CHALMERS

Multi-Robot Distributed Coverage in Realistic Environments

Master of Science Thesis in
Computer Science: Algorithms, Languages and Logic

EMIL BRYNGELSSON

Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden 2016

Multi-Robot Distributed Coverage in Realistic Environments
EMIL BRYNGELSSON

The project work was carried out as a visiting student in the Distributed Intelligent Systems and Algorithms Laboratory (DISAL) at EPFL, Lausanne, Switzerland. With supervision by Alicja Wasik and José Nuno Pereira. DISAL is led by Prof. Alcherio Martinoli.

At Chalmers:
Examiner: Aarne Ranta
Supervisor: Peter Damaschke

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Cover: A group of MBot robots in the DISAL test lab at EPFL.

Abstract

A patrolling behavior is developed for a group of social robots used in a hospital. The robots interact with humans and are used for edutainment activities in the children's ward of IPOL, Portugal. Based on centroidal Voronoi coverage and extending existing ideas within robotic coverage, the behavior is scalable in the number of robots and robust to robot failures, allowing for exit and re-entry at run-time. Three types of experiments are performed and measured in terms of a cost function for coverage efficiency, and tested on different maps. Experiments are performed in simulation and in reality with robots in an office environment similar to the real hospital. The implementation works well in convex environments and one main challenge is to make it work in a nonconvex environment as well. The chosen approach uses virtual generating points, and other approaches are discussed with potential for improvements by using the geodesic distance measure or discrete graphs.

Acknowledgements

I first of all wish to thank my supervisor at EPFL, Alicja Wasik, for all the help and guidance during the project. I also thank José Nuno Pereira for co-supervision and providing additional input. Andreas Breitenmoser provided valuable ideas and comments from his work, which parts of this project are inspired by. Peter Damaschke, academic supervisor at Chalmers has been very helpful with the report writing process. Further thanks to Lorenzo Sarti, David Mansolino and Alessio Canepa for helping out with Webots errors and hardware problems. Thanks to Chalmers and EPFL for making possible my exchange studies, and this project in particular.

Emil Bryngelsson, Oslo, 2016-03-02

Contents

1 Introduction

Robots are increasingly becoming part of everyday life. Although industrial robots have been used for many years within production and automation, new classes of *social* robots are leaving the factories behind to serve the public. From the therapeutic Paro[1] robot, resembling a cuddly toy, to Pepper[2], *"the first humanoid robot designed to live with humans"*, social robots are designed to interact with humans within education, healthcare, tourism and many other fields[1], [2]. Two important technical aspects that separate many social robots from their industrial ancestors are a) independent mobility and b) human social interaction. Both of these functions are central to the **MOnarCH**[3] project, "Multi-Robot Cognitive Systems Operating in Hospitals", which this thesis is a part of.

MOnarCH is a collaborative research project funded within the EU-FP7 framework[4], including five European partner universities, a hospital and three private companies. The purpose is to develop socially aware robots for edutainment activities in the children's ward of the Portuguese Oncology Institute in Lisbon (**IPOL**) [5]. The robots are currently being used at the hospital, while continuous development is done by the partners. At École polytechnique fédérale de Lausanne (**EPFL**), the **DISAL**[6] group develops algorithms for cooperative patrolling, human-aware navigation, sensor fusion and interactive game playing.

1.1 Project aim

One planned functionality for MOnarCH is to have robots patrol the environment, searching for children who want to interact, as well as signs of

[1] http://www.parorobots.com
[2] https://www.aldebaran.com/en/a-robots/who-is-pepper
[3] http://monarch-fp7.eu
[4] http://ec.europa.eu/research/fp7/index_en.cfm
[5] http://www.ipolisboa.min-saude.pt
[6] Distributed Intelligent Systems and Algorithms Laboratory, http://disal.epfl.ch

unexpected events or emergencies. The purpose of this thesis project is to develop the navigational parts of this behavior. Based on recent and related research, the DISAL group wanted to apply Voronoi coverage (sec 2.2) for this task. The behavior is to be tested with up to 4 robots in reality and simulation, but must also scale well with larger numbers. The result of the implementation is a new software package which will become part of the MOnarCH repository.

The main research question is: *How can Voronoi coverage methods be applied for distributed coverage and patrolling in the MOnarCH project?*

1.2 Requirements and limitations

Four requirements were stated during planning, presented in order of priority for this project:

- Distributed calculation: No centralized controller may be used to direct or make decisions for the group. The main motivation for this is to create a robust system without a single point of failure. A single robot may fail because of low battery charge or other software and hardware errors, or it may be taken out of the group to perform other tasks, so we cannot allow one of the robots to direct the whole system.
- Allow changing group size at run-time: Robots may enter and exit the group during deployment, and the behavior should adapt to the current group size and continue without interruption and without requiring a restart or human intervention.
- Robustness to communication problems: Network quality may vary depending on the target environment. It is valuable to test the algorithm and routines developed for latency and packet loss.
- Nonobstructive behavior: For use in a hospital environment, it is important that the robots do not obstruct other activities or block paths and doorways for people. One example of when this may happen is when two robots try to pass a doorway simultaneously from opposite directions, they may block each other and cause a deadlock.

Some limitations are also stated, improving focus and leaving room for extended study in the future:

- No recharge: The robots use batteries that need recharging from time to time. This will not be accounted for in experiments and evaluations. With a run time of up to 3 hours on a full charge, real experiments will be conducted during a shorter time so that battery capacity is not a constraint. Monitoring the battery levels and sending robots to

recharge is also outside of the project, handled by a behavior manager software on a higher level.

- Fewer obstacles: The test environment is an office space at the university. The layout is similar to the target hospital environment, but with fewer obstacles and people moving around. Static obstacles such as walls, doors and furniture are present in both environments. Only walls are present in simulation. Although there are humans in the environment, there is no focus on human-aware navigation, which is a different part of the MOnarCH project.
- No added noise: For distributed systems in general and robots in particular, it is important to study system performance under less than perfect conditions. Two important factors are network communication (packet loss and latency), and navigation errors (due to imprecise localisation and/or odometry). When running simulations, it is possible to add arbitrary levels of noise for these factors, but this will not be done during the project. During real experiments, there will naturally be imperfect communication and positioning with the given hardware, but no focus is put on how this impacts performance.

2 Theoretical framework

This chapter introduces some theoretical knowledge and research on which the implementation is based. We start with a presentation of Voronoi diagrams in the plane, and based on this we proceed to Voronoi coverage methods used in robotics.

2.1 Voronoi diagrams

A Voronoi diagram[1] is a subdivision of a Euclidean space according to a given finite set of *generating points*[2], such that each generating point is assigned a *Voronoi cell* containing the space which is closer to this generating point than to any other. An example is given in Fig. 2.1. For the purpose of this thesis, only the 2-dimensional plane is interesting, but the Voronoi diagram can also be constructed in higher dimensions by the same principles. The following definition is given in [3, p. 148]:

> Denote the Euclidean distance between two points p and q by $\mathtt{dist}(p, q)$. In the plane we have
>
> $$\mathtt{dist}(p, q) := \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$
>
> Let $P := \{p_1, p_2, ..., p_n\}$ be a set of n distinct points in the plane; these points are the sites. We define the Voronoi diagram of P as the subdivision of the plane into n cells, one for each site in P, with the property that a point q lies in the cell corresponding to a site p_i if and only if $\mathtt{dist}(q, p_i) < \mathtt{dist}(q, p_j)$ for each $p_j \in P$ with $j \neq i$. We denote the Voronoi diagram of P by $\mathtt{Vor}(P)$.

Each Voronoi cell is the intersection of a number of half-planes, and therefore convex[4, p.20]. Each edge within a Voronoi diagram is equidistant to two of the generating points, and is called *Voronoi edge*. A Voronoi diagram has

[1] The term *Voronoi tessellation* is sometimes used synonymously.

[2] also referred to as *sites* or *generators*

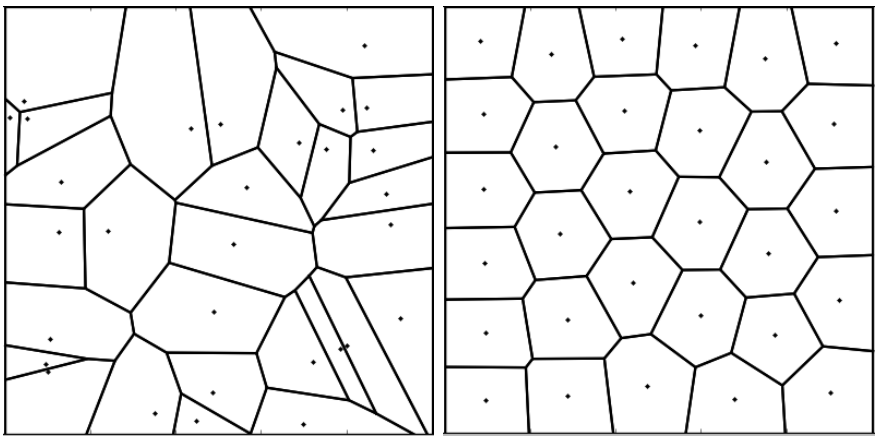


Figure 2.1: A Voronoi diagram (left) and a centroidal Voronoi tessellation (right), each with 30 generating points bounded by a square.

no natural outer bounds since the plane extends to infinity in all directions. However, for the applications within this project, it will always be used within a bounding polygon (a room or building) and illustrations will also include boundary lines which are not part of the Voronoi diagram itself.

There are different ways to calculate the Voronoi diagram from a given set of points in the plane, and Fortune's sweep line algorithm[5] is optimal, with complexity $O(n \log n)$ [3, p. 151].

