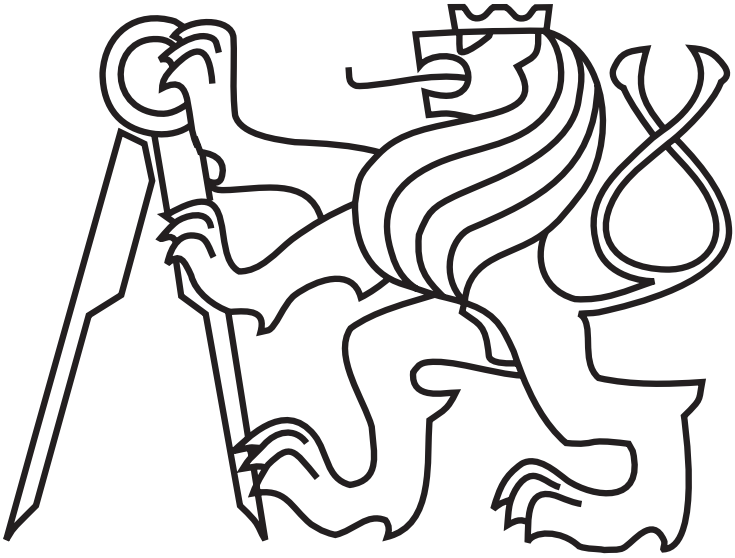
CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF ELECTRICAL ENGINEERING

DIPLOMA THESIS



Tomáš Juchelka

Exploration algorithms in a polygonal domain

Department of Cybernetics

Thesis supervisor: **RNDr. Miroslav Kulich, Ph.D.**

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Bc. Tomáš Juchelka

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Diploma Thesis: Exploration Algorithms in a Polygonal Domain

Guidelines:

1. Familiaraze yourself with exploration algorithms in mobile robotics.
2. Familiarize yourself with algorithms and open-source libraries for boolean opearions on polygons.
3. Implement selected exploration algorithms for polygonal domain in the ROS system.
4. Compare behavior of the algorithms in a simulator and on real robots in the SyRoTek system.
5. Document and discuss the obtained experimental results.

Bibliography/Sources:

[1] Basilico, N.; Amigoni, F.: Exploration Strategies Based on Multi-criteria Decision Making for Searching Environments in Rescue Operations. Autonomous Robots 31, 401-417, (2011).
[2] Gonzalez-Banos, H. H.; Latombe, J.C.: Navigation Strategies for Exploring Indoor Environments. The International Journal of Robotics Research 21, no. 10-11: 829-848 (2002).
[3] Makarenko, A. A.; Williams, S. B.; Bourgault, F.; Durrant-Whyte, H.F.: An experiment in integrated exploration. Robotics 1: 534-539, (2002).
[4] Robot Operating System, www.ros.org, accessed:12/2011.
[5] Systém pro robotickou tele-výuku (SyRoTek), syrotek.felk.cvut.cz, accessed:2011.
[6] Vatti, B. R.: A generic solution to polygon clipping. Communications of the ACM, Vol 35, Issue 7, pp. 56–63, (July 1992).

Diploma Thesis Supervisor: RNDr. Miroslav Kulich, Ph.D.

Valid until: the end of the summer semester of academic year 2012/2013

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 9, 2012

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Tomáš Juchelka

Studijní program: Kybernetika a robotika (magisterský)

Obor: Robotika

Název tématu: Algoritmy prohledávání v polygonální doméně

Pokyny pro vypracování:

1. Seznamte se s algoritmy prohledávání v mobilní robotice.
2. Seznamte se s algoritmy a open-source knihovnami pro booleovské operace s polygony.
3. Vybrané algoritmy prohledávání implementujte pro polygonální doménu v systému ROS.
4. Chování algoritmů porovnejte v simulátoru a na reálných robotech systému SyRotek.
5. Experimentální výsledky zdokumentujte a diskutujte.

Seznam odborné literatury:

[1] Basilico, N.; Amigoni, F.: Exploration Strategies Based on Multi-criteria Decision Making for Searching Environments in Rescue Operations. Autonomous Robots 31, 401-417, (2011).
[2] Gonzalez-Banos, H. H.; Latombe, J.C.: Navigation Strategies for Exploring Indoor Environments. The International Journal of Robotics Research 21, no. 10-11: 829-848 (2002).
[3] Makarenko, A. A.; Williams, S. B.; Bourgault, F.; Durrant-Whyte, H.F.: An experiment in integrated exploration. Robotics 1: 534-539, (2002).
[4] Robot Operating System, www.ros.org, accessed:12/2011.
[5] Systém pro robotickou tele-výuku (SyRoTek), syrotek.felk.cvut.cz, accessed:2011.
[6] Vatti, B. R.: A generic solution to polygon clipping. Communications of the ACM, Vol 35, Issue 7, pp. 56–63, (July 1992).

Vedoucí diplomové práce: RNDr. Miroslav Kulich, Ph.D.

Platnost zadání: do konce letního semestru 2012/2013

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 9. 1. 2012

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 1. 1. 2013 .....

.....
Podpis autora práce

Acknowledgements

I would like to thank my supervisor RNDr. Miroslav Kulich, Ph.D. for his guidance and patience. His advices always helped me to overcome difficulties in this thesis. I would also like to thank my family for their continuous support during my studies.

Abstrakt

Diplomová práce se zaměřuje na exploraci neznámého prostředí skupinou mobilních robotů. Každý z robotů objevuje své okolí získáváním senzorických dat a přispívá tím k vytvoření mapy prostředí. Explorace skupinou mobilních robotů se typicky používá při vojenských úkolech, čištění prostředí, hledání obětí a obecně v místech, která jsou lidem nebezpečná. Cílem je prozkoumat prostředí co nejrychleji, tedy minimalizovat celkový čas explorace. Mapa prostředí se obvykle reprezentuje pomocí mřížky obsazenosti, ale tato práce používá polygonální reprezentaci. Tento přístup má mnoho výhod, ale přináší určité problémy a vyžaduje speciální úpravy. Explorační proces závisí na explorační strategii. Hlavním cílem této práce je implementovat některé existující explorační strategie a framework s využitím polygonální reprezentace, kde mohou být tyto strategie testovány. Dosažené výsledky s použitím frameworku jsou prezentovány a diskutovány.

Abstract

This work is focused on the exploration of an unknown environment by a team of mobile robots. Each robot in the team discovers its neighbourhood by gathering sensory information from the environment and contributes to a global map. Typical applications are military assignments, environment cleaning, searching for victims, and generally locations dangerous for people. The goal is to explore the environment as soon as possible, thus minimize the exploration time. The common representation of the environment is an occupancy grid while a polygonal representation is used in this thesis. This approach has many advantages but it brings some problems, and requires special handling. The exploration process depends on the exploration strategy. The main goal of this thesis is to implement the general exploration framework in ROS based on polygonal representation of the environment together with some existing exploration strategies. Finally, the results achieved with the framework are presented and discussed.

Contents

List of Figures

List of Tables

List of scenarios

Chapter 1

Introduction

The problem of exploring an unknown environment is a fundamental problem in the mobile robotics. Typical applications are, e.g., search and rescue missions in a dangerous or hostile environment. In this task, the mobile robots are autonomously driven according to a sensor in order to discover potential victims and create a map of the environment. Each robot is equipped with sensors, so it is able to gather information about its neighbourhood. This thesis deals with a team of mobile robots in the exploration with several advantages. Firstly, a group of robots can finish a task faster than a robot. Secondly, the team of robots can cover the environment more uniformly and is also more fault-tolerant. On the other hand, more robots operating in the same area may cause interferences or collisions. Because the exploration time is crucial in search and rescue applications, the robots need to cooperate and make the exploration effective.

Several techniques dealing with the problem of coordinated multi-robot exploration were presented in [1, 2, 3, 4]. In majority of nowadays approaches, the maps are represented by an occupancy grid [5]. As the size of the environment can be very large, the grid representation requires a lot of memory. A polygonal representation is therefore more efficient map representation which can easily handle large environments in detail. This approach has many advantages but it brings also problems. Some of the algorithms designed for occupancy grids will not work on polygons. This thesis aims to use the polygonal representation in the existing exploration approaches. The goal is to improve their parameters by using of the polygonal representation. The second contribution of the thesis is in proposing a new framework where the strategies can be experimentally tested and analysed. The robot operating system ROS [6] was selected for this purpose. In order to make the exploration working on the polygons it is necessary to incorporate some external libraries, e.g., for a polygon clipping, Dijkstra's algorithm, visibility graph and triangulation into the developed framework.

The paper is organized as follows. The state of the art with description of the existing approaches is presented in Chapter 2. The proposed polygonal representation is described in Chapter 3. Chapter 4 describes the framework used for the experiments with the strategies. The achieved results are presented and discussed in Chapter 5. The final evaluation of the thesis is in Chapter 6.

Chapter 2

State of the art

The mobile robot exploration is the process in which robots autonomously operate in an unknown environment. The robots are navigated through the environment in order to create a map of it. The map is incrementally built and serves as a model of the environment for further exploration steps. The process consists of a goal selection and navigation towards the selected goals. This is repeated until there are unexplored areas in the map. This paper focuses on the multi-robot exploration. The whole exploration process is summarized in Scenario 1. The exploration process involves the robots $R = \{r_1, r_2, \ldots, r_n\}$ and the goals $G = \{g_1, g_2, \ldots, g_m\}$. The robots are equipped with a laser range finder sensor. The exploration is a complex set of actions that leads to a complete model of the environment. It starts by reading actual sensor information by individual robots. After some data processing, the existing map is updated with this information. New goal candidates are determined in the map. An exploration strategy assigns the goal candidates to the robots. Having assigned the goals to the robots, the shortest path from the robots to the goals are found. Dijkstra's algorithm [7] is a suitable solution. Finally, the robots are navigated along the paths.

There exist many exploration strategies, see a nice description and comparison in Amignoni [8]. Having *n* robots and *m* goals, the problem is to find the optimal robot-goal assignment according to a defined strategy. The strategies described in this section use cost functions, utility functions or both, and they represent different approaches to the problem. The cost function evaluates the goal candidates and it is defined as the shortest collision-free distance. The distance is defined between each robot and goal as $l(r, g)$, where $r_i \in R$ and $g_i \in G$. A utility can be defined for the goals as presented in Burgard et al. [2]. The utility is a defined reward for visiting a goal while it does not depend on the distance. In each particular step, the task is to minimize the cost or maximize the utility.

The exploration can be either centralized or decentralized. In the first one, the robots share the common map and there is only a central element assigning goals to the robots. In the second one, each robot selects a goal individually. The centralized approach coordinates robots to work more effectively but suffers in the robustness. The decentralized approach is not dependent on the single element but it requires more complicated communication between the robots.

while *exists unexplored areas* **do**
    read current sensor information;
    update map with obtained data;
    select a new goal candidates;
    assign the goals to the robots;
    plan paths for the robots;
    move the robots towards the goals;

Scenario 1: The exploration algorithm

All the algorithms described in this section work on an *occupancy grid*. The occupancy grid is a representation of an environment where a map is divided into small cells with the defined size. Each cell stores information about the corresponding piece of the environment in the form of a probabilistic estimate of its state. It is obvious, that more cells give a more precise model of an environment with higher demands on memory and computational power. The choice of an appropriate granularity is therefore a trade-off between the model precision and system resource requirements. Each cell holds a probability value p_c that the cell is occupied and this probability lies in range $[0, 1]$. The cells are updated during the exploration according to the Bayes law [5]:

$$p(occ|r) = \frac{p(r|occ) * p(occ)}{p(r)},$$

where occ is the state of the cell and r is a new measurement to be incorporated. In the mobile robot exploration, the probability is threshold. The whole range is divided into three intervals, where cells in the same interval have assigned a label indicating their state. Thus the map cell can have one of the three different states. By default, cells have assigned p_c between p_{free} and p_{occ} which means that they are in the unexplored state.

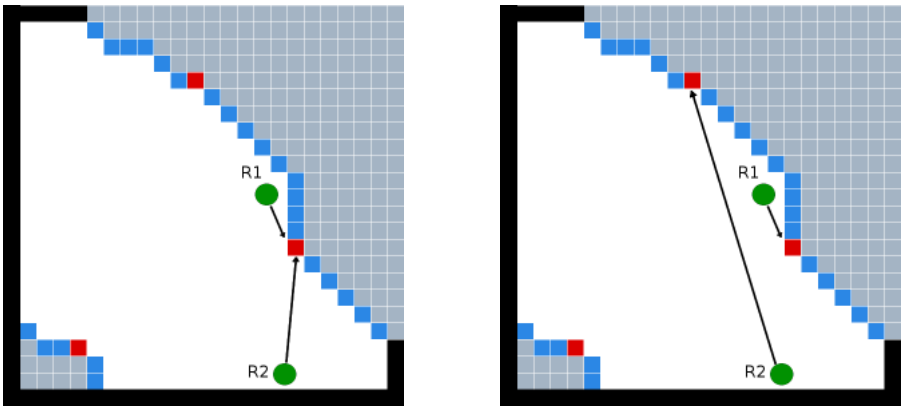
$$cell = \begin{cases} free, & p_c < p_{free} \\ occupied, & p_c > p_{occ} \\ unexplored, & p_{free} < p_c < p_{occ} \end{cases}$$

Yamauchi [9] presented a basic exploration strategy which introduces a *frontier*. The frontier is a location between the free and unexplored space. Figures 2.1 show an example of an occupancy grid with frontiers. There are two frontiers on the picture marked with the blue color. Both consist of adjacent cells and they can be reduced to a few goal candidates - the red cells. One may place the goal candidates either into the middle of the frontier or they can be distributed in a sensor range distance. The frontier is used in many approaches which are called frontier-based methods.

