. Kemppainen, "A global self-localization technique utilizing local anomalies of the ambient magnetic field.", International Conference on Robotics and Automation, pp 3142 – 3147, 2009.

[6] H.S. Jeon, M.-C. Ko, R. Oh and H.K. Kang, "A practical robot coverage algorithm for unknown environments", in Proceedings of the 9th Mexican international conference on Advances in artificial intelligence: Part I Pages 129-140, 2010.

[7] J.-A. Meyer and D. Filliat, "Map-based navigation in mobile robots. II. A review of map-learning and path-planning strategies", Cognitive Systems Research, pp. 283 – 317, 2003

[8] R. Morlok and M. Gini, "Dispersing robots in an unknown environment", in 7th International Symposium on Distributed Autonomous Robotic Systems (DARS), June 2004.

[9] S. Park and S. Hashimoto, "Indoor localization for autonomous mobile robot based on passive RFID.", International Conference on Robotics and Biomimetics, pp 1856 – 1861, 2009.

[10] J. Schlitter, "Calculation of coordinates from incomplete and incorrect distance data.", *Journal of Applied Mathematics and Physics*, vol 38, pp. 1 – 9, 1987.

[11] S. Schneegans, P. Vorst and A. Zell, "Using RFID Snapshots for Mobile Robot Self-Localization.", European Conference on Mobile Robots, pp. 1 – 6, 2007.

[12] A. Sekiguchi and Y. Nakamura, "The Chaotic Mobile Robot", IEEE Transactions on Robotics and Automation, volume 17, issue 6, pp 898 – 904, 2001.

[13] T. Tammet, J. Vain, A. Puusepp, E. Reilent, and A. Kuusik, "RFID-based communications for a self-organizing robot swarm.", Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, pp. 45 – 54, 2008.

Paper D

Puusepp, A.; Tammet, T.; Reilent, E. (2014). Covering an Unknown Area with an RFID-Enabled Robot Swarm. Applied Mechanics and Materials, 490-491, 1157 - 1162.

Applied Mechanics and Materials Vols. 490-491 (2014) pp 1157-1162 *Online: 2014-01-28*
© (2014) Trans Tech Publications, Switzerland
doi:10.4028/www.scientific.net/AMM.490-491.1157

Covering an Unknown Area with an RFID-enabled Robot Swarm

Andres Puusepp[1, a], Tanel Tammet[1,b] and Enar Reilent[2,c]

[1]Tallinn University of Technology, Tallinn, Estonia

[2]ELIKO Competence Center, Tallinn, Estonia

[a]anduoma@hot.ee, [b]tanel.tammet@ttu.ee, [c]e.reilent@gmail.com

Keywords: Multirobot systems, robotics, RFID, robot swarm.

Abstract. Our goal is to improve the coverage of an area using robots with simple sensors and simple, robust algorithms usable for any kind of room. We investigate the advantage of the swarm - compared to a single robot – and three different algorithms for the task of searching landmarks in a previously unknown area. The guidance of the robot is based on landmarks, implemented by RFID tags irregularly placed in the room. The experiments are conducted using a custom made simulator of RFID-equipped Roomba cleaning robots, based on our previous work with real-life Roomba swarms. We show that for the simple room coverage algorithms the speedup gained from increasing the size of the swarm diminishes as the swarm grows and most importantly, for larger swarm sizes the information available and the intelligence of the algorithm becomes less important.

Introduction

Simple autonomous robots are becoming common for applications like cleaning [1], rescue [2], mowing [3] and surveillance [4]. Regardless of the robots' mission they all depend on localization and navigation capabilities. The general context of our paper is improving the landmark search algorithms – similar to room coverage - of a room by a swarm of robots. We assume the swarm to be a set of robots following the same algorithm of behaviour and solving a common task in parallel.

There exist several approaches and sensor equipment kits for indoor navigation, like using a laser range finder, magnetometer [5] etc. It is also possible to guide robots using landmark-like radio beacons (e.g. [6]) or less expensive passive landmarks. The specific goal of the paper is to investigate the landmark search behaviour (henceforth called coverage) of swarms of robots without precise navigation capabilities, equipped with simple landmark-based navigation and following robust algorithms. In particular, we are interested in the effect of the size of the swarm and the effects of the search algorithm to the time it takes to cover a room.

The investigations reported in the current paper are performed on the custom developed simulator with a behavior very close to the physical RFID-equipped Roombas described above.

RFID tags can be spread around the area in various ways; the information discovered by a single member of the swarm is propagated to all the robots.