Centroid and Area

The *centroid* is the center of mass of a shape or physical body. For a plane figure with uniform density, it is positioned in the mean (x, y) position of all points within the body. Given a closed, non-self-intersecting polygon P, defined by its vertices $\{(x_0, y_0), (x_1, y_1), ..., (x_{N-1}, y_{N-1})\}$, the area A and centroid (C_x, C_y) of P can be calculated as given in [6]:

$$A = \frac{1}{2} \sum_{k=0}^{N-1} (x_k \, y_{k+1} - x_{k+1} \, y_k)$$

$$C_x = \frac{1}{6A} \sum_{k=0}^{N-1} (x_k + x_{k+1})(x_k \, y_{k+1} - x_{k+1} \, y_k)$$

$$C_y = \frac{1}{6A} \sum_{k=0}^{N-1} (y_k + y_{k+1})(x_k \, y_{k+1} - x_{k+1} \, y_k)$$

2.1.1 Centroidal Voronoi tessellations

A centroidal Voronoi tessellation (CVT) is a type of Voronoi diagram where in each cell, the generating point has the same position as the centroid. An example is shown in Fig. 2.1. The CVT has a wide range of applications in science, some of which are described in [7]. In robotics, it is used in particular because a CVT configuration is a local minimum for the locational optimization cost function [6], [8] (see section 2.2.1) .

Calculating a CVT

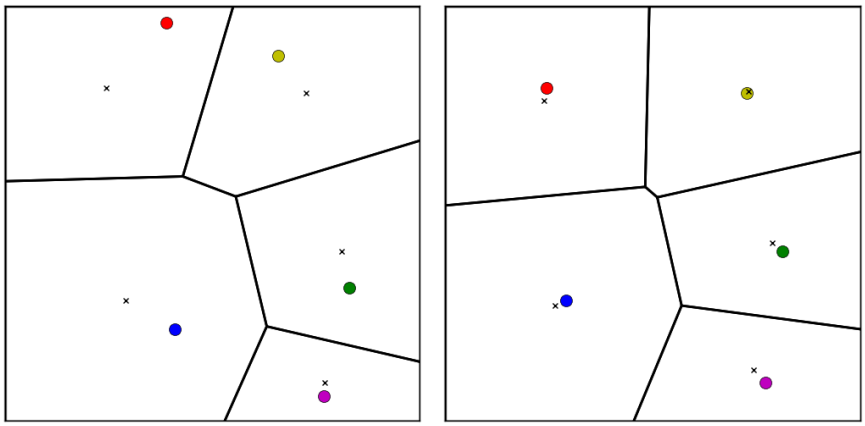
Some methods to calculate a CVT are given in [7], in particular **Lloyd's method**, an iterative algorithm which inspired the implementation in this thesis. Lloyd's method can be used to deterministically construct a CVT given a 2D plane and a set of starting positions[8]. The number of required iterations depends on the number of generators, their starting positions, and the termination condition. A sample termination condition might be "each generating point moved less than 0.1m in the last iteration". With a given bounding polygon P and k generating points, there can be several final states depending on the starting positions, as shown in Fig. 2.3. Lloyd's method is described below and also illustrated in Fig. 2.2

Algorithm 1 Lloyd's method, used to calculate a CVT

Given a polygon P and a set of k generating points at positions z_i
loop
  Construct the Voronoi diagram $\{V_i\}$ for the points $\{z_i\}$
  Compute the centroid c_i for each Voronoi cell V_i
  Set each point z_i to the position of its associated centroid c_i
  If this new set of points meets some convergence criterion, terminate.
end loop

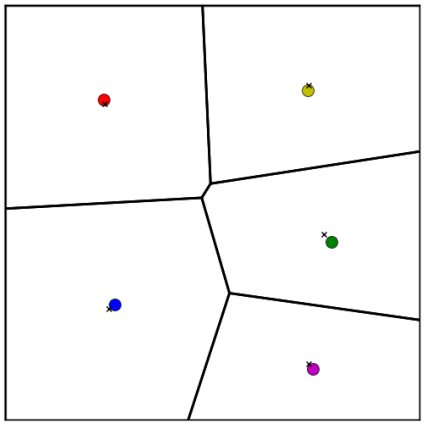
2.2 Robotic Coverage

Robotic coverage concerns movement planning with the purpose of covering an area, either by physically moving the robot or by positioning it such that its sensors can be used within the area. Some applications include cleaning robots sweeping a supermarket floor[9], demining robots scanning a field to detect and remove explosives[10], or inspector robots sensing for damage on jet turbine blades[11]. Coverage methods can be further classified according to the type and dimensions of free movement, the number of robots involved,

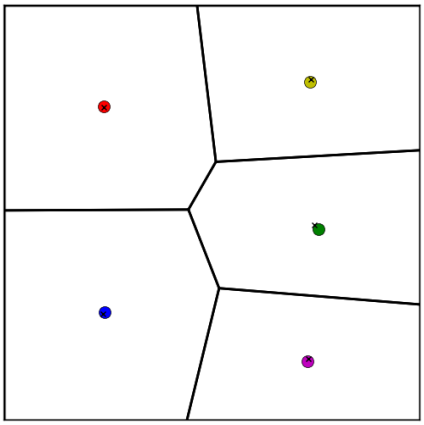


(a) Start configuration: t = 0

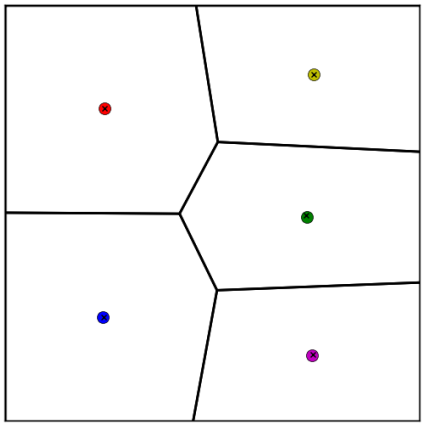
(b) t = 1



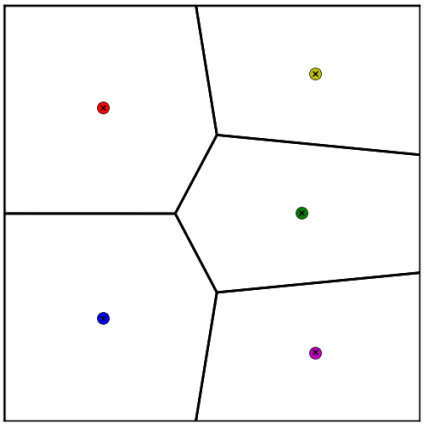
(c) t = 2



(d) t = 5



(e) t = 10



(f) t = 20

Figure 2.2: Lloyd iteration with 5 generators (colored circles) starting at random locations. The centroid of each Voronoi cell is marked with an x.

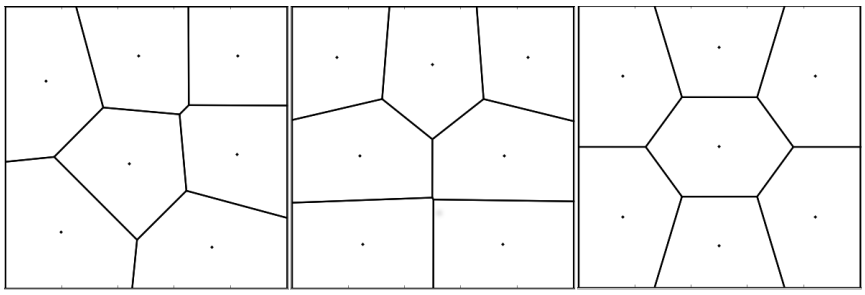


Figure 2.3: Three different CVT's resulting from 7 generating points on a square.

and other factors. Choset[12] provides an overview and classification of coverage algorithms for mobile robots in the plane, up to 2001.

Multi-robot systems generally rely on distributed computation and communication, rather than centralized. A distributed model may be more difficult to implement than a centralized one, but provides advantages for scalability and robustness [6], and avoids having a single point of failure. In some applications such as swarm robotics and sensor networks, a distributed model is often the only option due to limited communication range, battery and processing power.

When performing coverage in two dimensions, one common approach is to calculate a *cellular decomposition* of the plane into polygon cells, and then perform coverage in each cell [12]. With multiple robots, part of the problem is to assign cells to robots in an efficient way.

2.2.1 Voronoi coverage

Voronoi coverage [6] is one approach used with multiple robots where the target space is divided between the robots by forming a Voronoi diagram based on their positions. Each robot is thus assigned one Voronoi cell to cover. To divide the space evenly, a centroidal Voronoi tessellation is often desirable, and some application of the Lloyd algorithm can be used to achieve this [6]. Since robots cannot instantly move from one position to another, the Lloyd algorithm may need to be adapted to the use case at hand and the possible movement (sec 3.2.3).

Cost function

When performing Voronoi coverage, the locational optimization cost function can be used to measure efficiency of coverage [6], [8]. A centroidal Voronoi

tessellation provides a local minimum for the cost function, expressed as follows:

$$\mathcal{H}(P) = \sum_{i=1}^{n} \mathcal{H}(p_i) = \sum_{i=1}^{n} \int_{V_i} f(D(p_i, x))\phi(x)dx$$

With P the set of generating points p_i, the cost $\mathcal{H}(P)$ is calculated by summing $\mathcal{H}(p_i)$ for all i. Each generating point p_i has a Voronoi cell V_i, and the cost calculation for p_i is done by integrating over V_i the function $f(D(p_i, x))\phi(x)$. The x are the points in the Voronoi cell and the cost calculation has 3 parts:

- $\phi(\cdot)$ is a density function, which can be used if different parts of the space have different importance or weight. If no such weighting is needed, a uniform density function $\phi(\cdot) = 1$ can be used[8].
- $D(\cdot)$ is the distance function between the generating point p_i and a given point x in V_i. A default choice is the Euclidean distance function, but the geodesic distance is also a sensible option, in particular if the environment is nonconvex [8].
- $f(\cdot)$ is a function to account for sensor reliability of a robot. With increased distance, sensors are less reliable and therefore the cost should increase [8]. The function should be smooth and nondecreasing[6].