2.1 Yamauchi

In Yamauchi [9], each robot greedily heads towards the nearest goal according to a cost function without any coordination between robots. The strategy is simple and can be easily

implemented. On the other hand, one goal can be selected and explored by many robots as depicted in Figure 2.1(a). To remove this inefficiency it is possible to hide already selected goals for the further selection. This is used in the Broadcast of Local Eligibility (BLE) assignment algorithm developed by Werger & Mataric [10], see Scenario 2.



(a) (b)

Figure 2.1: Greedy assignment: (a) depicts two robots exploring the same goal; (b) shows an inefficient assignment of goals;

while *any robot remains unassigned* **do**
  find the robot-goal pair (i, j) with the highest utility;
  assign the goal j to the robot i and remove them from the consideration;

Scenario 2: BLE assignment algorithm.

Nevertheless, it is still a greedy algorithm which not necessarily produces the optimal solution. The solution depends on the order of the robot-goal assignments. Figure 2.1(b) depicts an example of an inefficient targets assignment.

2.2 Hungarian method

The more sophisticated method is the Hungarian method firstly introduced in Kuhn [11]. It is an optimization algorithm which solves the worker-task assignment. The assignment can be written in a form of the *n* x *n* matrix. In general, the element in the i-th row and j-th column represents the cost of assignment of the j-th task to the i-th worker. Let $c_{i,j}$ be the cost of assigning the *j*-th goal to the *i*-th robot. A naive approach is to test all combinations of the assignments and find the assignment with the minimum total cost. There are $n!$ possible assignments to test. The Hungarian method finds the optimal assignment for the given cost matrix C.

$$C = \begin{pmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,n} \\ c_{2,1} & c_{2,2} & \dots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \dots & c_{n,n} \end{pmatrix}$$

The algorithm which finds the optimal solution can be shortly described in four steps.

1. In each row, subtract the minimal element from each element in the row.
2. In each column, subtract the minimal element from its column.
3. Cover all zeros of the matrix with the minimal number of horizontal or vertical lines. If the number of lines equals n it is finished. The optimal assignment is given by the zeros covered by the lines. Otherwise proceed to the next step.
4. Find the smallest element not covered by any line. Subtract the element from each uncovered row and add the element to each covered column. Go to step 3

The problem can be also expressed as a linear program. The objective is to find integers α_{ij} that minimize

$$Z = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_{ij} c_{ij} \tag{2.1}$$

subject to

$$\begin{aligned} \sum_{i=1}^{n} \alpha_{ij} &= 1, \quad 1 \leq j \leq n, \\ \sum_{j=1}^{n} \alpha_{ij} &= 1, \quad 1 \leq i \leq n. \end{aligned} \tag{2.2}$$

The equation 2.1 expresses the total cost of the assignment, the same as the cost matrix C, and the equations 2.2 are constraints that ensure that the one goal is assigned to exactly one robot. The algorithm requires the number of robots to be the same as the number of goals which can not be guaranteed throughout the exploration. If the number of robots or goals is lower it is possible to add imaginary robots or goals to satisfy the assumption. They have assigned a fixed cost, so they don't affect the real ones. In the selection, the imaginary robots and targets are skipped. This strategy doesn't assign the same goal to different robots and it doesn't depend on the order of selection. The Hungarian method solves the robot-goal assignment in $O(n^3)$ polynomial time.

2.3 Burgard et al.

Burgard et al. [1, 2] use a decision theory to coordinate the exploration. The method estimates an expected information gain (EIG) of a frontier and combines it with a path cost. The EIG is the number of unexplored cells that are within the sensor range radius of the frontier. The EIG of the frontier is reduced considering the number of robots having it in their sensor range. Figure 2.2 shows the key features of the method. The circles indicate a sensor range and contain information gain regions. Dashed lines are rectangular approximations of the information gain regions. The rectangles are used to compute a potential overlap in the EIG. This approximation is fairly accurate while being much more efficient. A percentage of the overlap d_j for a frontier cell is computed between its information gain region and the information gain region of the other assigned frontiers as:

$$d_j = \frac{|IGR_j| \cap \sum_{i=1}^{n} |IGR_i|}{|IGR_j|}$$

Here $|IGR_j|$ is a size of the information gain region of a frontier cell, n is the number of robots and $|IGR_i|$ are sizes of the other assigned information gain regions. The EIG is decreased by the percentage d_j. The overall utility u_j is then computed as:

$$u_j = (1 - d_j) \times i_j - c_j,$$

where i_j is the EIG and c_j is the path cost. The method minimizes the overlap in the information gain among robots. The algorithm is presented in Scenario 3.

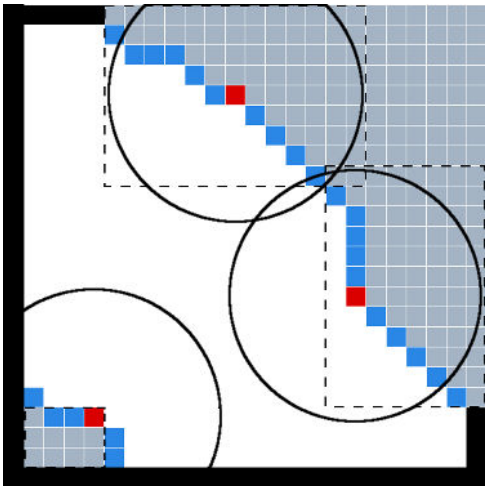


Figure 2.2: Expected information gain.

This strategy leads to a robot dispersion in small areas. Most of the computations are distributed amongst the robots. If robots can't see each other it is in fact a greedy strategy.

repeat
  find the target with the highest utility (EIG - cost);
  assign the target with a greedy algorithm;
  discount the utility of the remaining robots;
until *all targets have been assigned or no target has the EIG above a threshold* ;

Scenario 3: Targets assignment algorithm

2.4 Stachniss et al.

The method presented in Stachniss et al. [12] takes the structure of the environment into account. In general, indoor environments are structured and divided into rooms and corridors. This method tries to assign robots to the separated rooms. It partitions the explored space into segments and instead of the frontiers, robots are sent to the individual segments. Map is segmented using the *Voronoi Graphs*. The Voronoi Graph is a set of nodes for which at least two obstacle points with an equal distance exist and there is no other obstacle point closer to the nodes.

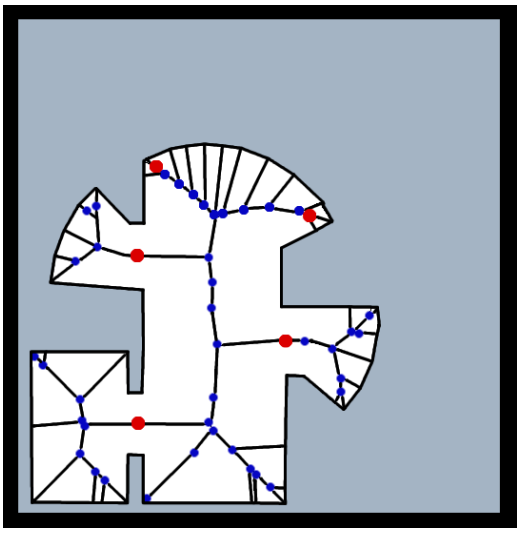


Figure 2.3: Environment with Voronoi Graph.

The Voronoi Graph consists of edges and nodes of a certain degree. The edges of the graph represent paths furthest from obstacles and without any collisions. So-called *critical points* are identified in the graph and they are marked with red colour. The critical points are nodes in the Voronoi Graph that satisfy several conditions.

- Their distance to the closest obstacle is a local minimum.
- They have to be nodes of degree 2.
- They must have a neighbour of degree 3.

Figure 2.3 shows the Voronoi Graph and the critical points found by the described method. The critical points are used as candidates for the goals. In most of cases, the critical point lies in a doorway. The problem here is that there is a lot of false positive candidates for the critical points. More complex environments require a more sophisticated segmentation algorithm based on a training data set.

2.5 Solanas et al.

In the majority of multi-robot tasks, robots start from a single area, e.g., building entrance. It leads to an exhaustive exploration of the starting area during the first phase of exploration. In search and rescue it is preferable that the robots quickly make a map in outline and then focus on the individual parts of the environment. Solanas et al. [13] presented an exploration strategy based on K-means clustering. The proposed technique divides the unknown space into K regions, where K is the number of robots.

At the beginning, K-means is applied and the map is partitioned into regions. The particular regions are assigned to the closest robots. After the assignment, each robot chooses a frontier according to a cost function. The cost function distinguishes whether the frontier belongs to the assigned region or not. The cost of F_j for the robot R_i assigned to the region ζ_i is defined as:

$$c_{i,j} = \begin{cases} \Delta + e(F_j, C_i) + o_{i,j} & F_j \notin \zeta_i \\ d(F_j, R_i) + o_{i,j} & F_j \in \zeta_i \end{cases}$$

where Δ is a constant penalization representing the diagonal length of the map, e is the euclidean distance, C_i is the centroid of the region, d is the real path cost defined by any path planning algorithm and $o_{i,j}$ is the accumulated penalization increasing the cost when the frontier has been already selected.

The frontier that does not belong to the assigned region receives a high penalization Δ. Thus robots prefer the frontiers in their assigned regions. If there is no frontier in the assigned region, robots select the closest frontier to their region. As the result, robots tend to work separately in their assigned regions. If a region is not directly accessible, other regions are explored on the way to the assigned one. Robots explore all these separated regions simultaneously because each robot heads to its own region. This leads to a dispersion between robots and different parts of the environment are explored at similar speeds.

In general, the K-means algorithm consists of the following steps.

1. Randomly choose K centroids C_i where $1 \leq i \leq K$.
2. Classify each cell to the class ζ_i of its closest centroid C_i.
3. Determine a new centroid for each class.
4. If all the centroids didn't change, it is finished. Otherwise continue with step 2.

2.6 Puig et al.

The exploration algorithm described in Puig et al. [14] also applies the K-means clustering to spread robots in the environment. The algorithm keeps the robots separated and working in different areas. It prevents some areas to be explored significantly later than others. It is similar approach to the previous method [13] in Section 2.5: unknown areas are partitioned into regions by the K-means algorithm, where the number of regions K is the same as the number of robots.

Two types of regions are distinguished. If the region assigned to the robot is directly accessible through a free space, the region is *accessible*. If the region isn't directly accessible, the region is *inaccessible* and a robot selects the goal from the accessible region which is the nearest to the assigned region according to a defined metric. The distances are defined for both region types separately. For the accessible region ζ_A, the distance from the robot r to the closest cell c of the region is defined as the minimal real path distance from r to all frontier cells adjacent to c. F_c is a set of the frontier cells adjacent to c.

$$d(r, c) = \min \{\delta(r, f) \mid f \in F_c\}, \quad c \in \zeta_A$$

For the inaccessible region ζ_I, the distance is defined as the geometric distance from r to c. If there is an obstacle between r and c a penalty ρ with the diagonal length of the map is added.

$$d(r, c) = g(r, c) + \rho, \quad c \in \zeta_I$$

Finally the distance between a robot r and a region ζ is the minimum of the distances to all regions.

$$d(r, \zeta) = \min d(r, c), \quad c \in \zeta_A \; or \; c \in \zeta_I$$

After the robot-region assignment it is necessary to assign goals to robots. The robots with assigned inaccessible region have priority to select goals first. The distance between a robot and a frontier is defined as:

$$d(r_i, f_j) = \begin{cases} \delta(r_i, f_j) + \rho(f_j, cc_{r_i}), & f_j \in \zeta_A \\ \infty, \quad otherwise \end{cases}$$

where cc_{r_i} is the closest cell of the region assigned to the robot r_i. The sum $\rho(f_j, cc_{r_i}) = g(f_j, cc_{r_i}) + \rho_1(f_j, cc_{r_i}) + \rho_2(f_j)$ consists of the geometric distance between f_j and cc_{r_i} and penalizations ρ_1 if is there is an obstacle between f_j and cc_{r_i} and ρ_2 is also a penalization with a positive value if the frontier has been already selected. If f_j is not accessible for r_i the distance is *infinite*.

Robots tend to explore their assigned regions as in [13] and the main difference is in penalization if there is an obstacle.

Chapter 3

Polygonal domain

The exploration process with the strategies described in chapter 2 works primarily on the occupancy grid. This chapter summarizes all modifications and tools necessary to use the polygonal representation. The goal is to make it working in the same way but with advantages of the effective polygonal approach of the environment. It requires much less memory in comparison with the occupancy grid while it keeps a high detail. Notice also that various operations can be done more easily on the polygon rather than on the occupancy grid, e.g., rotation, scaling, etc.

In geometry a *polygon* is a geometric shape composed of finite number of line segments. The line segments are called *edges* and the point where two of the line segments meet is called a *vertex*. The polygon edges form a closed contour surrounding the polygon interior. A chain of edges or a part of the polygon is called a *polyline*.

For the exploration itself it is necessary to extend the polygon with a frontier information, therefore an edge can be either a frontier or obstacle. The absence of the third unexplored state is a difference in comparison with the occupancy grid.