The task of the swarm was to move around the closed two-dimensional map and to discover a given number of tags as quickly as possible – in other words, minimizing the time of room coverage in the sense of visiting a given percentage of RFID tags in the room. Various maps and tag densities were used while conducting the tests. Several control algorithms were created, run, and compared on the simulated robot.

The most common solutions in literature for robot coverage are dividing the map into cells [7], [8] and path planning [9]. Our algorithms do not assume knowledge of the orientation or location of the robot and do not take advantage of the cited approaches. In contrast, we have experimented with the following simple algorithms.

First, the "random" algorithm has no knowledge of a surrounding environment and is simply driving around, bouncing against walls as it tries to find the tags. We have earlier [10] shown that the angle of a turn after bouncing into a wall is an important factor for the efficiency. The second, "history based" algorithm remembers and uses the sequence of tags seen to change its course. Third, the "map

aware" algorithm knows the coordinates of all tags and attempts to navigate the robot to the nearest unvisited tag.

The paper is organized as follows. The next section describes the background details of all the algorithms that were used in our experiments. The third section focuses on the experiments and the discussion of results.

Algorithms

As mentioned above, we have experimented with three simple algorithms – the random, history based and map aware algorithms. Our algorithms can be divided into three categories based on the amount of information obtained from tags found. All the algorithms can read the room name from the tag, understand whether a tag is a special door tag and decide whether the task is completed.

The Random Algorithm. The random algorithm does two things – registers each encountered tag and changes robot's direction each time it bounces into an obstacle. The degree of a turn has significant effect on the room coverage efficiency [11]: hence we turn by 30 degrees which appears to be the best in most cases.

For each test run the robot starts from the same place and drives straight until it bumps into an obstacle. There are two types of collisions: one where a single bumper (left or right) of the robot is toggled and another where both bumpers are toggled. In the first case we turn 30 degrees to the opposite direction of the toggled bumper. In the second case (both bumpers toggled) we will turn a random angle in the interval between 75 and 115 degrees.

The random algorithm has been tested with different maps with one to fifteen robots moving around simultaneously. During each test run all the robots together must find 85 percent of the tags spread around the map. The test is repeated multiple times to find statistical averages.

The History Based Algorithm. This algorithm adds additional logic to the random algorithm. The main idea of the algorithm is to avoid repeating already passed paths. The chronological history of tags seen during a single test run is stored and distributed throughout the robot swarm.

After seeing some tag sequence for the first time, the robot keeps driving straight until it bumps into an obstacle and then turns, based on the same principles as the random algorithm. However, when the robot detects a tag sequence which has been already followed (either by itself or any other robot) it will turn away to a direction different from the last visit and will then continue driving forward. The exact degree of a turn is increased by a small delta of 15 degrees each time the same sequence is travelled again by some robot.

The Map Aware Algorithm. This algorithm relies on the predefined map which is not available for the previous two algorithms.

Differently from the previous algorithms, the map aware algorithm knows all the tag locations on the map along with the room name the tag belongs to. No additional knowledge like the position of walls or obstacles or the location of the robot is provided. However, there are special tags for marking doors, used by the robot to stay in one room until all the tags in a specified area are found and the robot is free to move on to another area. The base parameters for the angle of a turn are the same as for the random algorithm.

The core of the algorithm is the same random movement as described above. However, finding more than one tag during a single straight line drive makes the algorithm stop the current move, find the closest not yet visited tag to the current position in the same room, turn the robot to that tag's direction and continue moving. The angle to turn is calculated by using the first and the last tag's coordinates from the current straight moving episode and the coordinates of the tag chosen as the next target. When the robot finds the target tag it continues to navigate to the next closest tag in the room, if it fails to find a target tag, it will simply switch to random movement until it accidentally passes two tags and regains its location and direction. Once the robot has found all the tags in the room it will attempt to exit the room by driving towards the special door marker tags.

Results

Testing Process. We conduct the experiments with the three before-mentioned algorithms for swarm sizes between one and fifteen.

There are four groups of experiments, with the results of each group depicted on a graph presented below. There are two kinds of rooms – single room without obstacles; whole map split into three rooms without any internal obstacles; and two kinds of starting configurations of the robots – all the robots start at the (almost) same place; the robots start at different places, distributed evenly in the space.

The robots participating in one test run share the same configuration values and follow the same algorithm. It is important to note that the guiding algorithm of the robots simulates the random fluctuations of both the robot turning angles, turning times and detecting tags as they actually occur in the real Roomba cleaning robots.