For more details on the use of the cost function, see the implementation part in sec. 3.2.2.

2.2.2 Nonconvex environments

One problem with a simple CVT algorithm is that it requires a convex and obstacle-free environment to ensure that movement is always possible to the centroids[13], otherwise it may not converge to a steady state. Most real applications in robotics take place in nonconvex environments with walls and other obstacles, which requires adaptations or different approaches. Different types of environments, in 2D or 3D, as well as different types of robots, movement and sensors, create a large number of related problems with their own sets of solutions. Breitenmoser et al [8] replace the robots with *virtual generators* to ensure convergence of the CVT algorithm, and then use projections from virtual generators to the accessible space to solve situations where the target is unreachable. In [14] Bhattacharya et al create a Voronoi coverage based on the *geodesic* distance rather than the Euclidean, ensuring that each robot is assigned a connected cell where it can reach all the points without moving into the cell of another robot.

2.2.3 Robotic Patrolling

Multi-robot systems can be used for surveillance and perimeter control, extending the capabilities of human agents or replacing them altogether. One important motivation is safety, since robots can be deployed in areas that are risky or dangerous for humans to work in[15]. Another benefit is economical, which applies more to this project: the use of robots enables more frequent and extensive patrolling than might be affordable if done by human personnel.

The goal of patrolling is to repeatedly visit a set of positions (waypoints) and often to minimize the downtime between visits. Portugal [16] provides an overview of patrolling algorithms, describing different strategies and optimization criteria. Among these are the *cyclic* strategy where the each robot in the group visits all the waypoints, compared to the *partition* strategy, where each robot is solely responsible for its own subset of waypoints. Performance evaluation is often based on *vertex idleness* (the downtime between visits) and one may want to minimize the average idleness, or minimize the maximum idleness for any vertex. Another suggested goal is to make the frequency as uniform as possible between vertices, with the downside that this becomes predictable and may be taken advantage of by an intruder [16].

3 Methodology

This chapter introduces the hardware and software tools used during the project, then it describes the implementation of algorithms and other main parts of the project.

3.1 Hardware and Software

The hardware and software tools available were mostly predetermined by what was already in use in the MOnarCH project. This section briefly describes the robots and the software used during development.

3.1.1 The Mbot robot

The robot used in the project is named **Mbot**, designed and built by Portuguese firms Idmind [1] and Selftech [2] in collaboration with the MOnarCH consortium. There are two versions: the socially oriented (SO) and perception oriented (PO), built on the same design and looking almost identical from the outside. However, the SO version is more complex and contains hardware that the PO lacks, for example a touch monitor, Kinect camera and AAXA P300 pico projector used for human interaction. Both versions have an onboard computer used for navigation control (PC-NAV), and the SO version also has a human-robot interaction computer (PC-HRI). The PC-NAV runs Ubuntu Linux 12.04 LTS and ROS Hydro, with the mbot-ros package[3].

Their movement is omnidirectional, using 4 mechanum wheels independently controlled by 4 motors. For obstacle detection, each robot has two laser rangefinders (LRF) with a 5 m detection range, and 12 sonar sensors placed in a ring around its body. It also contains a wireless router for network

[1] www.idmind.pt
[2] www.selftech.pt
[3] https://selftech.com/monarch/

communication. The robot runs on batteries and is capable of docking for recharge without human intervention, an important requirement for autonomous operations over long time periods.

In this thesis project, only the SO robot and its PC-NAV functionalities were used. A summary of information about the robot is given in Table 3.1, for full details see [17].

3.1.2 ROS

The Robot Operating System (ROS)[4] is a set of tools, libraries and conventions used to develop robot software for different platforms. It is an open-source, collaborative project dating back to the 2000's, today maintained by the Open Source Robotics Foundation[5]. ROS has been adopted on many levels and is used by hobbyists as well as educators and within industrial, commercial applications. The MOnarCH project uses the 7th release, Ros Hydro from 2013. The latest release was the 9th: Ros Jade Turtle from May 2015.

actionlib

Actionlib[6] is a ROS package which provides an interface to create and request tasks for a robot. Some use examples are (1) movement to a target coordinate or (2) performing a laser scan and returning the result. It uses a client-server architecture where the client initiates an action by sending a *goal* message to the server. The server may provide *feedback* messages in real time on how the action is proceeding, and finally a *result* message if successful. An action can also be *preempted* (canceled before being finished) by either sending a *cancel* message or by sending a new goal to the same server. Within this thesis project, the actionlib interface and **axclient** application were used together with the *move_base* actionserver to direct movement.

move_base

Move_base[7] is a ROS package, built as an actionserver that gives access to the navigation stack. It works as an interface between the application and the robot's path planner and hardware. In this project, an actionclient is created in the application running on PC-NAV, and when the CVT algorithm

[4] www.ros.org
[5] www.osrfoundation.org
[6] http://wiki.ros.org/actionlib
[7] http://wiki.ros.org/move_base

has calculated a new centroid or movement target, the (x, y) coordinates are sent as a move_base goal to the actionserver. If the target coordinate is feasible according to the environment map, the robot will attempt to move there and take care of path planning and obstacle avoidance.

navigation_by_target (NBT)

This is a ROS package built within MOnarCH. It uses an actionserver to navigate by continously following a target coordinate which may be updated in real time. It creates a smoother movement and often reaches closer to its target compared to move_base. NBT works well in open spaces but lacks the path planning around obstacles that move_base has. It is used in the convex coverage case where it performs better than move_base, but cannot be used in an environment with obstacles and walls.

3.1.3 Webots

Webots is a simulation software developed by Cyberbotics[8], used for education and professional development. It can realistically simulate a wide range of hardware sensors, actuators, physical interactions as well as radio communication, sound and light in three dimensions.

Webots has been extensively used in this project for running simulations and navigation testing in 3D replica of the EPFL and IPOL environments. Testing can be quickly set up and done safely thanks to the software simulation, causing no wear on the robots, no hardware downtime, and no competition with other users of the robots and the lab room. Webots integrates well with ROS so that the code developed and tested with Webots can be directly transferred and run on the real robots without modification.

3.1.4 Situational Awareness Module

The Situational Awareness Module (SAM)[18] is a software tool developed within MOnarCH to handle communication between ROS nodes. It is built on top of the ROS publish-subscribe architecture and adds some functionality such as enforced identification of subscribers, limits the number of publishers to 1 per communication channel, and enabling multi-master communication required for developing group behaviors for MBot robots. SAM was used in this thesis project for two purposes: (1) communicating robot positions asynchronously over the *tfPose* slot and (2) communicating robots' internal states through message broadcast, on a new slot set up for this behavior.

[8] www.cyberbotics.com

Size (H x W x L)	102 x 57 x 67 cm
Weight	49 kg
Battery autonomy	>3 hours
Max velocity	2.5 $\mathrm{m/s}$
Acceleration	1 $\mathrm{m/s^2}$
Emergency stop acceleration	3.3 $\mathrm{m/s^2}$

Table 3.1: Basic facts about the SO version of the mbot robot.

3.2 Implementation

This section describes how central parts of the project were implemented. We start with the computation of the Voronoi diagram and the cost function, then proceed with the algorithms used for CVT and patrolling. Finally, the use of maps and waypoints are explained.

Most of the code was written in Python 2.7 and makes use of the packages *Matplotlib, NumPy, SciPy* and *Shapely*. A minor amount of code was also written in C++ because parts of the MOnarCH codebase is in C++.

3.2.1 Voronoi diagram computation

The desired implementation should be distributed and function without central planning. It turns out that we do not need to calculate the complete Voronoi diagram in one place, but it is sufficient if each robot can calculate its own Voronoi polygon and centroid, to decide where to move next. Knowing the static boundary and the positions of the other robots, each robot can calculate the Voronoi edges and thereby its own polygon by iteratively "cutting" pieces from the bounding polygon. This algorithm was devised during the project and named "**polygon cutting**", see Algorithm 2 and Fig 3.1.

3.2.2 Cost function

Recall the cost function presented in sec. 2.2.1. For this project, the following choices were made:

- $\phi(\cdot) = 1$: a uniform density function is used since no weighting of different areas is done.
- $\mathsf{D}(\cdot)$ is set to the Euclidean distance function.

Algorithm 2 The polygon cutting algorithm, used to calculate a Voronoi polygon

With r^* the current robot,
$R = \{r_1, r_2, ...\}$ the list of other robots, sorted by distance
$P \leftarrow \partial\Omega$ the static (convex) boundary
for all r_i in R **do**
  calculate l, the perpendicular bisector of (r^*, r_i)
  intersect l, with P, dividing P into p_1, p_2
  determine whether r^* is located in p_1 or in p_2
  $P \leftarrow p_x$ which contains r^*
end for
P is now the Voronoi polygon of r^*