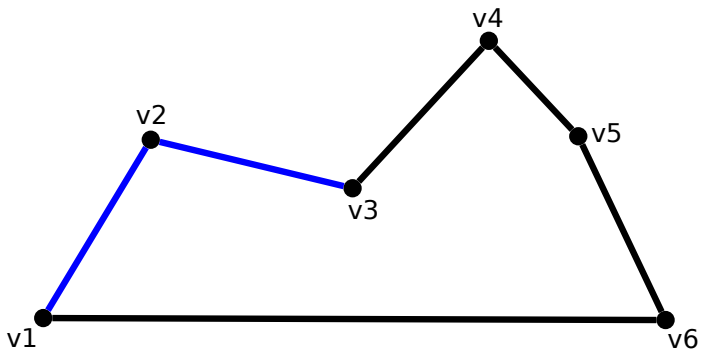


Figure 3.1: Polygon with frontiers.

An example of polygon is shown in Figure 3.1. The frontier edges are marked with the blue colour and the other edges belong to the obstacles. Algorithms used in the exploration need

to work with these types of edges. Furthermore, it is common that a polygon contains more frontier edges forming a polyline.

A library for operations on polygons necessary for the map building is presented in Section 3.1 followed by a description of the algorithm used in the library and its modifications.

3.1 Polygon clipping

The clipping is a general term for boolean operations, e.g., intersection, union, difference, etc. For the gradual building of the map it is needed to perform the union operation. There exists a wide range of libraries. This work uses the Clipper library [15] which is open-source library based on the Vatti clipping algorithm [16]. The algorithm is described in Section 3.1.1. It is fast and versatile, see the comparison of polygon clipping libraries [17]. Polygons with 174244 vertices were used in this benchmark.

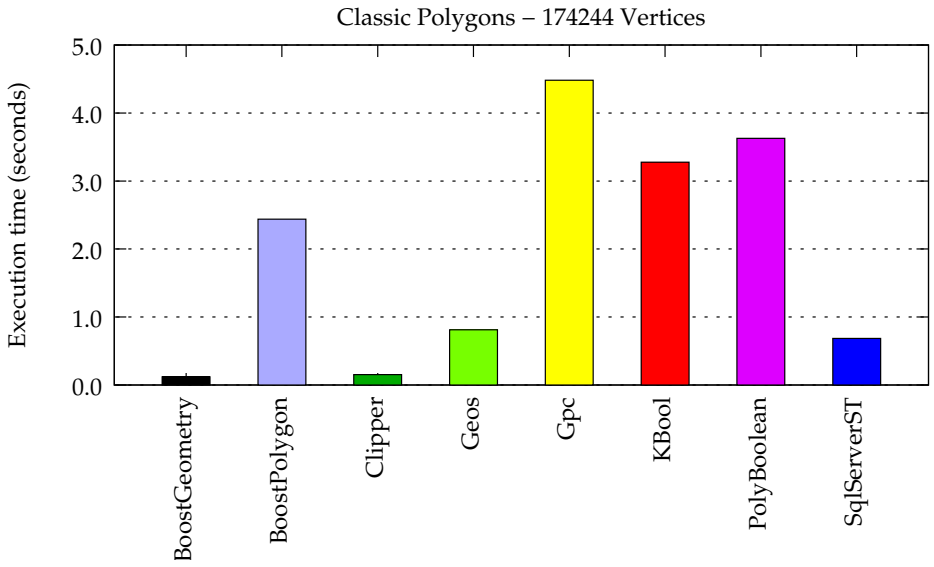


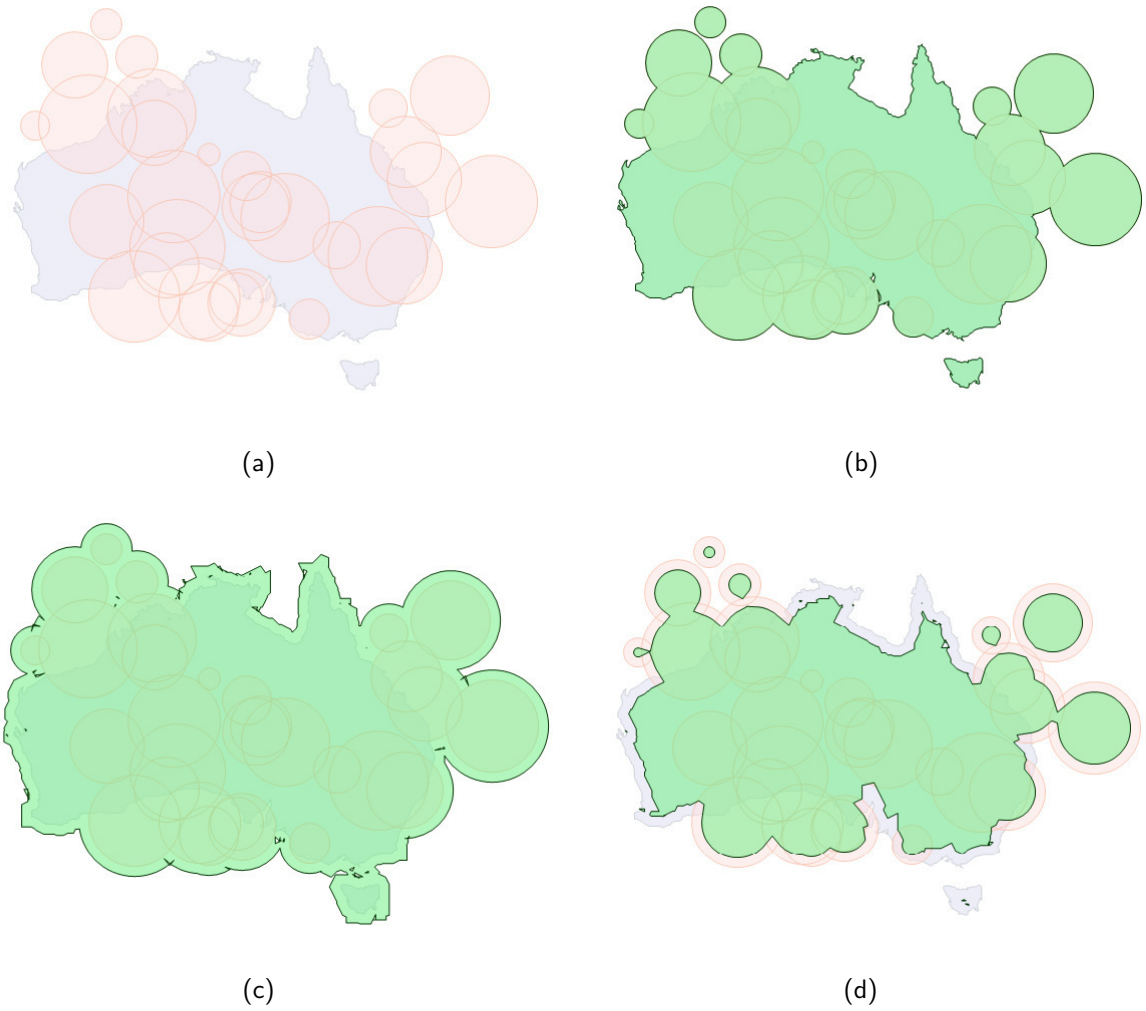
Figure 3.2: The comparison of several polygon clipping libraries.

The graph 3.2 was generated on the basis of the results presented in [17]. The Boost.Geometry has been the fastest followed by the Clipper. All the libraries primarily perform the boolean clipping operations. The selected library needs to handle polygons with holes, self-intersecting polygons and polygons with overlapping co-linear edges. All these attributes are crucial in this project except the self-intersection because the laser range finder produces a star-shaped polygons. The Clipper has been finally selected because it meets all requirements and offers an additional useful feature, i.e., a polygon offsetting. At the time of the polygon library selection, the Boost.Geometry was under development and not a part of the Boost library.

The only drawback which occurs across all the clipping libraries is that it is not possible to set the property of the edge of polygon. This project also uses the frontier based approach

that was mentioned in chapter 2. So there must be a way how to identify frontiers in the map. Practically it is needed to propagate the information throughout the clipping process. This can be done by the modification of the library described in Section 3.2.

Figure 3.3(b) shows an example of union operation which is used in this thesis for the incremental map building.



(a) (b)

(c) (d)

Figure 3.3: Operations on polygons: (a) shows default polygons; (b) describes union operation; (c) describes application of positive offset; (d) describes application of negative offset

3.1.1 Vatti algorithm

The Vatti algorithm can handle a large set of polygons like ones with holes, self-intersecting etc. Here, the simplified overview of the algorithm is described. The complete description is

beyond the scope of this thesis and can be found in [16]. The algorithm clips a subject polygon against a clip polygon. The algorithm distinguishes a left or right edge with respect to the interior of the polygon. Horizontal edges can be considered as left or right as it comes. Thus a polygon can be represented as a set of left and right bounds. The bound starts at a local minimum and ends at a local maximum. The vertices between these extremes are called an intermediate vertices. The algorithm builds the bounds in a bottom-up fashion using a scan beam. A scan beam is a horizontal area between two scan lines which contain at least one vertex from the polygons. There are no vertices between the scan lines. A polygon described with this notation is in Figure 3.4

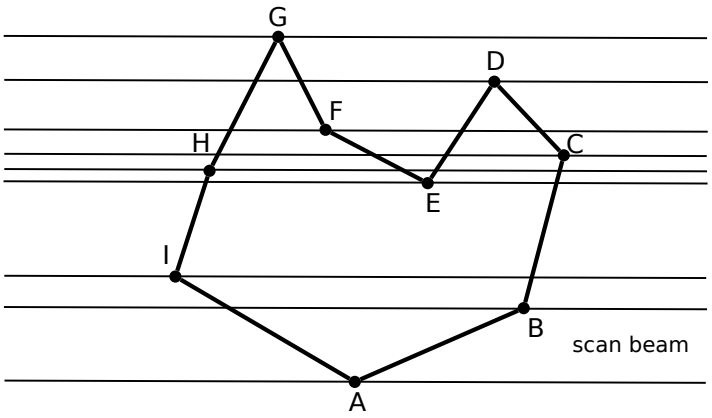


Figure 3.4: Vatti's representation of a polygon.

The vertices *A,B,C,D* form the right bound and the vertices *A,I,H,G* form the left bound. The vertex *A* is the local minimum and the vertex *D* is the local maximum. The scan beam passes through each vertex.

The polygons are scanned from the bottom to the top starting at the lowest scan beam. The first step is to compute the left and right bounds and build a *local minima list* (LML). Then an *active edge list* (AEL) is build. The AEL contains all the edges intersected by the current scan beam. Edge intersections are found and new vertices are created. These intersections are classified according to classification rules. Two edges are like if they belong to the same polygon and unlike otherwise. The classification rules are defined in Table 3.1 for like and unlike edges respectively. Letters are shortcuts for left (L), right (R), subject (S), clip (C), local minimum (MN), local maximum (MX), left intermediate (LI) and right intermediate (RI).

Unlike edges	**Like edges**
$(LC \times LS)$ or $(LS \times LC) = LI$	$(LC \times RC)$ or $(RC \times LC) = LI$ and RI
$(RC \times RS)$ or $(RS \times RC) = RI$	$(LS \times RS)$ or $(RS \times LS) = RI$ and LI
$(LS \times RC)$ or $(LC \times RS) = MX$	
$(RS \times LC)$ or $(RC \times LS) = MN$	

Table 3.1: Classification rules for the intersection points.

The rules produce the union of polygons. After the classification, various actions are applied. In a local minimum, a new polygon node is created and added into the output polygon. A left or right intermediate vertex is added into the left or right end of the vertex list of the output polygon. In a local maximum the polygon may be closed or is appended to the other polygon. There is a lot of exceptions and special handling so this description is very simplified. Especially horizontal edges cause problems so they are handled separately and collinear horizontal edges are joined into a single edge. The algorithm is summarized in Scenario 4.

while *an unprocessed scan beam exists* **do**
    compute the left and right bounds and build the LML;
    find intersections of edges in the current scan beam and build the AEL;
    classify the vertices according to the classification rules;
    process the vertices according to their class;
    process the horizontal edges;

Scenario 4: Vatti clipping algorithm

3.1.2 Polygon offsetting

For planning purposes it is common that a map is inflated by a robot radius. The inflated map forms a free configuration space, the space of the robot positions where a collision with obstacle can't happen. Finding the shortest path is then limited by this space. On an occupancy grid it is quite straightforward process, e.g., an application of dilation mask. In the polygonal domain the inflating resides in applying an offsetting operation. The library performs a polygon offsetting depicted in Figures 3.3(c),(d). When a negative offset is applied, outer polygons are contracted and holes are expanded by the offset amount. With a positive offset it is reversed. An example of the negative offset application is in Figure 3.5.

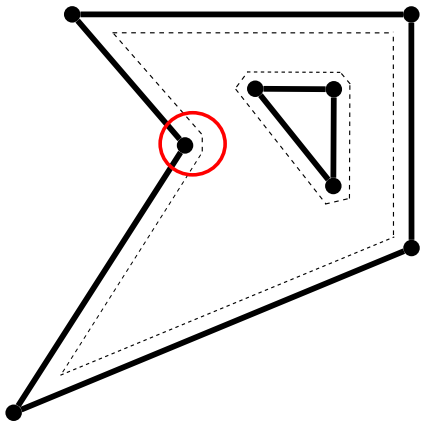


Figure 3.5: Offset polygon.

Vertices with an obtuse angle must be approximated, i.e., rounded or squared. The red circle shows the squaring. The first step of the polygon offsetting algorithm is to compute a

unit normal to an edge at each vertex. A new points are created on the normals in the offset distance. In general, there are two cases depicted in Figure 3.6.