Each test set has its own specific configuration describing turning angles, start coordinates etc. Each test is run five times. The graphs below indicate the average time of these five runs.

The results and the graph on the Fig. 1 below show that the history-based and the random algorithm perform almost identically: the timing differences between the runs are random fluctuations. Hence the seemingly useful idea of avoiding paths already travelled does not translate into measureable gains for any swarm size.

On the other hand, the map-aware algorithm consistently outperforms the simpler map-agnostic algorithms. In particular, the gains obtained from the map-aware algorithm are most significant for small swarm sizes (58% of the coverage time of the random/history based algorithms for the one-robot case) and become somewhat less marked as the swarm grows (68% for a 7-robot swarm).

Most importantly, for all the three algorithms the benefit from increasing the swarm sizes is initially very strong but starts diminishing quickly after a certain point.

The coverage time of the two-robot case is almost twice smaller than for the one-robot case (regardless of the algorithm), and the coverage time of the four-robot case is again almost twice smaller than for the two-robot case. However, the coverage time of the last point of the graph (15-robot swarm) when compared to the 14-robot swarm is insignificant (again, regardless of the algorithm). We could pose a hypothesis that one of the reasons for diminishing gains is the need to travel to the far corners and edges from the common starting point: the time it takes is roughly the same regardless of the swarm size.

This leads us to the next experiment: the same (single) room with the robots starting from different locations near the walls of the room and initially headed towards the center.

Most importantly, there is no significant difference between the results of the previous same-location start experiment on Fig. 1 and the different-location start experiment depicted on the Fig. 2 below. This basically indicates that robots spread very quickly in the same-location scenario and the initial spreading process has little effect on the overall time. Quick spreading is made possible by the random fluctuations of robot behaviour as described above.

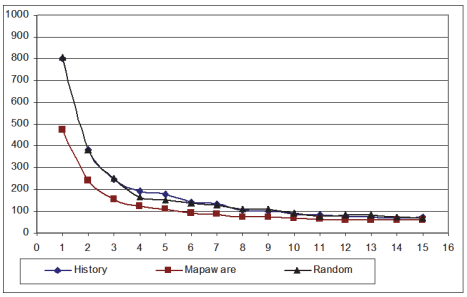

Figure 1. A single room with robots starting from one location.

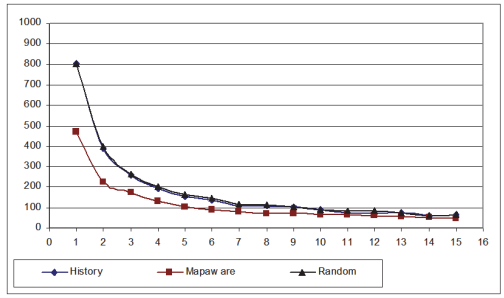

Figure 2. A single room with robots starting from different locations.

The next two experiments are conducted in a simple three-room space. The size of the space and the number of tags is the same as in the previous one-room experiment.

The principal complexity of covering a multi-room space stems from the problem of a robot (or several robots) being stuck in a few rooms and not reaching all the rooms necessary to obtain the 85% tag coverage condition. In the robot room coverage literature this problem is typically alleviated by different planning algorithms. Not so in our simple algorithms: the random and history-based algorithms are completely agnostic of the room the robot or the tags are located in at any given moment.

However, the map-aware algorithm always knows the room of the tag it has recently found. It first tries to find all the tags in the room and after this it attempts to escape the room by driving toward the closest door marking tags.

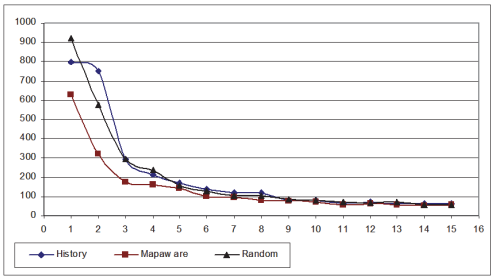

Figure 3. A three-room space with robots from different locations.

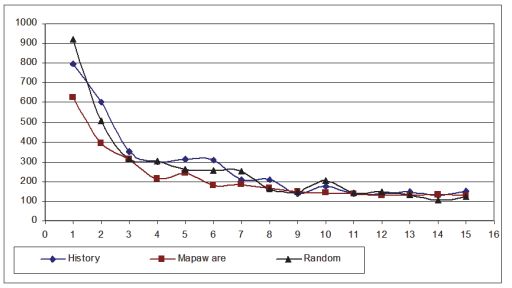

Figure 4. A three-room space with robots starting starting from one location.