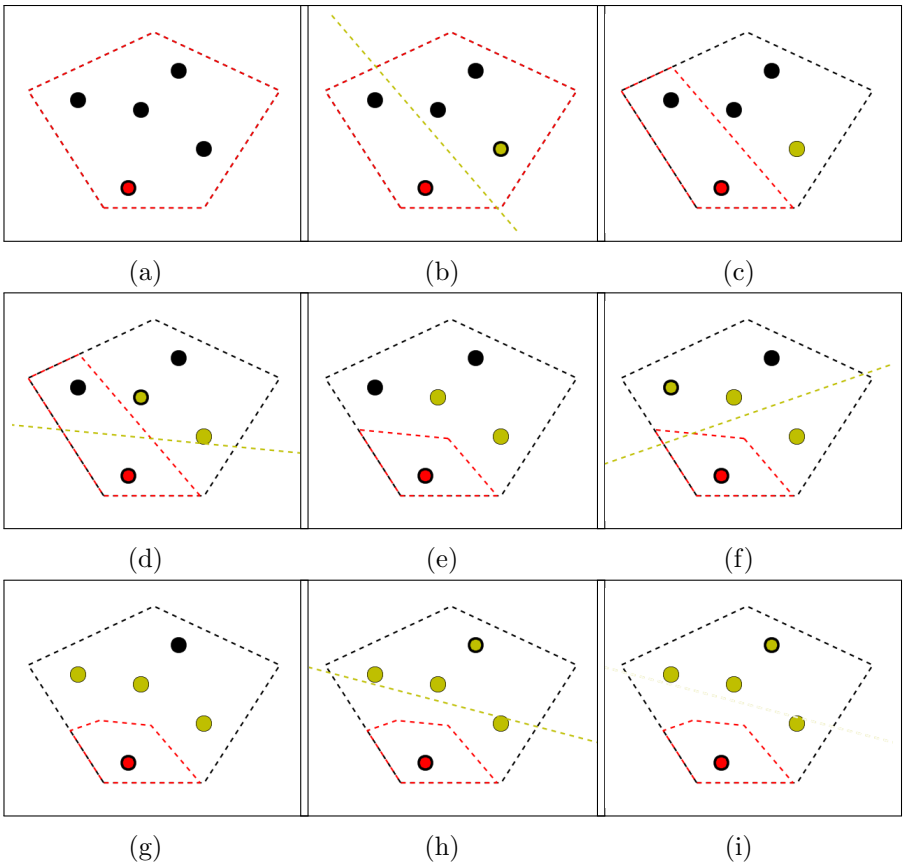


Figure 3.1: Polygon cutting algorithm illustrated. Starting with the full room and the nearest generator, the Voronoi edge is used to cut away part of the polygon, then repeated for each generator. When finished, the remaining polygon is the Voronoi cell of the red generator.

- $f(\cdot)$ the sensor reliability function is set to $f(x) = x$, which simplifies the cost calculation.

With these choices, the cost function becomes:

$$\mathcal{H}(P) = \sum_{i=1}^{n} \mathcal{H}(p_i) = \sum_{i=1}^{n} \int_{V_i} D(p_i, x)dx$$

In this implementation, a numerical integration is done to calculate the cost for a given Voronoi diagram. The space is divided into square cells representing the points x, and the distance is calculated from the center point of each square to the nearest robot p_i (see Fig. 3.2). The sum of these distances is then divided by the total number of squares so that the cost value represents the average distance in meters from any point to the nearest robot.

In some of the following experiments, the *normalised* cost per robot is mentioned, which has been calculated as the sum of distances in the robot's Voronoi cell, divided by the total number of squares in the full space. In this way, the normalised costs per robot add up to the total cost, indicating how big a fraction each robot covers, but the cost per robot does not represent the average distance within its Voronoi cell in the way the total cost does for the whole system.

The cost can be calculated with higher precision by making the squares smaller, with the tradeoff of increased computation time since the number of squares increase. For cost calculations in the experiments made, the spatial resolution (square side length) was set to values between 5 cm and 30 cm.

3.2.3 Lloyd iteration with robots

The Lloyd algorithm, adapted to create a CVT with robots, is implemented as described in Algorithm 3. It is used for the convex cases of experiments 1 and 2 (see Chapter 4). The third experiment takes place in a nonconvex environment which requires a different approach. The implementation here is inspired by Breitenmoser et al [8] and the use of virtual generators. A convex hull is formed around the target nonconvex area (see sec 3.2.4) and Algorithm 3 is run but with a **virtual generator** moving instead of each robot. The virtual generators only exist in the software and are allowed to move to any coordinate in the space, even through walls and inside obstacles. This resembles the original Lloyd algorithm and guarantees convergence. When the CVT has formed, each robot "owns" the Voronoi cell that corresponds to its virtual generator, and is responsible for patrolling the waypoints therein.

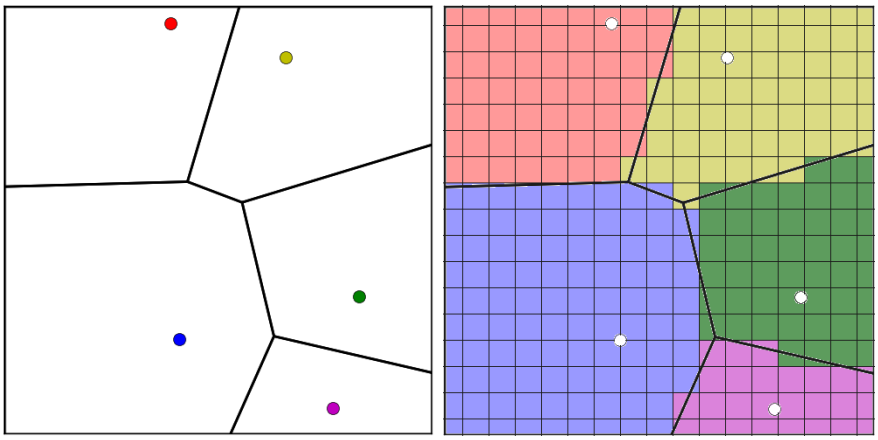


Figure 3.2: Numerical integration to compute the value of the cost function: a Voronoi diagram (left) and a division into square cells (right).

Algorithm 3 CVT algorithm running on each robot

loop
  With $\mathsf{P} = \{\mathsf{p}_1, \mathsf{p}_2, ...\}$ the positions of robots (updated asynchronously)
  Compute my Voronoi polygon by Algorithm 2
  Compute the position of the centroid C
  Send command to move to C
  Wait for 250ms while moving towards the target
end loop

3.2.4 Maps

The robots use maps as part of their navigation system. A map format existed within MOnarCH prior to this thesis, and it was decided to use the same format in this project.

Each map consists of one or more (YAML, GIF) file pairs, depending on the functionality required for a given use case. The YAML file is a text file containing the name of the corresponding GIF file, a coordinate transform, and grayscale color data to tell the navigation system what is allowed terrain or not. It may also contain information on zones in the map, if applicable. Zones are defined by a (string, integer) tuple, where the string is the name of the zone, and the integer is the color index for this zone as drawn in the corresponding GIF image. An example is given in Code 1 and Fig. 3.3a.

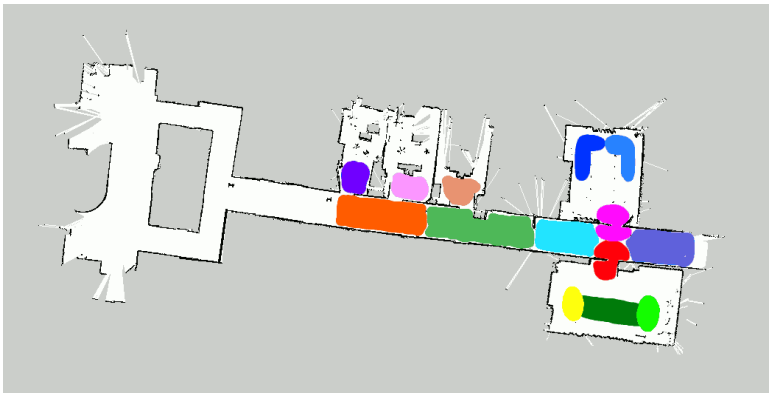
```
image: EPFL-sections.gif
resolution: 0.050000
origin: [0.000000, 0.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
zones:
  corridor1: 9
  corridor2: 11
  corridor3: 15
  corridor4: 7
  labcentre: 6
  labboard: 18
  labdesk: 14
  labdoor: 4
  kitchenboard: 8
  kitchencup: 3
  kitchendoor: 5
  office1: 12
  office2: 13
  office3: 2
```

Code 1: Example of a YAML file containing MOnarCH map data, corresponding to Fig. 3.3a.

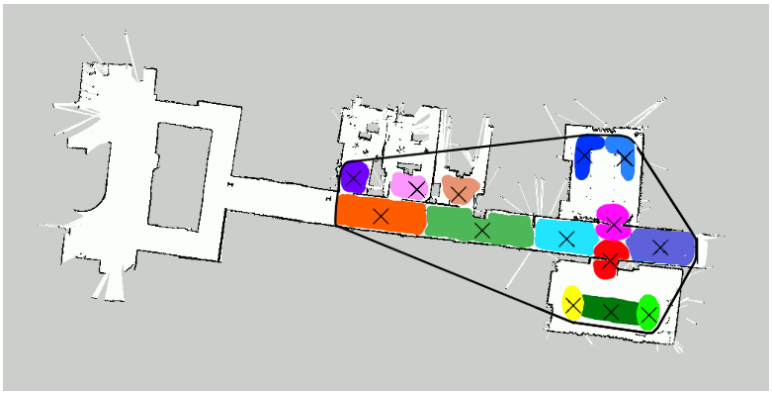
Map processing

Given the existing map format, new development was required to make it useful with the new behavior. The behavior needs two things to work: a space to divide using CVT, and a set of waypoints to patrol. The map files are read as input and used as follows.

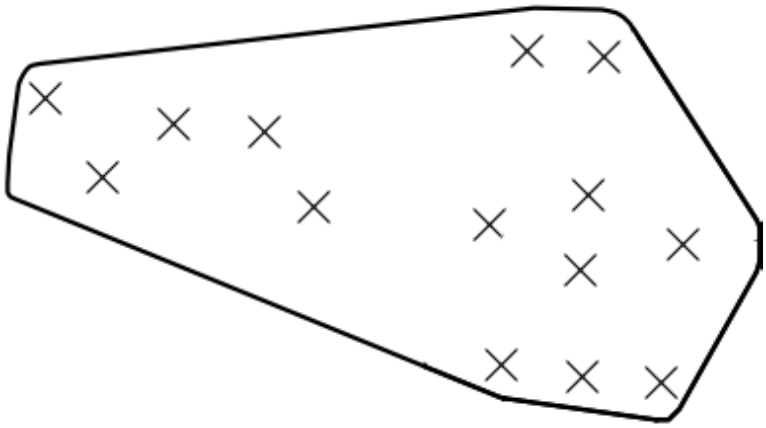
Waypoints are created from the zones of the map. The GIF file is read and for each zone, all the pixels of the associated color are read. Their mean



(a) Sample map GIF from MOnarCH, corresponding to the YAML data in Code 1.