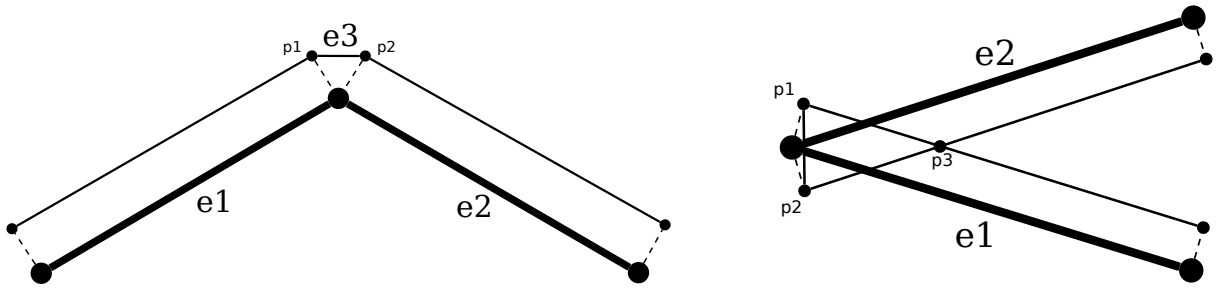


Figure 3.6: Angle processing: (a) for the obtuse angle a new edge and new points are created; (b) for the sharp angle a new point is created as an intersection of the edges;

Figure 3.6(b) shows the occurrence of a redundant area. The new points p_1,p_2 and the intersection p_3 form a triangle which must be excluded from the output. The output of the polygon offsetting contains the redundant areas as self-intersections which are removed by the union operation. The summary of the offsetting is described in Scenario 5.

forall *polygons in the map* **do**
- compute the unit normals to edges;
- create new points on the normals;
- add the new points to the output polygon;
- clip potential redundant areas;

Scenario 5: Polygon offsetting algorithm

3.2 Modifications

Let polygons passed to the clipping algorithm be original polygons and let polygons created by the algorithm be output polygons. The problem with the polygonal representation is that the clipping libraries are not capable of preserving user-defined point attributes through the clipping process. The Clipper library and the others do the clipping regardless of the attributes. Also the polygon offsetting must be done with respect to the attributes. The contracted and expanded polygons must contain the same information about the points as the original polygon. Unfortunately it is not currently supported by the library.

3.2.1 Modifications of clipping

Necessary modifications of the clipping algorithm can be divided in two approaches. The first approach lies in modifications of the specific parts of the algorithm. The modifications start with extending the internal data structures with the user-defined attribute. A vertex is represented by x,y values and information about the type of input and output edge, i.e., whether the edge is a frontier or obstacle. The edge is a structure containing two vertices and the whole edge can be marked as a frontier according to the types of the vertices. The idea was to use the marked edges and work with them according to their type. The modifications done were tested with unsatisfactory results. The problems are in joining horizontal edges, overlapping edges, local minima, etc. One of the reasons is that there is no relation between the output vertices and the input edges. For example, if two different bounds share their local minima, a new vertex is added into the output polygon. There is no guarantee which bound the algorithm takes first. In processing the next bound it skips the vertex because it was already added. The example situation is illustrated in Figure 3.7 where v_{min} is the local minimum which is added to the output either in processing edges e_1 and e_2 or e_3 and e_4. Because the first pair of edges is a frontier and the second pair is not, the parameters of the added vertex may be different. It would require a significant implementation effort and knowledge about the concrete implementation of the algorithm.

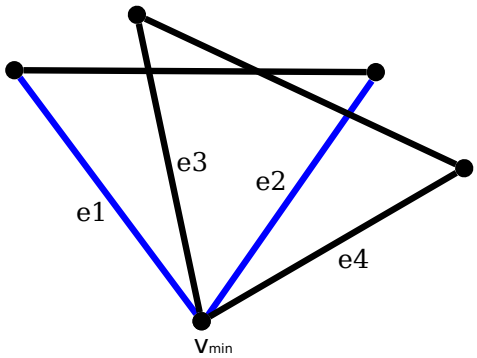


Figure 3.7: Problem in local minima.

The second approach, which was used in the thesis, is to post-process the edges of the output polygons, compare them with the original ones and assign them the correct property. Each edge from the result polygon is compared against the edges from the original polygon. A comparison is made by computing perpendicular distances from both vertices of the output edge to all edges in the original polygon and adding a penalty when the output edge is shifted or longer than the original edge. Thus each pair of edges is evaluated and its quality is expressed by a penalty value.

The penalty value can be expressed as a sum of all the distances depicted in Figure 3.8.

$$P = p_1 + p_2 + |d_1| + |d_2| \tag{3.1}$$

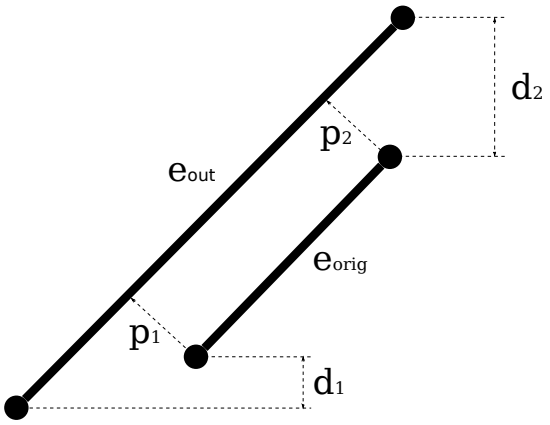


Figure 3.8: Comparison of edges.

Distances p_1,p_2 are the perpendicular distances from vertices to a line and d_1,d_2 are differences in y-axis. Notice that the distance d_1 is considered only if the bottom vertex of e_{out} has lower y-coordinate than the bottom vertex of e_{orig} and the distance d_2 is considered only if the top vertex of e_{out} has higher y-coordinate than the top vertex of e_{orig} The differences are added as the absolute values to eliminate the sign effect. The best matching pair, i.e., the pair with the lowest penalty, is considered as correct and the information from the found original edge is passed to the output edge.

This process is done after polygon clipping and it is unfortunately much more computationally complex. From the knowledge of the clipping algorithm it is possible to do some simplifications that speed up the matching. The algorithm creates the bounds with the edges in bottom-up fashion starting at the local minima. The most important fact is that the edges are ordered by y-coordinate. The bounds are also ordered by y-coordinate of their local minima. These internal structures can be used instead of the original polygons. Three general rules can be defined.

- The bounds with y-coordinate of its local minima higher than y-coordinate of the top vertex of the output edge can be completely skipped.
- Skip the rest of one bound if y-coordinate of the bottom vertex of the edge from the bound is higher that the top vertex of the output edge.
- Skip further comparison of an output edge if the penalty value is zero.

These criteria improve the speed of the algorithm significantly. Figure 3.9 shows how the simplifications affect the performance of the algorithm. Although the modifications slow-down the clipping, with the simplifications, the clipping can be applied on real problems with no worries. For example the biggest map used in the experiments, i.e., Hospital section map, contains approximately 1300 vertices and the clipping with the accelerated modifications is 2 times slower than the original clipping algorithm.

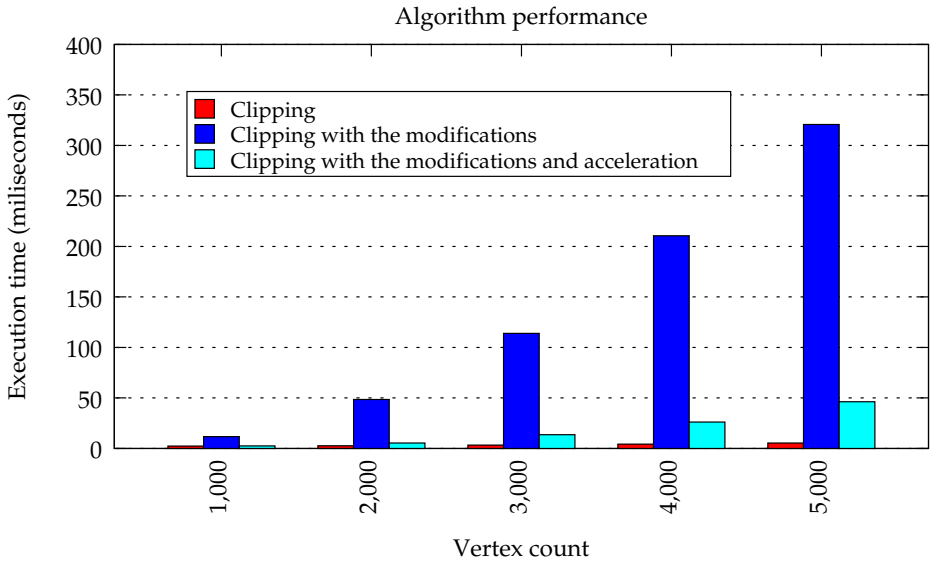


Figure 3.9: Performance of the modified algorithm.

3.2.2 Modifications of offsetting

Passing of the user-defined attribute, i.e., the frontier information, is much easier than in the clipping process. The modifications again reside in extending the internal data structures with the user-defined attribute. Here, the unit normal is extended by the frontier information. If an edge is a frontier, the unit normals at both vertices are considered as frontiers. The new point created on the unit normal inherits the frontier information from the edge. As shown in Figure 3.10, the point p_1 must inherit the frontier information from the edge e_1, etc. A new edge e_3 between the points p_1 and p_2 is created. The edge e_3 can have assigned the same information as one of its neighbours. It does not matter which neighbour is chosen but in this thesis, the obstacle edge is preferred.

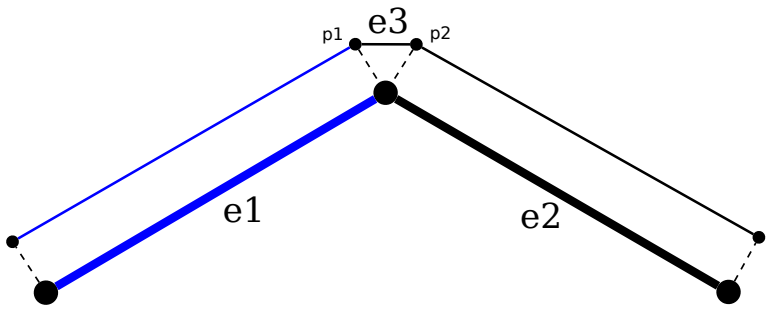


Figure 3.10: Unit normals processing.

Notice that because of the clipping of redundant areas it is necessary to do the edge matching too. There is exactly the same algorithm as the one used for the polygon clipping.

Chapter 4

Framework

The framework providing a complex support for creating and testing exploration strategies has been created as a part of this thesis. The framework contains all the components necessary for the mobile robot exploration in a polygonal domain. The framework includes robot control, map handling, path planning, exploration strategies, etc. The framework was designed to be modular and therefore the particular parts can be easily substituted as needed. A great emphasis was put on the modularity of the exploration strategies. The framework presented here allows to implement a new exploration strategy, test it, and see its parameters. The evaluation of the exploration strategies creates a log file containing results. The framework is written in *C++* language and it uses ROS [6] as a communication middleware. Section 4.1 briefly presents some informations about ROS and describes the framework.

4.1 ROS

ROS (Robot Operating System) is an open-source operating system for developing robot applications. ROS is described in a greater detail in [18]. ROS is a distributed framework of processes - *nodes* that enables executables to be individually designed and run. It provides interprocess communication, package system, tools, and libraries, etc. The system supports different languages: C++, Python, Octave, and LISP. The ROS node is one of the fundamental elements in ROS. It is a process performing an individual task. Each node has a defined rate that specifies a frequency at which the node loops at. Since ROS is modular and distributed there are typically many nodes in a project.

The nodes communicate with each other by sending messages. The messages may be delivered in a form of a topic or a service differing in the type of use. Firstly, the node may send the message by publishing it on the topic. The topic works as a broadcast. When a node is publishing a message on the topic, several other nodes can receive the message. Secondly, when a node needs a message at a given time it is possible to use the service. The service is called upon a request and serves for a synchronous communication but only between two nodes.

4.2 Framework structure

The graph 4.1 depicts the structure of the framework with one robot. The nodes in the graph represent individual ROS nodes and the arrows indicate ROS communication infrastructure such as topics or services. Several nodes are used in this project. The *stage* and *rviz* nodes highlighted in the graph are parts of ROS framework. The other nodes were implemented within the thesis. The following text describes the individual nodes.

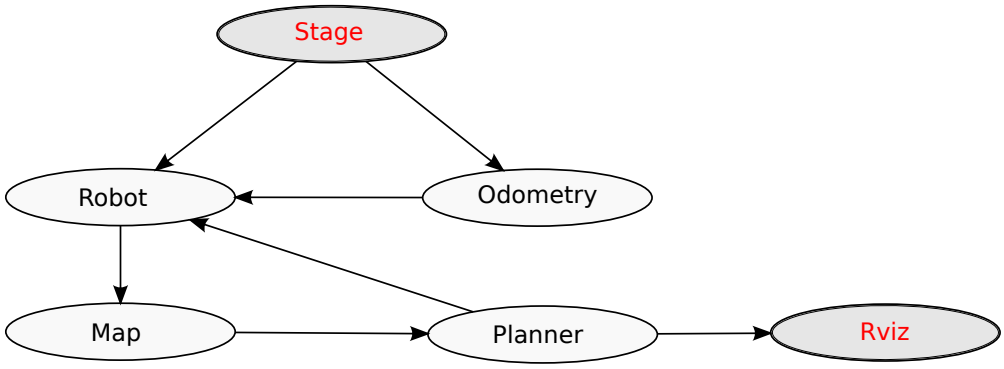


Figure 4.1: Framework topology with highlighted native ROS nodes.