First of all, observe that, as expected, for one robot it takes slightly more time (ca 15% for random and ca 25% for the map-aware) to cover the three-room space. However, for larger swarms the difference increases: it takes about twice as long to cover the three-room space for the 7-robot swarm when compared to the one-room space.

The diminished efficiency is mainly due to the abovementioned problem of spreading a swarm out evenly in multiple rooms. On the other hand, the effect of increasing the size of the swarm gives diminished returns somewhat similarly to the one-room space experiments

Most interestingly, the efficiency of the history-based algorithm has become noticeably different from the random algorithm. The history-based algorithm is a bit faster for the one-robot case and becomes a bit slower for the multi-robot swarms. The next experiment covers the three-room space with robots starting from different locations in different rooms (Fig. 3).

Obviously, there is no difference for a one-robot case when compared to the experiment on Fig. 4. As the swarm size increases, we see that the coverage time becomes similar to the one-room case, which is – again – not unexpected, since there will be at least one robot for each room in the initial situation. Again, the positive effect of increasing the swarm becomes smaller and smaller.

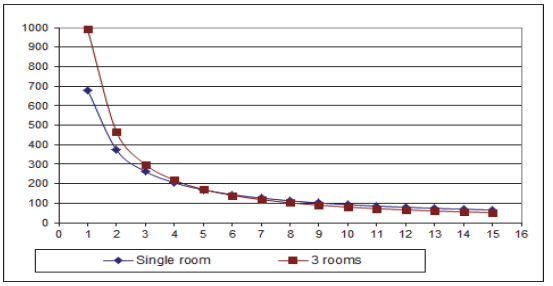

Figure 5. Averages for all the three algorithms

Finally we present a comparison of the average swarm behaviour for the two spaces described above. Each line in the following graph represents an average behaviour of a swarm for all the three algorithms considered.

The graphs (Fig. 5) representing coverage time as a function from the size of the swarm S are roughly $a*S^{-0.86}$ for a single room, $b*S^{-1.09}$ for the three-room setup, where a and b depend on the room size and robot speed.

Conclusions and Future Work.

Three different simple guidance algorithms were tested on four experimental room/initial position configurations for swarm sizes between one and fifteen. One of the goals of the experiments was to measure the impact of increasing the robot swarm on the time of room coverage by the swarm. As expected, for all the algorithms in all the experiments, gradually increasing the size of the swarm brings a smaller and smaller decrease of the coverage time.

Second, the experiments consistently showed relatively small differences between the random and optimized map-aware and history-based algorithms, leading to a conclusion that our medium-complexity algorithms do not give substantial benefits. Nevertheless, in simple rooms and a few robots the map-aware algorithm demonstrates practical speed improvement: for a single robot in one large room the coverage time was 58% of the random algorithm time. Also note that the paper [8] reports an almost consistent speedup of two times obtainable by their sophisticated coordination algorithm with robots having very detailed information about the rooms and their relative positions.

To summarize, for robots with limited knowledge the intelligence of the algorithm becomes less important as the swarm size increases.

Finally, we pose a hypothesis to be tested in the future work: it would be practical to optimize the spreading behaviour of the swarm by using only the tags located at doors to decide whether a robot should enter the door or not, disregarding all the other tags.

References

[1] H. Endres *et al*., Field Test of a Navigation System: Autonomous Cleaning in Supermarkets," in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 1998, pp. 1779–1781.

[2] R. Murphy, Human-robot Interaction in Rescue Robotics, IEEE Syst., Man, Cybern., C, Appl. Rev., vol. 34, no. 2, pp. 138–153, May 2004.

[3] Y. Huang *et al.*, "Automatic Operation for a Robot Lawn Mower, in SPIE Conf. Mobile Robots, vol. 727, 1986, pp. 344–354

[4] D. Hougen *et al.*, "A Miniature Robotic System for Reconnaissance and Surveillance, in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 2000, pp. 501–507.

[5] J. Haverinen and A. Kemppainen, "A Global Self-localization Technique Utilizing Local Anomalies of the Ambient Magnetic Field, International Conference on Robotics and Automation, pp 3142 – 3147, 2009.