(b) The waypoints (black X) and convex hull (black line) overlaid on the map image.



(c) The resulting convex hull used for CVT and waypoints for patrolling.

Figure 3.3: Sample map and processing into a convex hull and waypoints.

x and y positions are calculated and used as a waypoint coordinate (see coordinates overlaid on the map in Fig. 3.3b). This means that 1 pixel of each color would be enough for creating waypoints, but it is still useful to draw a bigger coloured shape to make the zone clearly visible for a human working with the map GIF.

A convex hull is calculated from all the zones' pixels together. This gives a convex polygon boundary which can be used for the CVT algorithm. It also excludes unused space, such as the white region on the left half of Fig. 3.3a, reducing the risk of a CVT division where one or several robots would have no waypoints in their respective Voronoi cells. The resulting hull and waypoints are displayed in Fig. 3.3c.

The coordinate transform (origin and scale) in the YAML file are used to calculate real world coordinates from the pixel positions in the image. The real world coordinates are sent to the robot for navigation targeting.

3.2.5 Greedy patrolling algorithm

Given a set of waypoints on the map, robots need instructions on how to patrol. A simple, greedy algorithm was used: see Algorithm 4. Since we use a division of the space into Voronoi cells, the strategy falls into the *partitioning* category (sec 2.2.3): each robot patrols its own unique subset of waypoints.

To prevent the robot from getting stuck, a timeout is used which allows the robot to skip a waypoint if not reached within the given time. Unreachable waypoints may be permanent if the map has been incorrectly drawn, or temporary in cases such as someone closing a door which is later opened again. When the robot is unable to reach a waypoint for 120 seconds, it will skip this waypoint in the current iteration, but it will try again the next time.

3.2.6 Communication and dynamic group size

Communication between the robots is done via SAM (sec. 3.1.4), over communication channels called *slots*. Each SAM slot can be set up with one publisher and multiple subscribers. Robot positions are sent on the tfPose slot, which is already in place within MOnarCH, and by letting each robot subscribe to the tfPose slot of every other robot, they asynchronously receive updated information on each other's positions. When using virtual generators for CVT in the nonconvex case, the position of the virtual generator needs to be communicated as well. This was done by creating another slot, named *DistributedCoverageVirtualCoord*, where each robot publishes

Algorithm 4 Greedy patrolling algorithm

With W the list of waypoints, W' an empty list, R the current robot
loop
  while W is not empty **do**
    Calculate the euclidean distance from R to each waypoint $w \in W$
    Select w^* the waypoint with the shortest distance
    Remove w^* from W and insert w^* into W'
    Send command to move to w^*
    Start a timer of 120 seconds
    Wait until timer runs out or R is within 0.25 m of w^*
  end while
  let $W \leftarrow W'$
  let $W' \leftarrow [\,]$
end loop

its virtual generator position and subscribes to read the virtual positions of the others.

A third slot *"DistributedCoverageRobotState"* was created and used to communicate status codes between the robots. There are three status codes representing the internal state of a robot: STATE_CVT, STATE_CONVERGE and STATE_PATROL. During CVT each robot continously broadcasts the STATE_CVT code, and when a robot (or virtual generator) reaches its target, it broadcasts STATE_CONVERGE instead. When they all broadcast STATE_CONVERGE for 30 time periods (30/4 seconds), they will switch to STATE_PATROL. A fourth possibility is that a robot does not broadcast any status code at all, which may happen if it loses network contact or otherwise fails. When no status is received for 40 periods (10 seconds), that robot will be dropped from the CVT configuration, and if the group is currently patrolling, it will switch back to CVT to create a new configuration with the smaller group size. Similarly, if a robot has been offline for a while and then starts broadcasting again, if the group is patrolling it will switch back to CVT to recreate the Voronoi coverage with the new group member. An illustration of the state machine behavior is given in Fig. 3.4.

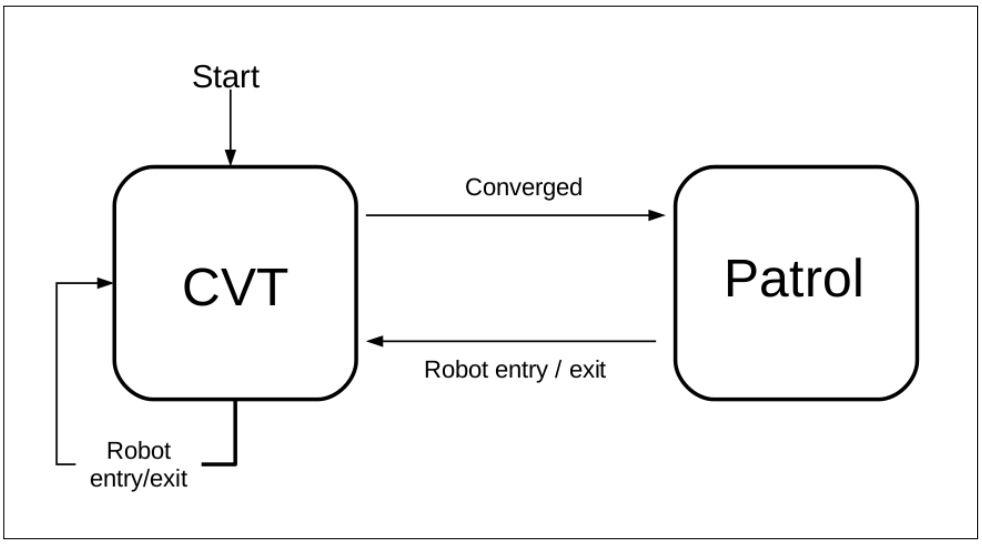


Figure 3.4: Illustration of the state machine. The start event takes care of initial one-time needs such as map processing, then the behavior alternates between CVT and Patrol. If the group size does not change, robots will patrol indefinitely once they reach this state.

4 Experiments

The behavior was developed step by step with a series of experiments used to test and validate different parts. These experiments are described in this chapter, with results and analysis from each.

4.1 Centroidal Voronoi coverage in a convex space

As a first milestone, it was decided to try and achieve centroidal Voronoi coverage in a convex space. After many simulations with Matplotlib and Webots, this is the first experiment on real robots, serving to confirm that the CVT implementation, SAM communication, and movement all work in reality, together.

This experiment has several simplifications when compared to the end goal, namely:

- The environment is a single, convex room instead of a nonconvex space with many rooms.
- No patrolling is done, only static coverage.
- No need to handle robot exit and entry during the experiment.

The experiment was carried out by choosing starting positions for the robots, running the CVT algorithm until convergence and storing robot positions for each time frame as output data. This experiment was performed in the DISAL lab room, which is rectangle shaped with a size of approximately 5×9 m. From the digital maps used by the robots, the room corners were manually approximated to [3.3, 6.5], [8.2, 5.8], [8.8, 14.5], [4.2, 14.8], and input as polygon bounds to the program.

While running, each robot outputs the system time and current position 4 times per second, and these data are then used in post-processing to calculate the cost function at each time frame. One representative run with 3 real robots was recorded on video and examined in more detail, with start

positions lined up at one end of the room. A further 40 experiments were run in Webots simulation to gather data for an aggregate view. The 40 simulations were split into 10 each for a group size of 1, 2, 3 and 4 robots, with random start positions to reduce bias in the results.

4.1.1 Results

The recorded experiment with 3 real robots took approximately 12 seconds from start to convergence, determined visually by when the robots were no longer moving. Fig. 4.1 shows images from the video recording of the real run, with corresponding Voronoi diagrams and path traces of robot movement. A plot of the cost function calculated for the whole system and per robot is shown in Fig. 4.2. The cost value starts at 2.9 and decreases to a final value of 1.5.

The costs calculated from the 4×10 simulations are aggregated and shown in Fig. 4.3. For each group size, the average cost of the 10 experiments is plotted with a bold colored line, and the minimum and maximum at each time step are plotted as borders of the area of the same color. The simulated experiments took up to 14 seconds to converge, depending on the number of robots and their random start positions. There is a large spread in intial cost values for different starting configurations, but the ranges of final values are pretty tight, with mean cost values of 2.5, 1.7, 1.5 and 1.3 respectively for a group size of 1, 2, 3 and 4 robots.

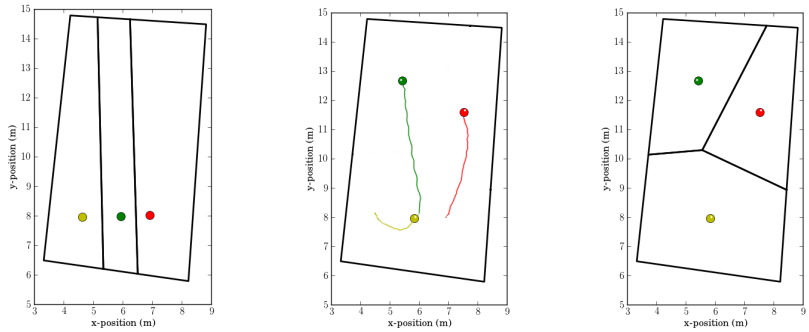
4.1.2 Discussion

Both experiments show the cost function decreasing gradually over time as robots move, then movement stops and we arrive at a minimum cost value. This is expected and indicates that the CVT/Lloyd implementation works as it should.

From the 4×10 plot, we observe that the final cost becomes lower the more robots we use. This confirms the idea that more robots can cover the room more efficiently. From the overlapping start values, we see that it is possible to configure 4 robots less efficiently than 3, or even 3 robots with less efficiency than 1. But in the final states, the costs rank up in reverse order to the number of robots used. The marginal improvement is largest from 1 to 2 and decreases further for each robot we add: we have diminishing returns.