Stage node
The stage node provides 2D simulation using the Stage multi-robot simulator [19]. The Stage simulates a world which is defined in a *world* file. This file defines the environment and its content like a map, robot and position models, laser models, and other objects. The stage node exposes a functionality of the Stage as the ROS topics. If there is more than one robot in the simulation, their topics are prefixed with their names, e.g., /robot_0/base_scan, etc.

- topics
  - /robot_i/base_scan - publishes scans from the laser range finder
  - /robot_i/base_pose_ground_truth - publishes positions of the robots
  - /robot_i/cmd_vel - subscribes to velocity commands to drive the model of the robot

Odometry node
The odometry node listens to the simulated odometry produced by the Stage. It provides the actual position for the robot node as a service. Reading odometry data in the separated node is much faster instead of reading it from the Robot node. The Odometry node runs with different (unlimited) frequency to receive data with the lowest possible delay.

- topics
  - /robot_i/odom - subscribes to the odometry published by the Stage

- services
  - /robot_i/get_robot_state - provides the actual robot position

Robot node
The robot node listens to the topics published by the Stage and receives odometry data and data from the sensors as laser scans. Its main goal is to gather the information about the environment. This information is published after a transformation to a global map which is shared between robots. There is a relation among the robot and stage nodes. The topics published by the stage node are primarily subscribed by the robot node and otherwise.

- topics
  - /robot_i/base_scan - subscribes to the scans from the laser range finder
  - /robot_i/path - subscribes to the paths from the planner
  - /robot_i/cmd_vel - publishes the velocity commands
  - /laser_scan - publishes the laser scan to the global map
- services
  - /robot_i/get_robot_state - requests the actual robot positions
  - /robot_i/get_log_path - provides the logged paths

The robot is driven to move along a received path from a planner using the Smooth Nearness Diagram (SND) algorithm [20] driver with which the robots are able to avoid obstacles. On the other hand, SND slows down the robot whenever it is close to obstacles. An another type of the robot navigation is to move the robot in discrete jumps along a received path. This robot navigation is not affected by the problems with SND control. This approach is suitable for comparison of the exploration strategies without the influence of robot control. A disadvantage of this approach is that it is not realistic and the robots are not protected against collisions. The Stage node has been modified to subscribe to the topic with robot positions:

- topics
  - /robot_i/cmd_pos - subscribes to position commands to change the position of the robot

Map node
The map node collects data from the individual robots and combines it to the global map. This node also performs some map processing, e.g., point reduction. It maintains the map in a format of two separated polygonal structures. The first one is for a free space and the second one contains obstacles. Both are finally merged into the map of the environment. The reason of this approach is described later in Section 4.3.1. The polygon clipping happens in the map node. It provides the map for the planner and publishes some visualizations.

- topics
  - /laser_scan - subscribes to the individual scans from the robots
  - /map_global - publishes the global map for all the robots
  - /visualization_marker - publishes visualizations related to the map

Planner node
The planner node is the central node controlling the whole exploration. The node receives the map and based on the exploration strategy it commands the robots. The following strategies has been implemented (see Section 4.3 for implementation details):

- Greedy strategy
- Greedy-ble strategy
- Hungarian strategy
- K-means strategy (Solanas et al.)
- Region strategy (Puig et al.)
- Segmentation strategy (Stachniss et al.)

The Hungarian strategy uses the C implementation of the Hungarian algorithm by Stachniss [21]. This node is responsible for creation of a log file containing information about the exploration progress. The main exploration loop is described in Scenario 6.

while *the global map contains frontiers* **do**
  receive the global map;
  offset the polygons in the map;
  request the robot positions;
  create the goal candidates;
  apply the exploration strategy to assign the goals to the robots;
  plan paths for the robots;
  send the goal paths to the robots;
  create log entry;

Scenario 6: Planning loop.

- topics
  - /map_global - subscribes to the global map
  - /robot_i/path - publishes the goal paths to the robots

  - /planning - publishes visualizations related to the planning

- services
  - /robot_i/get_robot_state - requests the actual robot positions
  - /robot_i/get_log_path - requests the logged paths

Rviz node
The rviz node visualizes the important parts of the exploration. It subscribes to specific topics and draws the map, robot positions, paths, frontiers, etc.

- topics
  - /planning - subscribes to visualizations related to the planning
  - /visualization_marker - subscribes to visualizations related to the map
  - /map - subscribes to the same map as defined in the world file

4.3 Implementation details

The strategies described in Chapter 2 work on the occupancy grid. Because of the polygonal approach they can't be implemented exactly as the original. This section describes changes made in the strategies, specific algorithms for the polygonal domain and libraries working on polygons used as part of the framework. First, common techniques necessary to perform in the exploration are proposed. Second, specific changes in the strategies are described.

4.3.1 Map representation

In this thesis, all objects are represented as polygons of different sizes. It is possible to maintain the all-in-one map as a set of polygons with some edges marked as frontiers. This complicates the manipulation with polygons mainly in the polygon simplification described below in Section 4.3.3. Another problem is with the frontiers that are very close to obstacles. Because of a laser range finder or odometry errors the frontier may be generated along the obstacle or slightly in the obstacle and it is not possible to reach the goal generated there.

The most robust approach proved to be the separation of a free-space map and obstacle map. Both are represented as sets of polygons independently. Whenever a new scan is added into the map it is added into the free-space map as is. The scan is next checked if it contains obstacles. If so, the obstacles are a little bit offset (proportionally to the map size) and added into the obstacle map. The offset is performed due to the error described above.

Before the map is used by the planner both maps are temporary combined together into a single map. The modified clipping is performed here and the resulting map contains the information about frontiers. If the resulting edge comes from the free-space map it is considered as a frontier, while it is marked as an obstacle if it comes from the obstacle map.

4.3.2 Path planning

The main problem, common to all the strategies, is that it is not possible to directly apply any path planning algorithm. The polygonal description is not sufficient because it offers a map contour only. The visibility graph is useful for path planning. With this instrument it is possible to find the shortest path between the specific locations inside the polygon. Given a set S of n polygon edges, the visibility graph G_S is the undirected graph that has the vertices of the polygon as nodes. Two nodes are adjacent if they can be connected with a line disjoint from all edges in S or they are contained in an edge. In other words, two nodes are adjacent if they see each other.

There are several ways how to compute it and the method posted by Overmars [22], the one used in this thesis is described in this section. The comparison made by Kitzinger [23] shows that although the selected method is not optimal it is fast and easy to implement. It runs in a time $O(m\ log\ n)$ where m is the number of edges in G_S The visibility graph computation is started at the vertex for which a scanline is moved from $-\pi/2$ to $\pi/2$ then it proceeds to the vertex identified in its path. An example of visibility graph is in Figure 4.2

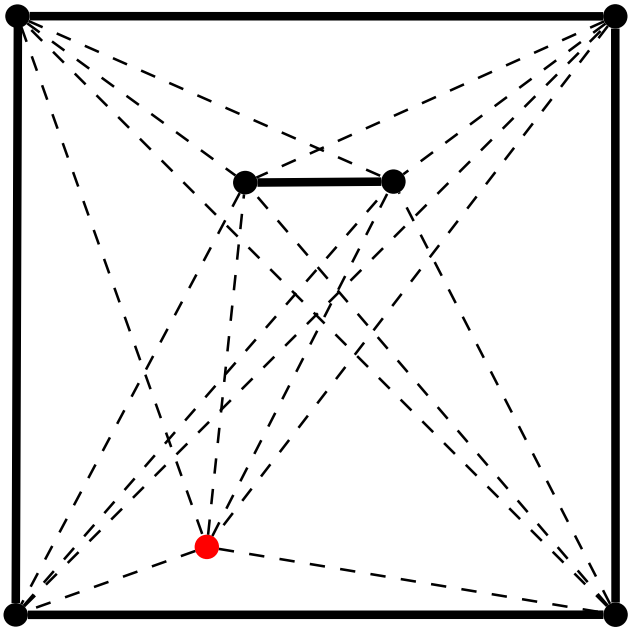


Figure 4.2: Visibility graph.

Dashed lines are edges of the visibility graph. The red node is the robot position which is included in the graph. Having all robot and goal positions as nodes in the graph, it is possible to find the shortest path from any robot to any goal.

This thesis uses the C implementation of visibility graph algorithm based on rotation trees [22] provided by the thesis supervisor. Although the implementation of Overmarses algorithm is straightforward, handling collinear edges is problematic because the scanline may reach any of

the set of collinear vertices. The solution is to look ahead and move to the last of the collinear points, remembering the nearest one or eliminate the collinear vertices at the beginning of the visibility graph computation. At last, the algorithm is defined for non-intersecting edges.

The path planning is performed on the visibility graph which is created on offset polygons. The polygon offsetting was previously described in Section 3.1.2. Notice that the polygon offsetting happens in order to create a feasible paths for robots. Dijkstra's algorithm is used to find the shortest path between two nodes in the visibility graph.

4.3.3 Polygon simplification

Since a level of a map detail is not set in advance, the number of vertices is increasing enormously. After each addition of a laser scan to the map, the total number of vertices increases. In fact, the most of the vertices is useless for a description of the environment because they are collinear. It is preferable to have the number of vertices as lowest as possible because of memory requirements and a time complexity of the individual algorithms, e.g., path planning, visibility graph computation, etc.

There exist many approaches how to remove redundant vertices from a polygon. The most of them is based on a tolerance, i.e., a threshold defining redundant vertices. Specification of the appropriate threshold is most difficult part of the algorithm. If a tolerance is too small the redundant points are not removed and a high tolerance may deform the polygon. Removing an important vertex may cause irreversible defects in the polygon. The first attempt in this thesis was to find the optimal tolerance. Every three consecutive vertices v_1,v_2,v_3 were compared according to a criterion:

$$[e(v_1, v_2) + e(v_2, v_3)] - e(v_1, v_3) < \varepsilon$$

where e is the euclidean distance and ε is the threshold. If the vertices satisfy the criterion the vertex v_2 is considered to be removed because it lies on the line between the two neighbouring vertices v_1 and v_3. Figure 4.3(a) depicts which euclidean distances are computed. This method proved to be unsuitable for big maps with high detailed areas.

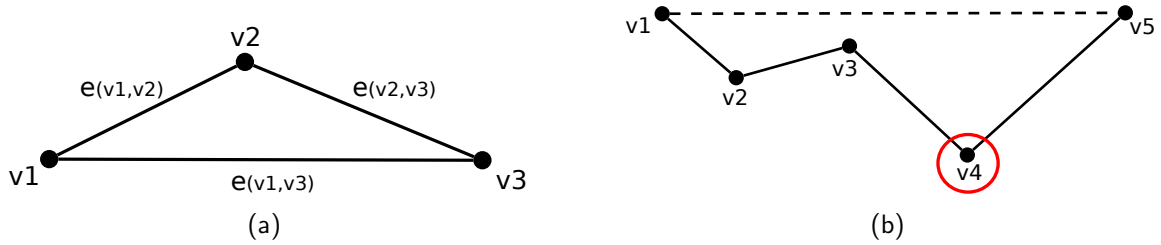


Figure 4.3: Polygon simplification: (a) criterion based on a comparison of edge lengths using a tolerance; (b) RDP - recursive algorithm for the polygon simplification;

The Ramer-Douglas–Peucker algorithm was the second method of the polygon simplification. Given a series of consecutive points it creates a line between the first and the last vertex that

are always kept. It then finds the furthest vertex from the line (see Figure 4.3(b) where the furthest vertex v_4 is marked by the red circle). If the distance is smaller than a defined tolerance ε any points between the start and the end point may be removed. If the distance is greater than ε the point is kept and the algorithm is recursively called with the first vertex and the furthest vertex and also with the furthest vertex and the last vertex. Figure 4.3 depicts the both approaches.

Finally a modified RDP algorithm is used in this thesis. The algorithm normally marks the vertices that will be kept. Moreover, the vertices that lie on a boundary between obstacles and frontiers are always kept. The marked vertices are then checked if there is a frontier in some distance, e.g., in a laser range distance. Any point within the distance is also kept and not affected by the simplification algorithm. This feature allows to set greater ε without negative effects because a deformation happens frequently on the boundary.

4.3.4 Goal candidates

Having a series of adjacent frontier edges (frontier) it is necessary to reduce the frontier into goal candidates. The goal candidates are the places that must be visited by robots in a defined order. The candidates are commonly distributed in a sensor range which leads to the effective covering of an unknown space. On the occupancy grid a frontier cell lying in the defined distance is simply marked as the goal candidate.

On polygons a vertex does not have to exist in the distance. In this case a temporary vertex is added and used as the goal candidate. The temporary vertices are inserted only for planing purposes and are lost after the planning. The algorithm must count with several situations that may happen. While processing edges of the frontier, several goal candidates may be inserted on a single edge or the edge can be completely skipped. A goal candidate may also correspond with an existing vertex and the vertex is then marked as the candidate.

Figure 4.4 shows a frontier with generated goal candidates (the red points) with all the situations described above. The frontier is defined by the vertices (the black points) with assigned labels and starts with the vertex v_1 and ends with the vertex v_5. The goal candidates are distributed in a sensor range distance R. The first goal candidates from the start and the end are in $R/2$ distances so the frontier is completely covered. The vertex v_4 is marked as the goal candidate.