[6] M. A. Batalin and G. S. Sukhatme, Coverage, Exploration and Deployment by a Mobile Robot and Communication Network, in Proc. International Workshop on Information Processing in Sensor Networks, 2003, pp. 376 – 391

[7] N. Agmon, N. Hazon, and G. A. Kaminka, "Constructing Spanning Trees for Efficient Multi-robot Coverage, in Proceedings of the 2006 IEEE International Conference on Robotics and Automation, vol. 1-10, (Orlando, FL, USA), pp. 1698-1703, 2006.

[8] W. Burgard, M. Moors, C. Stachniss, Schneider, F.E., Coordinated Multi-robot Exploration, Robotics, IEEE Transactions on, vol.21, no.3, pp.376,386, June 2005

[9] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon decomposition", in: Proc. of Int. Conf. on Field and Service Robotics, Canberra, Australia, December 1997.

[10] Puusepp, A.; Tammet, T.; Puju, M.; Reilent, E., "Robot movement strategies in the environment enriched with RFID tags," in Proceedings of the 16th International Conference on System Theory, Control and Computing (ICSTCC), pp. 1 – 6, Oct. 2012

[11] A. Tanoto, U. Rückert, "Local Navigation Strategies for Multi-Robot Exploration: From Simulation to Experimentation with Mini-Robots", Procedia Engineering, vol. 41, 2012, pp. 1197-1203

Mechanical Design and Power Engineering
10.4028/www.scientific.net/AMM.490-491

Covering an Unknown Area with an RFID-Enabled Robot Swarm
10.4028/www.scientific.net/AMM.490-491.1157

Appendix B

CURRICULUM VITAE

Personal data

Name: Andres Puusepp

Date of birth: 28.11.1982

Place of birth: Estonia

Citizenship: Estonian

Education

2001 – 2006 Tallinn University of Technology,

MSc in Computer Science

1989 – 2001 Pärnu Koidula Gümnaasium

Language competence

Estonian Native language

English Fluent

Russian Basic

Professional employment

2007 – … Swedbank AS, developer

2005 – 2007 WebMedia AS (Nortal AS), developer

ELULOOKIRJELDUS

Isikuandmed

Nimi: Andres Puusepp

Sünniaeg: 28.11.1982

Sünnikoht: Eesti

Kodakondsus: Eesti

Hariduskäik

2001 – 2006 Tallinna Tehnikaülikool, informaatika magister

1989 – 2001 Pärnu Koidula Gümnaasium

Keelteoskus

Eesti keel Emakeel

Inglise keel Kõrgtase

Vene keel Algtase

Teenistuskäik

2007 – … Swedbank AS, arendaja

2005 – 2007 WebMedia AS (Nortal AS), arendaja

DISSERTATIONS DEFENDED AT TALLINN UNIVERSITY OF TECHNOLOGY ON *INFORMATICS AND SYSTEM ENGINEERING*

1. **Lea Elmik**. Informational Modelling of a Communication Office. 1992.

2. **Kalle Tammemäe**. Control Intensive Digital System Synthesis. 1997.

3. **Eerik Lossmann**. Complex Signal Classification Algorithms, Based on the Third-Order Statistical Models. 1999.

4. **Kaido Kikkas**. Using the Internet in Rehabilitation of People with Mobility Impairments – Case Studies and Views from Estonia. 1999.

5. **Nazmun Nahar**. Global Electronic Commerce Process: Business-to-Business. 1999.

6. **Jevgeni Riipulk**. Microwave Radiometry for Medical Applications. 2000.

7. **Alar Kuusik**. Compact Smart Home Systems: Design and Verification of Cost Effective Hardware Solutions. 2001.

8. **Jaan Raik**. Hierarchical Test Generation for Digital Circuits Represented by Decision Diagrams. 2001.

9. **Andri Riid**. Transparent Fuzzy Systems: Model and Control. 2002.

10. **Marina Brik**. Investigation and Development of Test Generation Methods for Control Part of Digital Systems. 2002.

11. **Raul Land**. Synchronous Approximation and Processing of Sampled Data Signals. 2002.

12. **Ants Ronk**. An Extended Block-Adaptive Fourier Analyser for Analysis and Reproduction of Periodic Components of Band-Limited Discrete-Time Signals. 2002.

13. **Toivo Paavle**. System Level Modeling of the Phase Locked Loops: Behavioral Analysis and Parameterization. 2003.

14. **Irina Astrova**. On Integration of Object-Oriented Applications with Relational Databases. 2003.

15. **Kuldar Taveter**. A Multi-Perspective Methodology for Agent-Oriented Business Modelling and Simulation. 2004.

16. **Taivo Kangilaski**. Eesti Energia käiduhaldussüsteem. 2004.