It can also be observed that for the cases of 1 or 2 robots, the cost function converges to a single value for all 10 experiments, whereas for 3 and 4 robots, there is a spread of final cost values. This is explained by observing the final Voronoi diagram of the experiments: with 1 or 2 robots it seems empirically

(a) Initial state (b) Middle state (c) Final state



(d) Initial state (e) Path trace (f) Final state

Figure 4.1: Experiment 1, real run. Images from the video recording (a,b,c), corresponding Voronoi diagrams (d,f) and path trace (e).

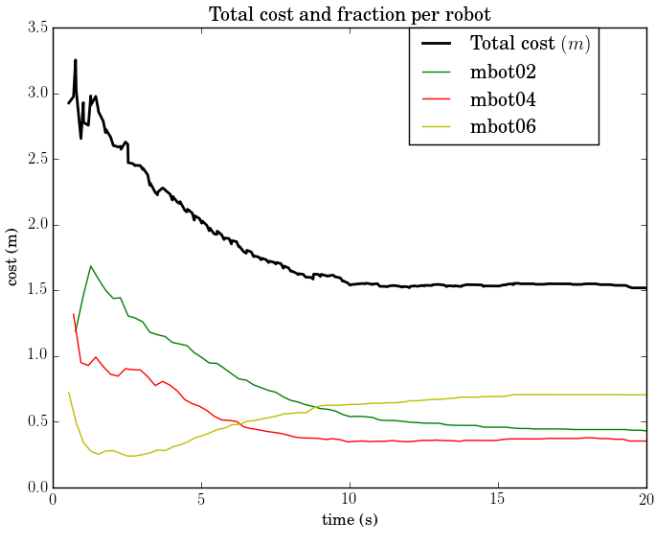


Figure 4.2: Experiment 1, real run. Cost plot showing the cost for each robot in different colors, and the sum cost in black.

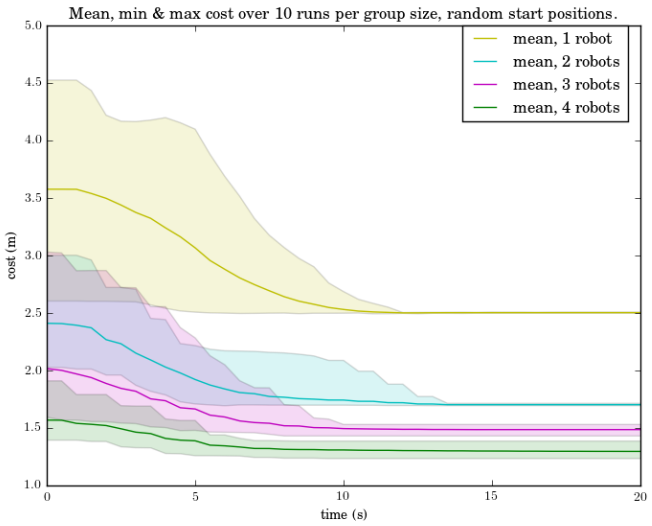


Figure 4.3: Experiment 1, simulation. Aggregate cost plot from the 4×10 experiments with random start positions and changing group size.

that there exists just one final configuration where all 10 experiments end up. For 3 or 4 robots, there are several different final states which all are local minima of the cost function, but they are not equally good. Some of this spread occurs because there is a minimum distance below which the real robots will not move when given a new target. The spread is smaller in a software-based calculation model with perfect positioning and millimeter-scale movement.

The running times are not fully comparable between simulation and reality, because simulation speed varies with CPU load, which mostly depends on the number of robots in the experiment. More robots generally means a higher CPU load and a slower simulation speed, resulting in simulation times higher than they should be in reality.

The simulation is run on a single computer with one system clock, while in the real experiment each robot has its own system time. This initially caused problems for the cost calculation time frames since the robots' clocks differed by as much as 13 seconds. They were synchronized using the Ubuntu NTP tool, and any remaining difference between them (on the order of 0.1 seconds) is considered too small to be significant for the experiment.

From the aggregate values, we can see that convergence generally happens earlier as we increase the number of robots. This should be expected since more robots fill the room to a larger extent, no matter how they are placed, and the travel distance to a converged state should be shorter on average.

The effect is probably even larger in reality than it seems from these plots, since the simulation speed decreases with more robots; without this bias the time should be even shorter than reported here.

The cost plot and final Voronoi diagram from the real experiment show that the yellow robot ends up covering a disproportionate fraction of the space: more than the green or red robot does. Repeat experiments on different room shapes indicate that the shape of the room determines how even the distribution can become.

To summarize, the first experiment was a success: it behaves largely as expected and confirms the functionality of fundamental parts of the implementation.

4.2 Dynamic entry and exit during coverage

The second experiment serves to test the dynamic entry and exit functionality during coverage. The environment is still convex during this experiment, but a different room was used. Many variants of this experiment were run in simulation, and the real experiment was performed with 3 robots in the lab corridor and recorded on video. The corridor is approximately rectangle shaped with side lengths 2.5 and 24 meters. The coordinates used are [8.5, 5.7], [10.8, 5.2], [13.7, 29.2], [11.2, 29.4]. Like the lab room, the corridor has some obstacles and is not truly convex, but these obstacles are situated along the walls in such a way that they do not impede the movement of the robots, and therefore it can be considered convex for the purpose of the experiment.

The three robots were placed in the southern half of the corridor roughly 4 m apart (see Fig.4.5a). The CVT algorithm was run until convergence, then one of the robots was manually deactivated and moved out of the environment. After a 10 second delay, the remaining two robots reconfigure and create a new CVT on their own. They are again given time to converge. With a CVT of 2 robots, the 3rd robot is moved back into the environment and activated again. The three robots create a CVT together and we end up with the same configuration as we started with. During this experiment, as in the previous one, the robot positions are output at 4 Hz and this data is then used in post-processing to calculate the cost function at every time step of the experiment, with the cost of the whole system and the cost for each robot plotted on a graph.

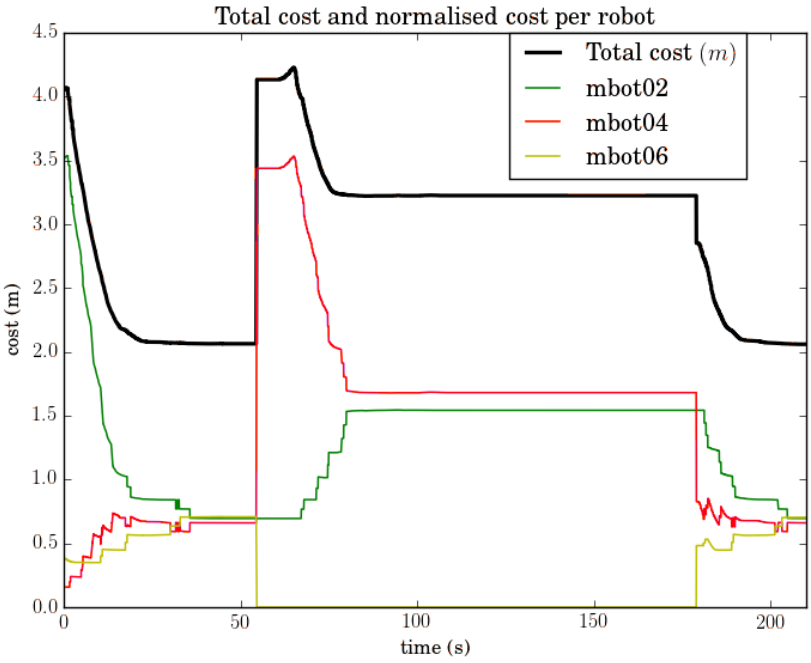


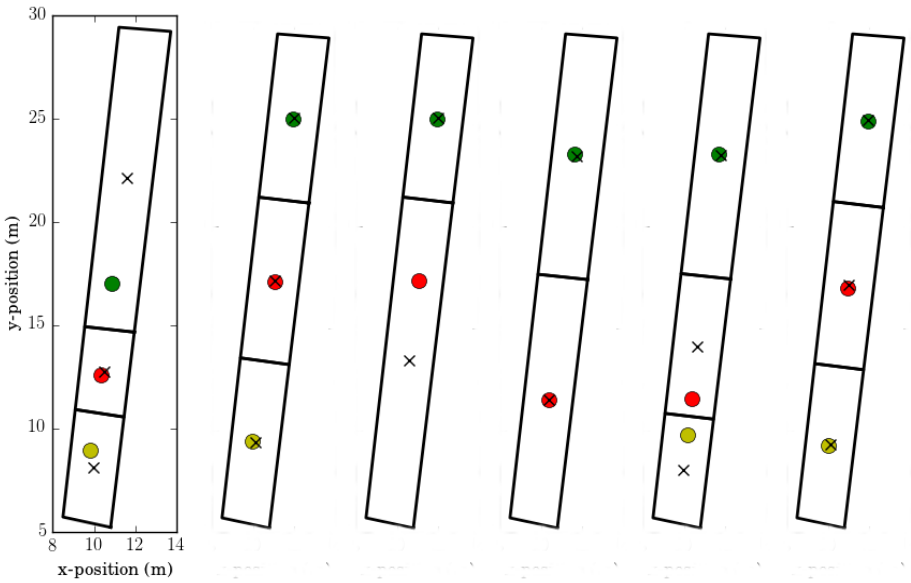
Figure 4.4: Experiment 2, cost plot of the system and per robot during the experiment.

4.2.1 Results

The three robots start in an inefficient configuration with a total cost just above 4 (Figs 4.4, 4.5a). As the behavior runs, they converge to a CVT with cost 2.1 (Fig. 4.5b). Then mbot06 is turned off, and the cost jumps to a higher value (coincidentally, just above 4 again) (Fig. 4.5c). The two remaining robots find a new CVT with a minimum cost of 3.25 (Fig. 4.5d). When mbot06 is turned back on, the cost drops instantly (Fig. 4.5e) and then decreases further as the robots find the final CVT at cost 2.1 (Fig. 4.5f, same configuration as in 4.5b). Some images from the video recording are shown in Fig. 4.6.