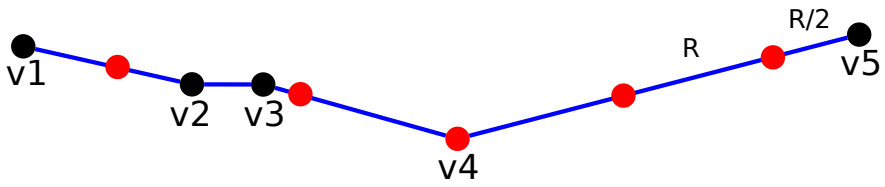
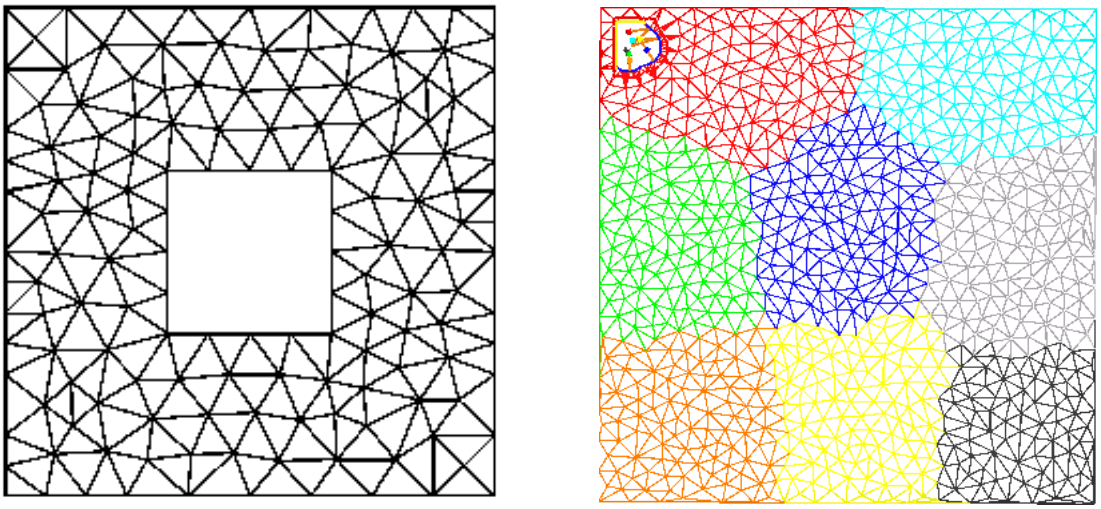


Figure 4.4: Goal candidates generated on a frontier.

Because the visibility graph can not handle collinearity goal candidates are slightly shifted in a perpendicular distance from the edge in order to be not collinear. This feature has no side effects and should be removed as soon as the algorithm will be repaired.

4.3.5 K-means based strategies

The exploration strategies using K-means clustering described in Chapter 2 needs to partition the unexplored space into K regions with similar volumes, which is not straightforward in a polygonal representation. Our idea is therefore to represent the unexplored space with a set of points. The points are then passed to the K-means clustering algorithm. For sampling the polygons with points the Triangle library [24] is used which is a two-dimensional mesh generator. A triangle mesh is a set of triangles which are connected by their edges or vertices. The Triangle is able to generate triangle meshes, (constrained) Delaunay triangulations, Voronoi graphs, etc. Its features include user-specified constraints, e.g., a triangle area.



(a) (b)

Figure 4.5: Triangular meshes: (a) an example of triangle mesh; (b) a triangle mesh with applied K-means clustering algorithm at the beginning of an exploration with 8 robots using the K-means strategy;

Figure 4.5(a) shows an example of a generated triangle mesh on a polygon with hole. Figure 4.5(b) visualizes a triangle mesh with applied K-means clustering. The number of clusters equals to the number of robots. The vertices of the triangles are shared among the other triangles. The generated points covering the polygons are formed by the unique vertices from all the triangles. The triangle area constraint allows to set the point density. The number of generated points is not critical in our case so the point density is determined proportionally to the map size.

Chapter 5

Experiments

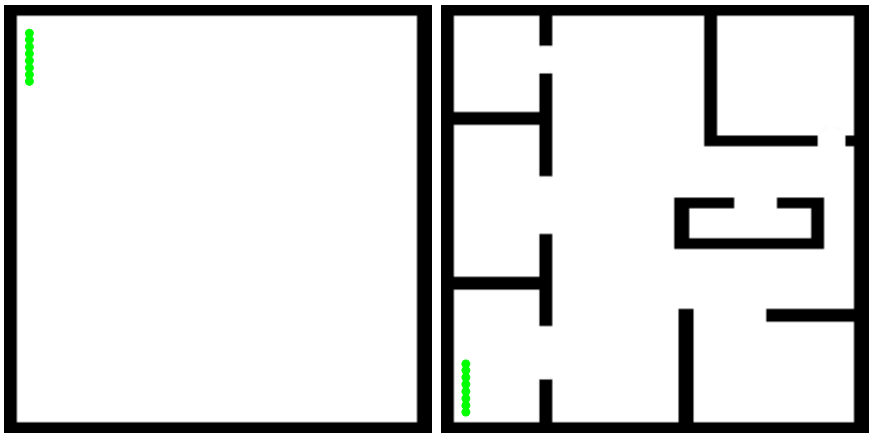
This chapter aims to combine the previous work on the framework described in Chapter 4 with the necessary modifications described in Chapter 3 using the exploration strategies presented in Chapter 2 to test the framework functionality. The second goal of this chapter is to compare the selected strategies and describe their performance on various problems.

5.1 Experiment setup

The experiments have been performed using maps with various sizes and structures. The Empty map 5.1(a) has been created to simulate a trivial case of a big room with no obstacles. The Arena map 5.1(b) represents a slightly structured environment with large corridors and rooms. The Jari-huge map 5.2(a) represents the real administrative building with many separated rooms. The hospital small map 5.2(b) is a part of the hospital-section map from the Stage simulator. However it is a section, it is the largest map in the main experiment. The robot starting positions are marked by the green circles and are located close to the entrance.

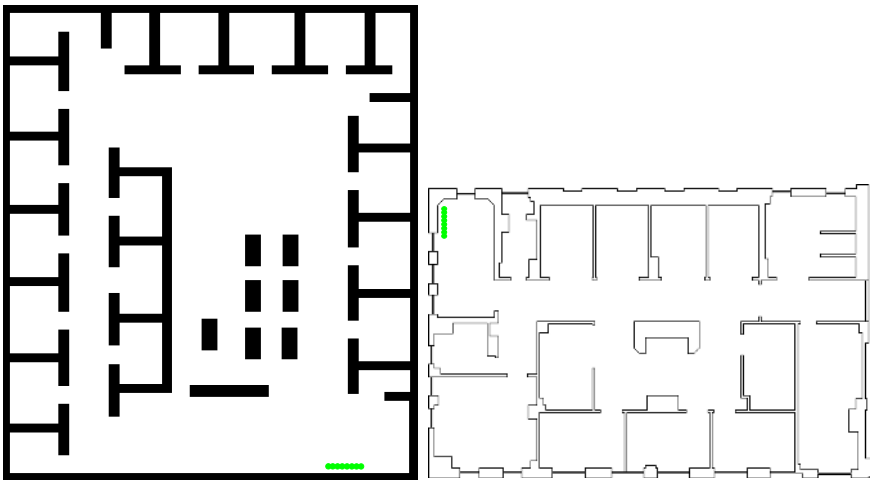
The strategies used in the experiment are selected from the list in the planner node description in Section 4.2, i.e., Greedy, Greedy-ble, Hungarian and K-means. The remaining strategies are not considered because of implementation issues. The Segmentation strategy has been implemented but the critical points detection is far more complicated on real maps. The Region strategy is not implemented exactly as on the occupancy grid. There is nothing like a cell adjacent to a frontier, vertices are in varying distances, etc. Both strategies are then rejected because of poor results that are not caused by the strategies themselves.

Furthermore the K-means works similar to the Region strategy and takes the robot dispersion in the environment into account. The first three strategies are in fact greedy or greedy-like with some adjustments. The numbers of robots are $4, 6, 8$, while the sensor range is set to 5 meters with 270° field of view. The robots are controlled using the SND driver. The planning period has been set to 1 second.



(a) (b)

Figure 5.1: Simple maps used in the thesis with robot starting positions marked as the green circles: (a) Empty map with dimensions 50x50 m; (b) Arena map with dimensions 50x50 m.



(a) (b)

Figure 5.2: Complex maps used in the thesis with robot starting positions marked as the green circles: (a) Jari-huge map with dimensions 52.5x60 m; (b) Hospital-small map with dimensions 138x110.75 m.

5.2 Evaluation of strategies

The strategies are evaluated from several points of view. The first criterion is the number of planning steps t_{exp} which corresponds with the exploration time (assuming that the planning period is constant). The exploration time is the time needed to explore the whole unknown environment so the map does not contain reachable frontiers. The maximal distance d_{max} travelled by a robot is the second criterion. This is important when sources of individual robots are limited, e.g., energy.

In search and rescue problems it is crucial to find potential victims instead of mapping the environment. It is needed to explore as much area as possible in the shortest time instead of complete and detailed mapping. A criterion which takes into account an effort to find the object [25] is inspired by the expected value of the random variable defined as:

$$E(X) = \sum_{i=0}^{n} x_i p_i \tag{5.1}$$

According to the expected value the mean effort needed to find the object is the weighted average of n exploration steps:

$$T_M = \sum_{i=0}^{n} t_i w_i \tag{5.2}$$

where t_i is the exploration step (time) and w_i is the amount of map explored since the previous step by which the step is weighted. All the criteria are presented as the arithmetic mean (average) together with their variances describing how far the measures lie from the average.

5.3 Methodology

All the experiments were examined on the same hardware with a quad-core processor on 3.30 GHz, 8 GB RAM running x86_64 GNU/Linux kubuntu 3.0.0-20, ROS version electric and gcc version 4.6.1. Each experiment was repeated 30 times. The number of experimental runs is 1440 in total. With approximately 10 minutes for a single run it would take about 240 hours. As the computations are not time consuming, the experiments are speeded up 3 times in the Stage's configuration file. This acceleration has no effect on the quality of the exploration but it has its limits in the computational complexity. The chosen acceleration is therefore a trade-off between the run time and the system load.

5.4 Results

The tables contain the average value of the number of planning steps with explicitly stated the minimal and the maximal values, the standard deviation computed and presented also in graphs as a black y-bar, the maximal distance travelled by a single robot and the expected time needed to find the object. The graphs contain only some of these parameters.

For a simplicity, the presented graphs with map built time are generated only for 6 robots and K-means strategy. The map built time is affected mainly by the map dimensions and the map complexity yielding the number of vertices. It can be seen that the map built time curve does not always exactly correspond with the number of vertices because it depends also on the polygon shapes that affect the complexity of the modified clipping process. The map built time includes the modified polygon clipping and the polygon simplification. This time should not exceed the planning period, otherwise the planner will not have the actual map. The results are presented separately for each map and discussed in Section 5.7.

5.4.1 Empty map

There are not big differences in the exploration time between the Hungarian and the K-means strategy. Both yield a similar exploration time. A surprise is the good performance of the Greedy strategy. Although the greatest number of planning steps was expected, the mean effort T_M is at least comparable to the other strategies. Because there are no obstacles in the map the robots head towards goals with no need of coordination and the robots naturally disperse in the environment. The effect of the BLE algorithm in the Greedy-ble strategy is significant in each experiment.

The maximal distance d_{max} apparently indicates the effect of the map segmentation in the K-means strategy. The robots focus their regions which reduce the maximal travelled distance by any robot that is smaller by 7% than in Hungarian strategy but it has longer exploration time by 5.4% for 6 robots. For 8 robots, the K-means strategy is better in both of the parameters.

The simple structure of the map produces low number of vertices and leads to the small map built time, see 5.5(d). The K-means strategy has also the lowest standard deviation so the solution has not a big variance.

Robots	Strategy	t_{exp}	t_{min}	t_{max}	$\sigma_{t_{exp}}$	d_{max}	$\sigma_{d_{max}}$	T_M	σ_{T_M}
4	greedy	960.2	790	1186	93.39	567.99	55.90	329	16.32
	greedy-ble	909.8	824	1042	52.65	538.32	39.45	330	20.14
	hungarian	893.0	804	1001	51.06	526.85	33.50	328	13.31
	kmeans	903.5	783	998	44.37	484.84	30.12	348	33.67
6	greedy	744.7	595	923	89.02	458.50	54.94	263	12.94
	greedy-ble	682.6	607	838	49.54	422.43	32.12	262	12.17
	hungarian	667.7	561	921	63.20	412.40	30.15	257	18.63
	kmeans	705.6	665	779	25.96	383.51	21.32	279	7.65
8	greedy	693.6	564	839	74.54	429.67	49.41	234	11.37
	greedy-ble	629.4	517	846	75.36	377.19	45.07	245	19.43
	hungarian	579.2	512	641	34.91	357.16	29.63	229	10.01
	kmeans	575.3	517	618	22.50	310.19	17.03	238	7.61

Table 5.1: Map empty: Comparison

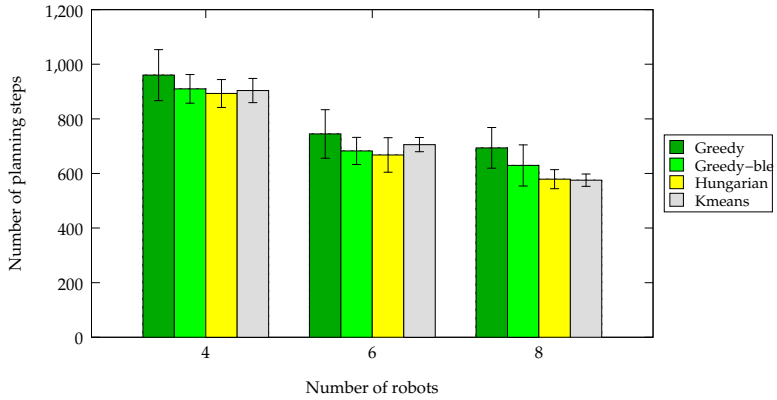


Figure 5.3: Empty map - planning steps comparison.