17. **Artur Jutman**. Selected Issues of Modeling, Verification and Testing of Digital Systems. 2004.

18. **Ander Tenno**. Simulation and Estimation of Electro-Chemical Processes in Maintenance-Free Batteries with Fixed Electrolyte. 2004.

19. **Oleg Korolkov**. Formation of Diffusion Welded Al Contacts to Semiconductor Silicon. 2004.

20. **Risto Vaarandi**. Tools and Techniques for Event Log Analysis. 2005.

21. **Marko Koort**. Transmitter Power Control in Wireless Communication Systems. 2005.

22. **Raul Savimaa**. Modelling Emergent Behaviour of Organizations. Time-Aware, UML and Agent Based Approach. 2005.

23. **Raido Kurel**. Investigation of Electrical Characteristics of SiC Based Complementary JBS Structures. 2005.

24. **Rainer Taniloo**. Ökonoomsete negatiivse diferentsiaaltakistusega astmete ja elementide disainimine ja optimeerimine. 2005.

25. **Pauli Lallo**. Adaptive Secure Data Transmission Method for OSI Level I. 2005.

26. **Deniss Kumlander**. Some Practical Algorithms to Solve the Maximum Clique Problem. 2005.

27. **Tarmo Veskioja**. Stable Marriage Problem and College Admission. 2005.

28. **Elena Fomina**. Low Power Finite State Machine Synthesis. 2005.

29. **Eero Ivask**. Digital Test in WEB-Based Environment 2006.

30. **Виктор Войтович**. Разработка технологий выращивания из жидкой фазы эпитаксиальных структур арсенида галлия с высоковольтным p-n переходом и изготовления диодов на их основе. 2006.

31. **Tanel Alumäe**. Methods for Estonian Large Vocabulary Speech Recognition. 2006.

32. **Erki Eessaar**. Relational and Object-Relational Database Management Systems as Platforms for Managing Softwareengineering Artefacts. 2006.

33. **Rauno Gordon**. Modelling of Cardiac Dynamics and Intracardiac Bio-impedance. 2007.

34. **Madis Listak**. A Task-Oriented Design of a Biologically Inspired Underwater Robot. 2007.

35. **Elmet Orasson**. Hybrid Built-in Self-Test. Methods and Tools for Analysis and Optimization of BIST. 2007.

36. **Eduard Petlenkov**. Neural Networks Based Identification and Control of Nonlinear Systems: ANARX Model Based Approach. 2007.

37. **Toomas Kirt**. Concept Formation in Exploratory Data Analysis: Case Studies of Linguistic and Banking Data. 2007.

38. **Juhan-Peep Ernits**. Two State Space Reduction Techniques for Explicit State Model Checking. 2007.

39. **Innar Liiv**. Pattern Discovery Using Seriation and Matrix Reordering: A Unified View, Extensions and an Application to Inventory Management. 2008.

40. **Andrei Pokatilov**. Development of National Standard for Voltage Unit Based on Solid-State References. 2008.

41. **Karin Lindroos**. Mapping Social Structures by Formal Non-Linear Information Processing Methods: Case Studies of Estonian Islands Environments. 2008.

42. **Maksim Jenihhin**. Simulation-Based Hardware Verification with High-Level Decision Diagrams. 2008.

43. **Ando Saabas**. Logics for Low-Level Code and Proof-Preserving Program Transformations. 2008.

44. **Ilja Tšahhirov**. Security Protocols Analysis in the Computational Model – Dependency Flow Graphs-Based Approach. 2008.

45. **Toomas Ruuben**. Wideband Digital Beamforming in Sonar Systems. 2009.

46. **Sergei Devadze**. Fault Simulation of Digital Systems. 2009.

47. **Andrei Krivošei**. Model Based Method for Adaptive Decomposition of the Thoracic Bio-Impedance Variations into Cardiac and Respiratory Components. 2009.

48. **Vineeth Govind**. DfT-Based External Test and Diagnosis of Mesh-like Networks on Chips. 2009.

49. **Andres Kull**. Model-Based Testing of Reactive Systems. 2009.

50. **Ants Torim**. Formal Concepts in the Theory of Monotone Systems. 2009.

51. **Erika Matsak**. Discovering Logical Constructs from Estonian Children Language. 2009.

52. **Paul Annus**. Multichannel Bioimpedance Spectroscopy: Instrumentation Methods and Design Principles. 2009.