4.2.2 Discussion

The experiment worked as planned. The timeout mechanism marks a robot as inactive if it does not communicate for 10 seconds, and disregards that robot in the CVT computation. Even though there is no explicit synchronisation of group size between the remaining team members, they react close enough in time that no problems occur. If this was not the case, problems could



(a) t = 0 s (b) t = 50 (c) t = 60 (d) t = 100 (e) t = 180 (f) t = 205

Figure 4.5: Experiment 2, Voronoi diagrams at different points in time. Picture (a) shows the initial configuration, which leads to convergence with 3 robots (b). mbot06 is deactivated in (c) and a new convergence is reached in (d) with 2 robots. mbot06 is reactivated in (e) and the final convergence is shown in (f), similar to (b).

(a) (b)

Figure 4.6: Experiment 2, Images from the video capture. Picture (a) shows convergence with 2 robots, corresponding to Fig. 4.5d. In (b) we see convergence with 3 robots, corresponding to Fig. 4.5f.

happen when one robot performs CVT based on a group of 3, while the other considers a group of 2. Such situations have occured during development, due to implementation bugs, and they tend to be unstable.

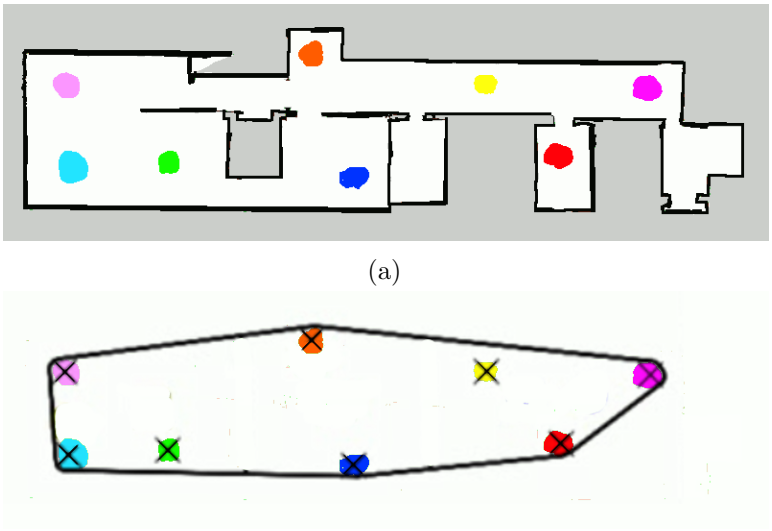
We can observe on the cost plot (Fig 4.4) that at convergence with 3 robots, their respective fractions of the cost are almost perfectly balanced. This is due to the shape of the room, which can be compared to experiment 1 where no such balance was attainable with 3 robots.

The cost values are useful to compare different configurations within a given room, but not necessarily to compare the results of experiment 1 vs experiment 2.

4.3 Coverage and patrolling in a nonconvex space

The third experiment tests the full behavior in a non-convex environment, with CVT, patrolling and dynamic entry and exit at run-time. A partial experiment with patrolling was performed with 1 real robot and recorded on video. Multi-robot patrolling was only performed in Webots simulation due to the difficulty of monitoring multiple robots during a real run. Accidents can happen and it would be preferable at this stage to have one person closely follow each robot and be able to turn it off manually if something goes wrong. For these reasons, only 1 robot was tested in reality for patrolling. Multi-robot simulations were done on the EFPL corridor map and on the IPOL map (Fig. 4.7a), to test the adaptability to a different environment from the one on which it was developed.

The experiment is set up with three robots in different starting positions. They perform CVT with virtual generators dividing the space between them. After convergence, the robots change from the CVT state to the PATROL state, each of them determines which of the waypoints are within its Voronoi cell, and performs a greedy patrolling cycle between these waypoints. After a certain time, one of the robots is manually deactivated. As in experiment 2, after a 10 second timeout the remaining two robots react, returning to the CVT state and perform a new CVT on their own. When they have converged, they again switch to the PATROL state. Each robot may have a different set of waypoints from before. The 2 robots patrol for a while, and then the third robot is manually started up again. It rejoins the group, they all perform CVT anew and then switch to patrolling. The experiment is recorded on screen.



(a)

(b)

Figure 4.7: Experiment 3. IPOL map input file (a) and resulting convex hull and waypoints (b).

4.3.1 Results

The experiment worked mostly as planned. Three robots performed CVT and patrolling, then one was deactivated and the remaining two made a new CVT configuration, which they then patrolled. The disabled robot was reactivated, rejoined the group and a new CVT was created, and finally three robots were patrolling again. The convex hull and waypoints on the IPOL map are shown in Fig. 4.7b. The resulting CVT configurations and patrol paths are shown in Fig. 4.8.

One problem happened a few times during the simulation: robots ran into walls and got stuck, unable to get out of them. This error has also happened in reality, and had to be corrected by manually moving the robot a few centimeters to help it find its way back. This is a known problem with the current state of navigation for the MBots, and it is unrelated to what has been developed during this project. It seems to happen more often in simulation than in real runs.

4.3.2 Discussion

The third experiment brings everything together and shows that we have a working implementation. It works on the IPOL map which it was not specifically developed for, indicating some generality. However, during the

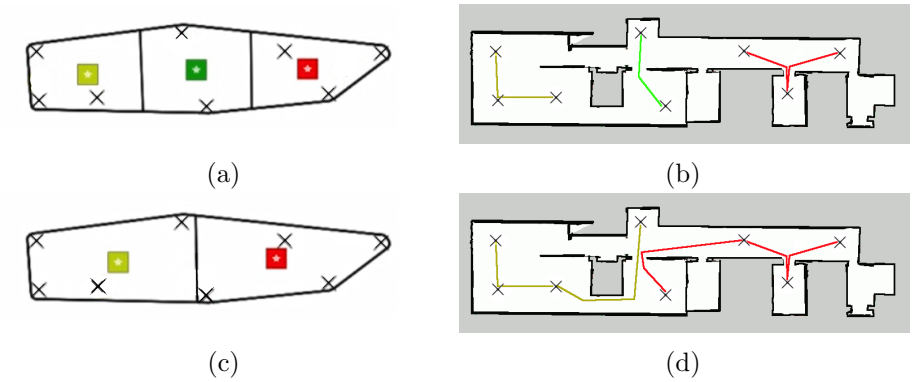


Figure 4.8: Experiment 3. Voronoi configuration and patrol paths with 3 robots (a,b) and with 2 robots (c,d).

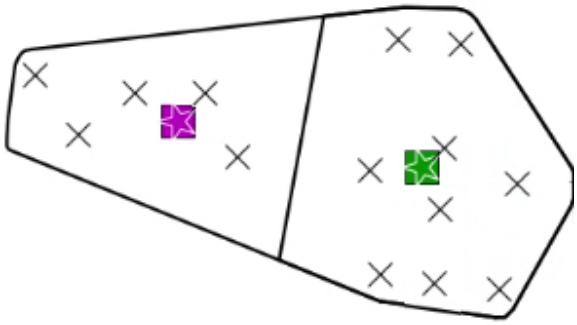
development it also became clear that there are several situations where the behavior will be suboptimal. These weak points suggest areas where further development is needed, or parts of the implementation could be changed.

Uneven distribution of waypoints

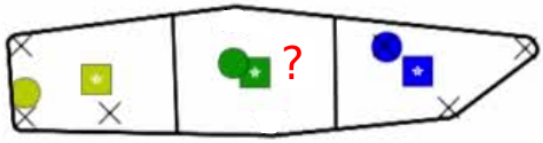
The CVT algorithm used produces a division of space as a function of the shape of the environment and the number of robots, but does not consider the placement of waypoints. This can easily lead to an uneven distribution of waypoints between robots, causing large differences in the frequency of visits to different regions and/or robots becoming idle because they are assigned no waypoints at all. Looking again at the EPFL map (Fig.4.9a), with 2 robots the waypoints are split 5 vs 9. With 3 robots on the IPOL map (Fig. 4.9b), if the two "green" waypoints weren't there, then the Voronoi diagram would be the same anyway, and the green robot would be assigned nothing to do. To improve the situation, one should also consider the paths between waypoints; a simple 7-7 split between two robots gives no guarantee of similar travel time for the patrol paths.

Inefficient map splitting

The CVT computation based on Euclidean distance measure is not well suited for use with a nonconvex environment. Depending on the shape of the map, it may work or it may give very bad results. The experimental results were acceptable since these maps are open and well connected, but on other maps we could get a different outcome. One example is the map from the Institute for Systems and Robotics (ISR) at IST, Portugal. This map



(a) Uneven distribution of waypoints between 2 robots on the EPFL map.



(b) Green robot is not assigned any waypoints.

Figure 4.9: Examples of weaknesses caused by the disconnection between waypoints and CVT algorithm.

has not been tested in simulation, but from the U-shape one can easily see potential problems (Fig 4.10). One robot could be assigned a Voronoi cell split to each end of the map, and have to walk long distances through areas where it has no waypoints. This problem can occur to various degree on many maps, and one suggested remedy is to use a CVT computation based on geodesic distance. An interesting approach is used by Bhattacharya et al [14], where the environment is discretized into square cells, then a geodesic Voronoi computation performed. This gives a division of the space with regards to real walking distance for the robots, which should solve several of the existing problems, although at a higher computational cost.