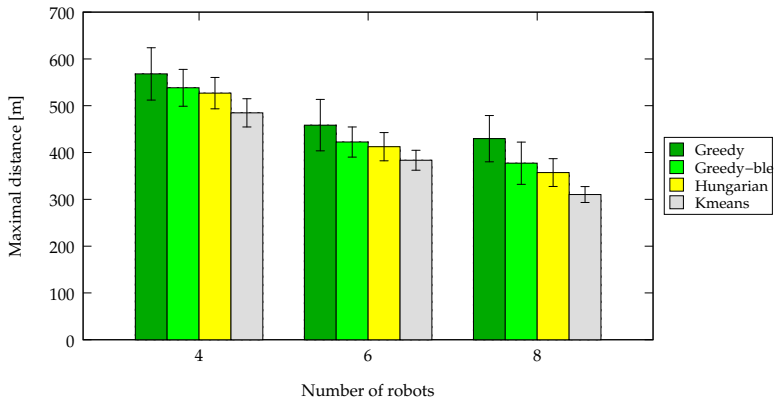


Figure 5.4: Empty map - maximal distance comparison.

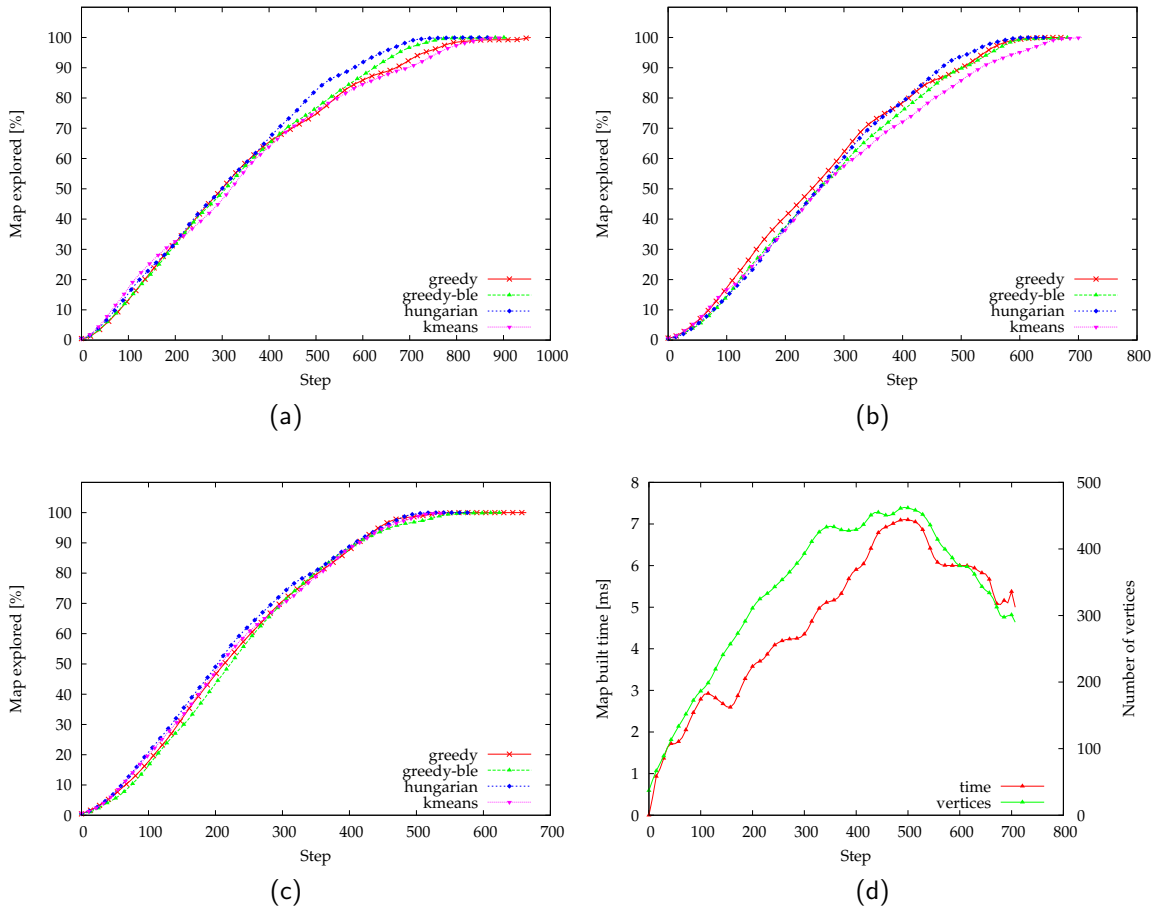


Figure 5.5: Empty map size progress: (a) for 4 robots; (b) for 6 robots; (c) for 8 robots; (d) shows the map built time with the number of vertices.

5.4.2 Arena map

The results are similar to the results on Empty map with one difference. Here the greedy strategy does not excel in T_M which is caused by the existence of the rooms that prevent the natural dispersion of the robots. Although for 4 and 6 robots the best results are achieved by the K-means strategy an interesting effect can be seen for 8 robots where the results are worse. The partitioning of map forces the robots to explore small regions partially spreading over two or three rooms which slows down the exploration, while for the lower number of robots the regions are bigger and contain the whole rooms.

Robots	Strategy	t_{exp}	t_{min}	t_{max}	$\sigma_{t_{exp}}$	d_{max}	$\sigma_{d_{max}}$	T_M	σ_{T_M}
4	greedy	1278.4	1046	1567	138.76	588.33	65.39	494	46.67
	greedy-ble	1197.5	994	1708	143.63	572.64	58.33	476	29.19
	hungarian	1155.5	988	1305	81.86	556.12	50.08	473	27.84
	kmeans	1139.2	1053	1394	84.95	546.10	50.27	448	20.82
6	greedy	1023.0	741	1382	143.27	473.39	55.86	394	40.63
	greedy-ble	900.4	761	1014	66.75	435.57	36.44	377	25.64
	hungarian	867.1	783	1050	65.23	429.18	31.70	361	17.70
	kmeans	848.4	748	1112	74.34	427.65	37.85	365	24.30
8	greedy	987.5	791	1378	150.74	439.90	57.56	377	44.96
	greedy-ble	807.3	677	977	82.61	395.00	47.62	337	20.31
	hungarian	748.3	627	985	77.04	374.67	44.65	309	12.73
	kmeans	788.0	681	925	72.37	403.98	31.93	322	18.65

Table 5.2: Map arena: Comparison

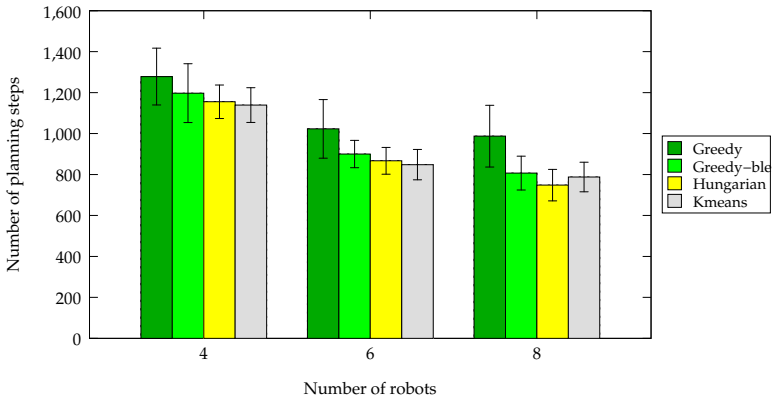


Figure 5.6: Arena map - planning steps comparison.

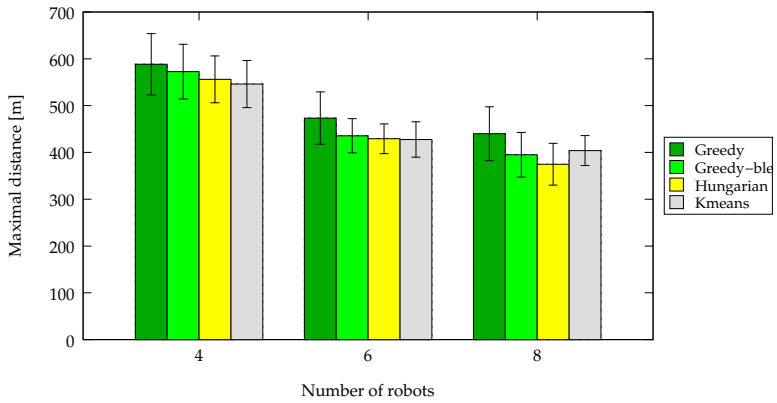


Figure 5.7: Arena map - maximal distance comparison.

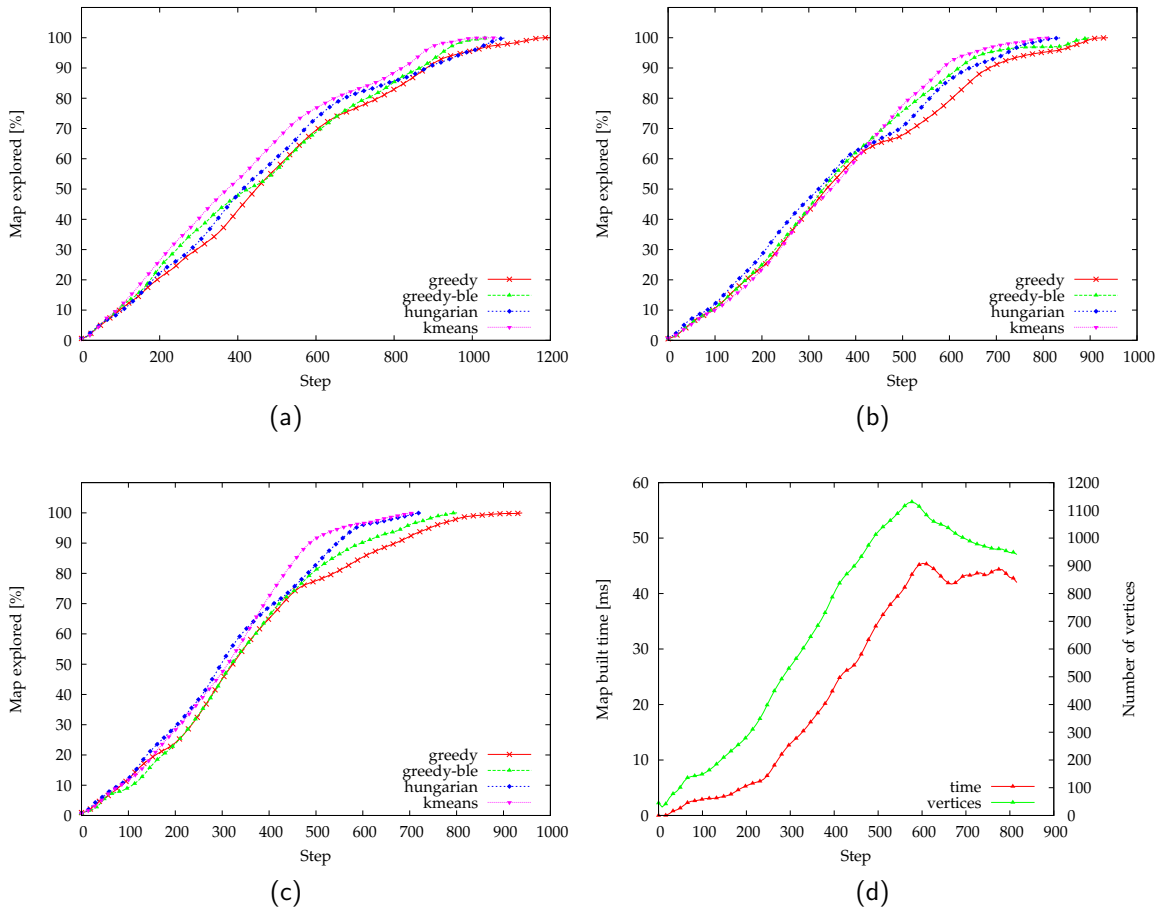


Figure 5.8: Arena map size progress: (a) for 4 robots; (b) for 6 robots; (c) for 8 robots; (d) shows the map built time with the number of vertices.

5.4.3 Jari-huge map

The results show that the K-means strategy behaves worse on this map. A lot of small rooms leads to the regions generated over several separated rooms. A robot must explore the assigned rooms in the region alone. This is in fact the desired behaviour that each robot visits a single room but it requires correctly generated regions including the whole rooms not only their parts. The Hungarian strategy is far more better in comparison with the other strategies. For 8 robots it is faster by 28% than the Greedy strategy, by 16% than the Greedy-ble strategy, and also by 12% than the K-means strategy.