53. **Maris Tõnso**. Computer Algebra Tools for Modelling, Analysis and Synthesis for Nonlinear Control Systems. 2010.

54. **Aivo Jürgenson**. Efficient Semantics of Parallel and Serial Models of Attack Trees. 2010.

55. **Erkki Joasoon**. The Tactile Feedback Device for Multi-Touch User Interfaces. 2010.

56. **Jürgo-Sören Preden**. Enhancing Situation – Awareness Cognition and Reasoning of Ad-Hoc Network Agents. 2010.

57. **Pavel Grigorenko**. Higher-Order Attribute Semantics of Flat Languages. 2010.

58. **Anna Rannaste**. Hierarcical Test Pattern Generation and Untestability Identification Techniques for Synchronous Sequential Circuits. 2010.

59. **Sergei Strik**. Battery Charging and Full-Featured Battery Charger Integrated Circuit for Portable Applications. 2011.

60. **Rain Ottis**. A Systematic Approach to Offensive Volunteer Cyber Militia. 2011.

61. **Natalja Sleptšuk**. Investigation of the Intermediate Layer in the Metal-Silicon Carbide Contact Obtained by Diffusion Welding. 2011.

62. **Martin Jaanus**. The Interactive Learning Environment for Mobile Laboratories. 2011.

63. **Argo Kasemaa**. Analog Front End Components for Bio-Impedance Measurement: Current Source Design and Implementation. 2011.

64. **Kenneth Geers**. Strategic Cyber Security: Evaluating Nation-State Cyber Attack Mitigation Strategies. 2011.

65. **Riina Maigre**. Composition of Web Services on Large Service Models. 2011.

66. **Helena Kruus**. Optimization of Built-in Self-Test in Digital Systems. 2011.

67. **Gunnar Piho**. Archetypes Based Techniques for Development of Domains, Requirements and Sofware. 2011.

68. **Juri Gavšin**. Intrinsic Robot Safety Through Reversibility of Actions. 2011.

69. **Dmitri Mihhailov**. Hardware Implementation of Recursive Sorting Algorithms Using Tree-like Structures and HFSM Models. 2012.

70. **Anton Tšertov**. System Modeling for Processor-Centric Test Automation. 2012.

71. **Sergei Kostin**. Self-Diagnosis in Digital Systems. 2012.

72. **Mihkel Tagel**. System-Level Design of Timing-Sensitive Network-on-Chip Based Dependable Systems. 2012.

73. **Juri Belikov**. Polynomial Methods for Nonlinear Control Systems. 2012.

74. **Kristina Vassiljeva**. Restricted Connectivity Neural Networks based Identification for Control. 2012.

75. **Tarmo Robal**. Towards Adaptive Web – Analysing and Recommending Web Users` Behaviour. 2012.

76. **Anton Karputkin**. Formal Verification and Error Correction on High-Level Decision Diagrams. 2012.

77. **Vadim Kimlaychuk**. Simulations in Multi-Agent Communication System. 2012.

78. **Taavi Viilukas**. Constraints Solving Based Hierarchical Test Generation for Synchronous Sequential Circuits. 2012.

79. **Marko Kääramees**. A Symbolic Approach to Model-based Online Testing. 2012.

80. **Enar Reilent**. Whiteboard Architecture for the Multi-agent Sensor Systems. 2012.

81. **Jaan Ojarand**. Wideband Excitation Signals for Fast Impedance Spectroscopy of Biological Objects. 2012.

82. **Igor Aleksejev**. FPGA-based Embedded Virtual Instrumentation. 2013.

83. **Juri Mihhailov**. Accurate Flexible Current Measurement Method and its Realization in Power and Battery Management Integrated Circuits for Portable Applications. 2013.

84. **Tõnis Saar**. The Piezo-Electric Impedance Spectroscopy: Solutions and Applications. 2013.

85. **Ermo Täks**. An Automated Legal Content Capture and Visualisation Method. 2013.

86. **Uljana Reinsalu**. Fault Simulation and Code Coverage Analysis of RTL Designs Using High-Level Decision Diagrams. 2013.

87. **Anton Tšepurov**. Hardware Modeling for Design Verification and Debug. 2013.

88. **Ivo Müürsepp**. Robust Detectors for Cognitive Radio. 2013.

89. **Jaas Ježov**. Pressure sensitive lateral line for underwater robot. 2013.

90. **Vadim Kaparin**. Transformation of Nonlinear State Equations into Observer Form. 2013.