Improving patrol paths

Patrolling efficiency was not a main concern during the project, and a simple greedy algorithm was used which works but may be inefficient in many situations. This leaves room for improvement. Path planning can be done by running an Ant Colony optimization algorithm, or even brute forcing the optimal solution should be possible, since there are few waypoints on the given use cases. For either of these approaches, the environment should first be represented as a discrete graph, taking into account the walkable

(a) (b)

Figure 4.10: ISR map. The current approach with virtual generators and Euclidean distance measure can lead to inefficient divisions of space, such as (a) where the purple partition is disconnected. A better solution (b) could be achieved by using a geodesic distance measure instead.

paths between waypoints. It may be useful to introduce intermediate nodes between the waypoints on such a graph.

5 Ethical considerations

The interaction between robots and humans in a social setting is a young field in terms of both technology and research. This raises many ethical questions which need consideration as robots rise from the rank of "worker slaves" to take on the roles of nurses, teachers and friends. This chapter focuses on two ethical questions within social robotics and discusses them based on the MOnarCH project.

5.1 Antropomorphization and relationships

To antropomorphize is to ascribe human qualities such as feelings or intentional behavior to an inanimate object. This antropomorphization seems to originate in a human social need, and is also expressed by naming robots or caring for them in an affectionate way that may seem out of place considering that the robot is "just a machine". Many cases have been observed where humans do this with robots, in particular when working together or otherwise spending time with them over a longer period[19]. With a robot such as AIBO, which looks like a mechanical dog and imitates some dog behaviors, it may not be a surprise that people treat it as a pet. On the other hand, researchers were surprised to see how people care for their Roomba vacuum cleaning robots and how soldiers cared for a robot used for defusing land mines[19].

Antropomorphization can over time lead to humans developing relationships to their robots, and Scheutz warns for this type of one-sided relationships, which may lead to an unhealthy emotional dependence by the human on the robot [19]. The relationship is one-sided because the robot does not have any capabilities of reciprocating feelings or becoming "dependent" on the human in a similar way. The above examples are with robots which are not built for social interaction and have very little or no such functionality. Given that humans develop relationships to them, we should expect that this behavior will be stronger when humans interact with social robots made for that purpose. Scheutz further warns of a future scenario where these

unidirectional emotional bonds may be exploited for profit or other purposes on a large scale.

5.2 Social development of children

Sharkey & Sharkey warns that the social and mental development of children may be at risk if they are cared for by robots rather than humans to a large extent [19]. The problem is not the robots per se, but the lack of human contact, and the danger is larger the younger the children are, which has been documented in institutions for orphans [19]. We already have a development towards robot caregivers for children and elderly, with Japan and South Korea being at the forefront. If children were left entirely in robot care for long periods, it is likely that their development would suffer similarly to the orphans.

5.3 Looking at MOnarCH

The following discussion will be based on a video from the MOnarCH project, recorded on site at IPOL [20] and released in 2015. In this video, two staff members are interviewed about the robots and how they are used, and we also observe patient children and their parents. Rather than refer to the robot as "Mbot" or "the robot", it has been inofficially named *Gasparzhino*, Portuguese for "little Casper". We also see a mother talking to her child and referring to the robot as "he", while discussing why the robot behaved in a certain way, as if it had thoughts and intentions. A staff member from IST tells us that the robots are intended to be both a classroom assistant and a play-buddy for the children.

5.4 Discussion

The video shows several examples of antropomorphization: in naming the robot and in talking about it almost like a person with intentions. The distinction between human thoughts and the information processing of a machine may be clear for an adult, but not for a child. With daily interaction it becomes natural to refer to the robot almost as a person. It would perhaps be more strange if we created machines to become part of our social environment, but refused to use the pronouns "he/she" when talking about them. The robots are used to interact with the children, but cannot be said to care for them, and the interaction that takes place is always supervised by adults or staff, not exclusive between robots and children.

For the applications within MOnarCH at IPOL, these things are barely problematic. First, the social capabilities of the robot are not advanced enough that a human should develop feelings for it or a unidirectional relationship as discussed by Scheutz. Even if they did develop a relationship of some sort, there is no way for the robot to abuse such a relationship. Second, the hospital environment is well monitored and the children are being taken care of by professionals. They are not being left on their own with the robots to such an extent that we need to fear for their social development. On the contrary, as told by a staff member in the video, the robots can encourage and improve social interaction between the kids since the type of activities and play that the robot does are social and physical in their nature. This is different from most "gadgets" such as tablets or video games, often accused of making children physically passive and socially isolated.

It seems unlikely that the MOnarCH project will create ethical problems of the kinds discussed, but it is still important to be aware of and consider the potential issues. We may soon have far more advanced robot nannies which can independently take care of our children, and that is an entirely different situation.

6 Conclusions

The purpose of the project was to investigate how Voronoi coverage methods can be applied for distributed coverage and patrolling within MOnarCH. A multi-robot behavior was developed and centroidal Voronoi coverage proven useful for the implementation. One of the main challenges was how to handle a nonconvex environment, and the implementation was inspired by recent research in the field, mainly Cortes et al [6] and Breitenmoser et al [8].

The implementation works on different map layouts and allows for flexibility in placement of waypoints and patrol areas. At the same time, it has several drawbacks which can cause it to be inefficient on certain map types. Some of these drawbacks can be compensated for through careful placement of waypoints, but it would be preferrable if the behavior could autonomously adapt to a wider range of map layouts. The known drawbacks give ideas for future developments and improvements as discussed in the results section, by using geodesic distance measures instead of Euclidean, and by optimizing the patrol paths using graphs or otherwise discretizing the relevant parts of the map.

Some requirements were stated in the beginning of the project. The first two were fulfilled, which concern the distributed computation and allowing for a changing group size during run-time, without human intervention. The third requirement was for robustness to communication and network problems, and while part of the implementation took this into account, it was not a focus and there is much room for further work on this aspect. The fourth requirement, a nonobstructive behavior, was not addressed at all. It may finally be the responsibility of a higher level software planner/coordinator to make sure that robots do not block paths, or it may be an extension of the current behavior once the core navigation parts are settled on.

Bibliography

[1] T. Fong, I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots", *Robotics and Autonomous Systems*, no. 42, pp. 143–166, 2003.

[2] S. del Moral, D. Pardo, and C. Angulo, "Social robotic paradigms: an overview", *Bio-Inspired Systems: Computational and Ambient Intelligence*, vol. 5517, pp. 773–780, 2009. DOI: 10.1007/978-3-642-02478-8_97.

[3] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and applications, 3rd ed.* Springer, 2008, ISBN: 978-3-540-77974-2.

[4] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley and Sons, LTD, 2000, ISBN: 978-0471986355.

[5] S. Fortune, "A sweepline algorithm for voronoi diagrams", *Algorithmica*, no. 2, pp. 153–174, 1987.

[6] J. Cortés, S. Martínez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks", *IEEE Transactions on robotics & automation*, vol. 20, no. 2, pp. 243–255, 2001.

[7] Q. Du, V. Faber, and M. Gunzburger, "Centroidal voronoi tessellations: applications and algorithms", *SIAM Review*, vol. 41, no. 4, pp. 637–676, 1999. DOI: S0036144599352836.

[8] A. Breitenmoser, M. Schwager, J.-C. Metzger, R. Siegward, and D. Rus, "Voronoi coverage of non-convex environments with a group of networked robots", *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4982–4989, 2010.

[9] H. Endres, W. Feiten, and G. Lawitzky, "Field test of a navigation system: autonomous cleaning in supermarkets", *Proceedings of the 1998 IEEE International Conference on Robotics & Automation (ICRA)*, pp. 1779–1781, 1998.

[10] E. Acar, H. Choset, Y. Zhang, and M. Schervish, "Path planning for robotic demining: robust sensor-based coverage of unstructured

environments and probabilistic methods", *The International Journal of Robotics Research*, vol. 22, no. 7 - 8, pp. 441–466, Jul. 2003.

[11] N. Correll and A. Martinoli, "Multirobot inspection of industrial machinery", *Robotics & Automation Magazine*, vol. 16, no. 1, pp. 103–112, 2009.

[12] H. Choset, "Coverage for robotics - a survey of recent results", *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 31, pp. 113–126, 2001.

[13] C. H. Caicedo-Nunez and M. Zefran, "A coverage algorithm for a class of non-convex regions", *IEEE International Conference on Decision and Control (ICDC)*, 2008.

[14] S. Bhattacharya, N. Michael, and V. Kumar, "Distributed coverage and exploration in unknown non-convex environments", *Springer Tracts in advanced robotics*, vol. 83, pp. 61–75, 2013.

[15] D. B. S. Portugal, "Robocops: a study of coordination algorithms for autonomous mobile robots in patrolling missions", 2009.

[16] D. Portugal and R. Rocha, "A survey on multi-robot patrolling algorithms", *IFIP Advances in communication and technology*, vol. 349, pp. 139–146, 2011. DOI: 10.1007/978-3-642-19170-1_15.

[17] P. Alvito, C. Marques, P. Carriço, M. Barbosa, J. Estilita, D. Antunes, and D. Gonçalves, *Deliverable D2.1.1: MOnarCH Robots Hardware*, http://users.isr.ist.utl.pt/~jseq/MOnarCH/Deliverables/D2.2.1_update.pdf, [Online; accessed 26-July-2015], 2014.

[18] J. Messias, R. Ventura, and P. Lima, *The MOnarCH Situation Awareness Module*, [Internal documentation; accessed 26-July-2015], 2015.

[19] P. Lin, K. Abney, and G. A. Bekey, *Robot Ethics: The ethical and social implications of robotics*. The MIT Press, 2011, ISBN: 026252600X.

[20] MOnarCH. (2015). Little Casper (Gasparzhino) at IPOL, MOnarCH, [Online]. Available: http://users.isr.ist.utl.pt/~jseq/MOnarCH/GasparzinhoIPOL-subtitles-en.mp4.

[21] J. Sequeira, M. F. Pereira, V. Gonzalez, and I. Ferreira, "Deliverable D8.8.3 - Ethics, norms of behavior, and social rules for robots at IPOL and in the partners' testbeds. Version 6", 2013.