92. **Reeno Reeder**. Development and Optimisation of Modelling Methods and Algorithms for Terahertz Range Radiation Sources Based on Quantum Well Heterostructures. 2014.

93. **Ants Koel**. GaAs and SiC Semiconductor Materials Based Power Structures: Static and Dynamic Behavior Analysis. 2014.

94. **Jaan Übi**. Methods for Coopetition and Retention Analysis: An Application to University Management. 2014.

95. **Innokenti Sobolev**. Hyperspectral Data Processing and Interpretation in Remote Sensing Based on Laser-Induced Fluorescence Method. 2014.

96. **Jana Toompuu**. Investigation of the Specific Deep Levels in *p*-, *i*- and *n*-Regions of GaAs p^+-*pin*-n^+ Structures. 2014.

97. **Taavi Salumäe**. Flow-Sensitive Robotic Fish: From Concept to Experiments. 2015.

98. **Yar Muhammad**. A Parametric Framework for Modelling of Bioelectrical Signals. 2015.

99. **Ago Mõlder**. Image Processing Solutions for Precise Road Profile Measurement Systems. 2015.

100. **Kairit Sirts**. Non-Parametric Bayesian Models for Computational Morphology. 2015.

101. **Alina Gavrijaševa**. Coin Validation by Electromagnetic, Acoustic and Visual Features. 2015.

102. **Emiliano Pastorelli**. Analysis and 3D Visualisation of Microstructured Materials on Custom-Built Virtual Reality Environment. 2015.

103. **Asko Ristolainen**. Phantom Organs and their Applications in Robotic Surgery and Radiology Training. 2015.

104. **Aleksei Tepljakov**. Fractional-order Modeling and Control of Dynamic Systems. 2015.

105. **Ahti Lohk**. A System of Test Patterns to Check and Validate the Semantic Hierarchies of Wordnet-type Dictionaries. 2015.

106. **Hanno Hantson**. Mutation-Based Verification and Error Correction in High-Level Designs. 2015.

107. **Lin Li**. Statistical Methods for Ultrasound Image Segmentation. 2015.

108. **Aleksandr Lenin**. Reliable and Efficient Determination of the Likelihood of Rational Attacks. 2015.

109. **Maksim Gorev**. At-Speed Testing and Test Quality Evaluation for High-Performance Pipelined Systems. 2016.

110. **Mari-Anne Meister**. Electromagnetic Environment and Propagation Factors of Short-Wave Range in Estonia. 2016.

111. **Syed Saif Abrar**. Comprehensive Abstraction of VHDL RTL Cores to ESL SystemC. 2016.

112. **Arvo Kaldmäe**. Advanced Design of Nonlinear Discrete-time and Delayed Systems. 2016.

113. **Mairo Leier**. Scalable Open Platform for Reliable Medical Sensorics. 2016.

114. **Georgios Giannoukos**. Mathematical and Physical Modelling of Dynamic Electrical Impedance. 2016.

115. **Aivo Anier**. Model Based Framework for Distributed Control and Testing of Cyber-Physical Systems. 2016.

116. **Denis Firsov**. Certification of Context-Free Grammar Algorithms. 2016.

117. **Sergei Astapov**. Distributed Signal Processing for Situation Assessment in Cyber-Physical Systems. 2016.

118. **Erkki Moorits**. Embedded Software Solutions for Development of Marine Navigation Light Systems. 2016.

119. **Andres Ojamaa**. Software Technology for Cyber Security Simulations. 2016.

120. **Gert Toming**. Fluid Body Interaction of Biomimetic Underwater Robots. 2016.

121. **Kadri Umbleja**. Competence Based Learning – Framework, Implementation, Analysis and Management of Learning Process. 2017.

122. **Andres Hunt**. Application-Oriented Performance Characterization of the Ionic Polymer Transducers (IPTs). 2017.

123. **Niccolò Veltri**. A Type-Theoretical Study of Nontermination. 2017.

124. **Tauseef Ahmed**. Radio Spectrum and Power Optimization Cognitive Techniques for Wireless Body Area Networks. 2017.

125. **Andre Veski**. Agent-Based Computational Experiments in Two-Sided Matching Markets. 2017

126. **Artjom Rjabov**. Network-Based Hardware Accelerators for Parallel Data Processing. 2017.

127. **Fatih Güllü**. Conformity Analysis of E-Learning Systems at Largest Universities in Estonia and Turkey on the Basis of EES Model. 2017

128. **Margarita Spitšakova**. Discrete Gravitational Swarm Optimization Algorithm for System Identification. 2017.