Robots	Strategy	t_{exp}	t_{min}	t_{max}	$\sigma_{t_{exp}}$	d_{max}	$\sigma_{d_{max}}$	T_M	σ_{T_M}
4	greedy	558.4	459	750	78.75	247.85	34.89	199	13.89
	greedy-ble	489.2	433	568	36.75	218.17	19.55	193	28.67
	hungarian	473.4	406	553	39.80	210.93	19.68	179	32.58
	kmeans	524.0	470	573	31.10	235.21	15.68	203	19.06
6	greedy	488.8	365	848	138.15	208.14	52.23	169	27.49
	greedy-ble	369.9	311	439	31.62	165.14	17.27	148	12.50
	hungarian	336.2	301	402	26.84	150.98	13.34	142	9.38
	kmeans	365.7	334	453	26.59	163.22	12.77	157	15.04
8	greedy	400.9	285	573	79.82	176.38	30.51	156	34.22
	greedy-ble	345.6	281	471	42.30	159.77	20.61	136	13.83
	hungarian	289.8	258	331	19.70	133.43	9.58	124	14.66
	kmeans	328.5	285	371	21.96	149.48	12.66	140	6.14

Table 5.3: Map jari: Comparison

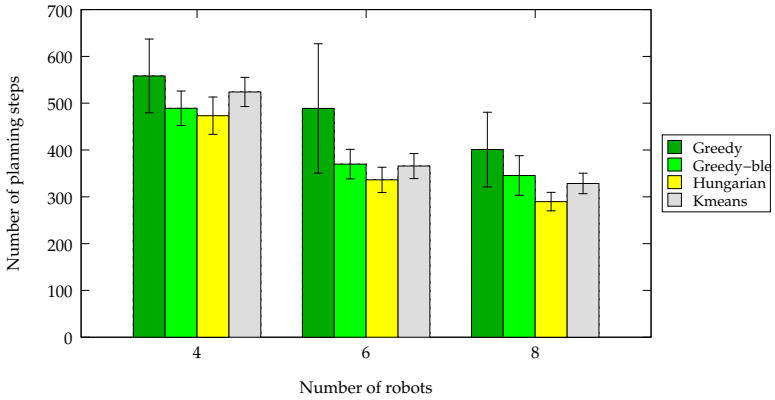


Figure 5.9: Jari huge map - planning steps comparison.

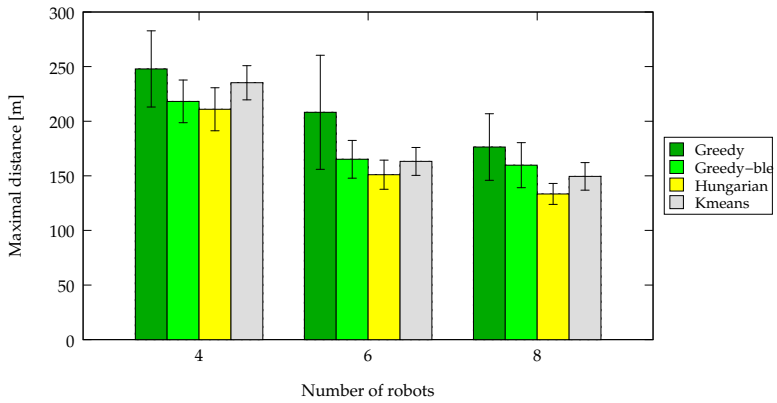


Figure 5.10: Jari huge map - maximal distance comparison.

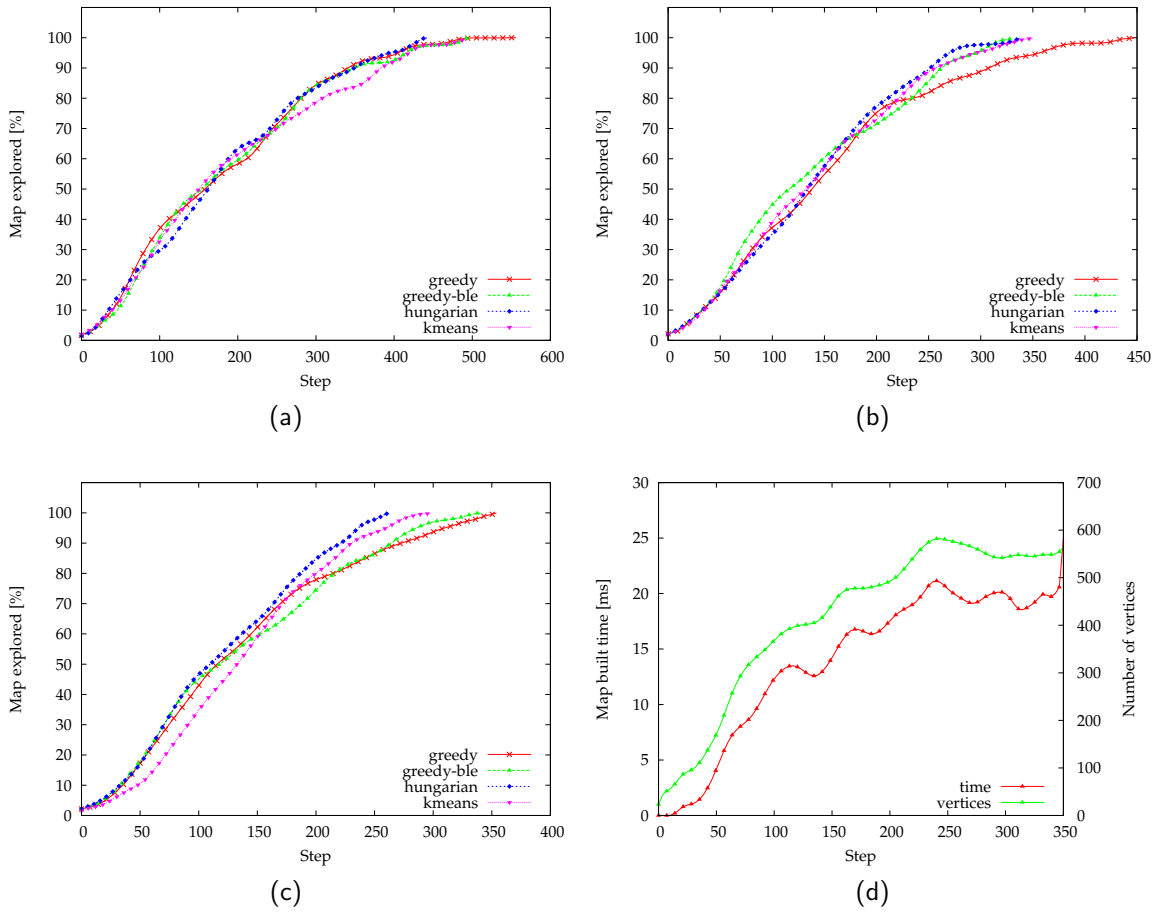


Figure 5.11: Jari-huge map size progress: (a) for 4 robots; (b) for 6 robots; (c) for 8 robots; (d) shows the map built time with the number of vertices.

5.4.4 Hospital-small map

The results are quite balanced except the Greedy strategy. The exploration time and the maximal travelled distance is slightly better for the K-means strategy and 4 robots. The K-means strategy is better in the mean effort parameter. This experiment confirms that there exists a combination of the map size and the number of robots in which the K-means strategy yields bad results (6 robots in this case) because the number of robots corresponds with the sizes of regions that may be inappropriate for the given map. The Hungarian strategy again seems to be the best strategy.

Robots	Strategy	t_{exp}	t_{min}	t_{max}	$\sigma_{t_{exp}}$	d_{max}	$\sigma_{d_{max}}$	T_M	σ_{T_M}
4	greedy	798.5	704	901	62.97	455.31	39.09	375	14.43
	greedy-ble	736.2	658	876	64.56	425.88	40.18	339	11.75
	hungarian	726.1	648	861	55.39	430.05	30.26	338	24.12
	kmeans	728.1	686	781	28.88	423.44	19.47	312	7.41
6	greedy	644.4	538	779	77.01	374.48	53.31	300	18.96
	greedy-ble	605.9	543	672	45.99	364.79	25.34	282	15.08
	hungarian	576.1	532	611	29.13	350.71	14.66	261	9.05
	kmeans	637.2	605	667	19.04	384.39	13.18	271	7.81
8	greedy	568.9	500	653	49.18	343.54	31.74	270	19.13
	greedy-ble	567.7	519	628	35.23	345.94	25.43	271	9.48
	hungarian	533.9	458	613	44.92	331.44	23.53	247	12.79
	kmeans	560.7	519	606	26.92	346.78	18.69	235	10.48

Table 5.4: Map hospital small: Comparison

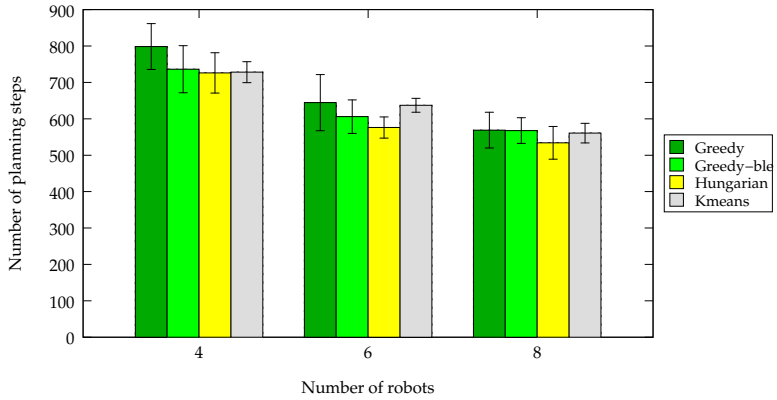


Figure 5.12: Hospital small map - planning steps comparison.

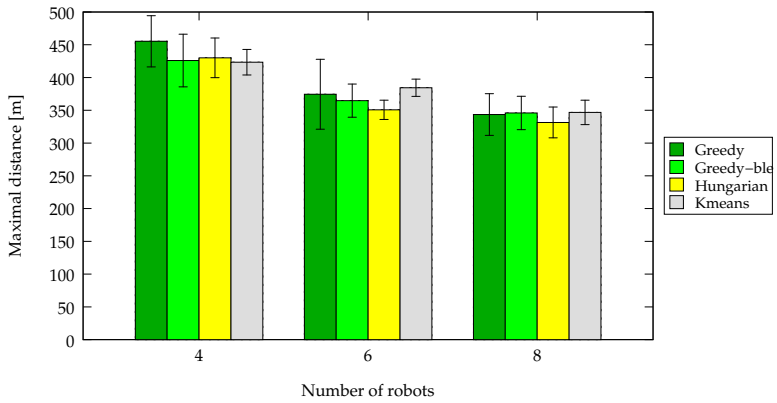


Figure 5.13: Hospital small map - maximal distance comparison.

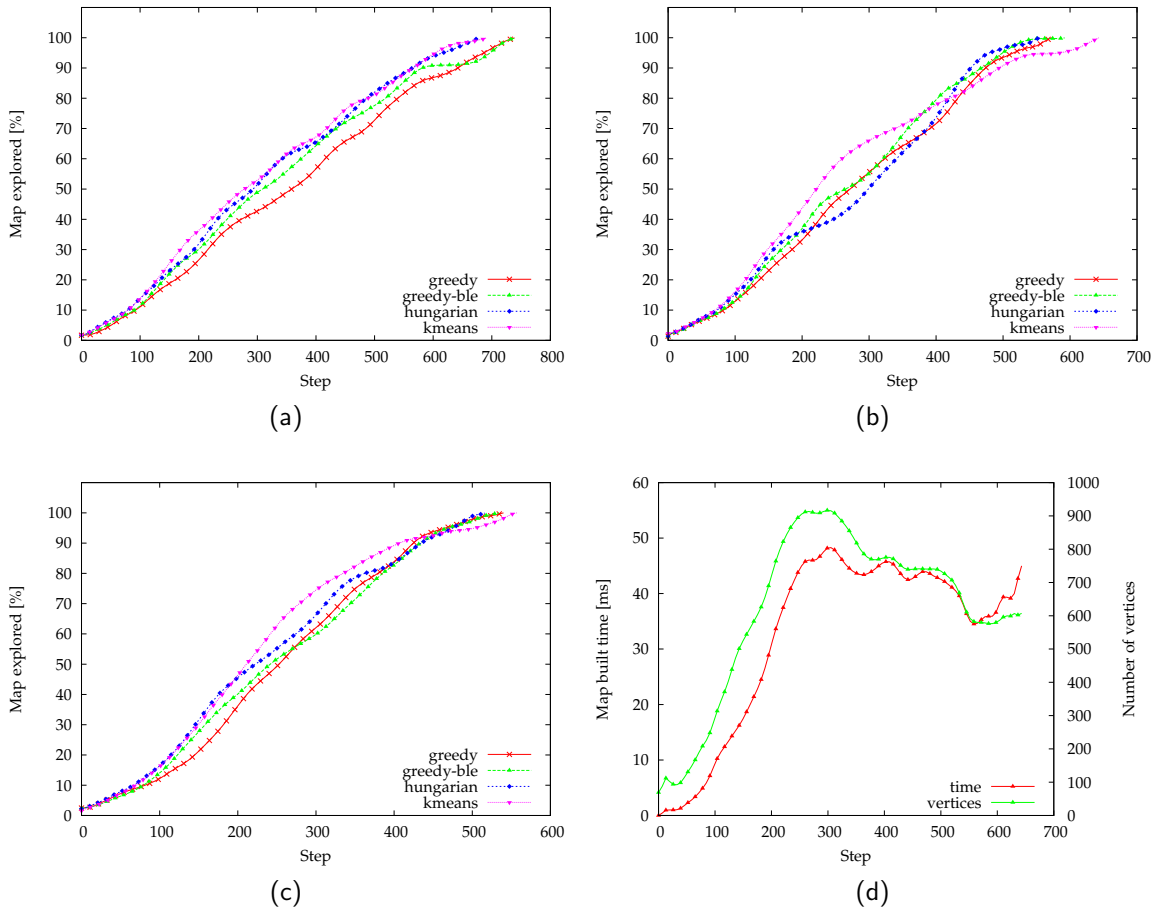
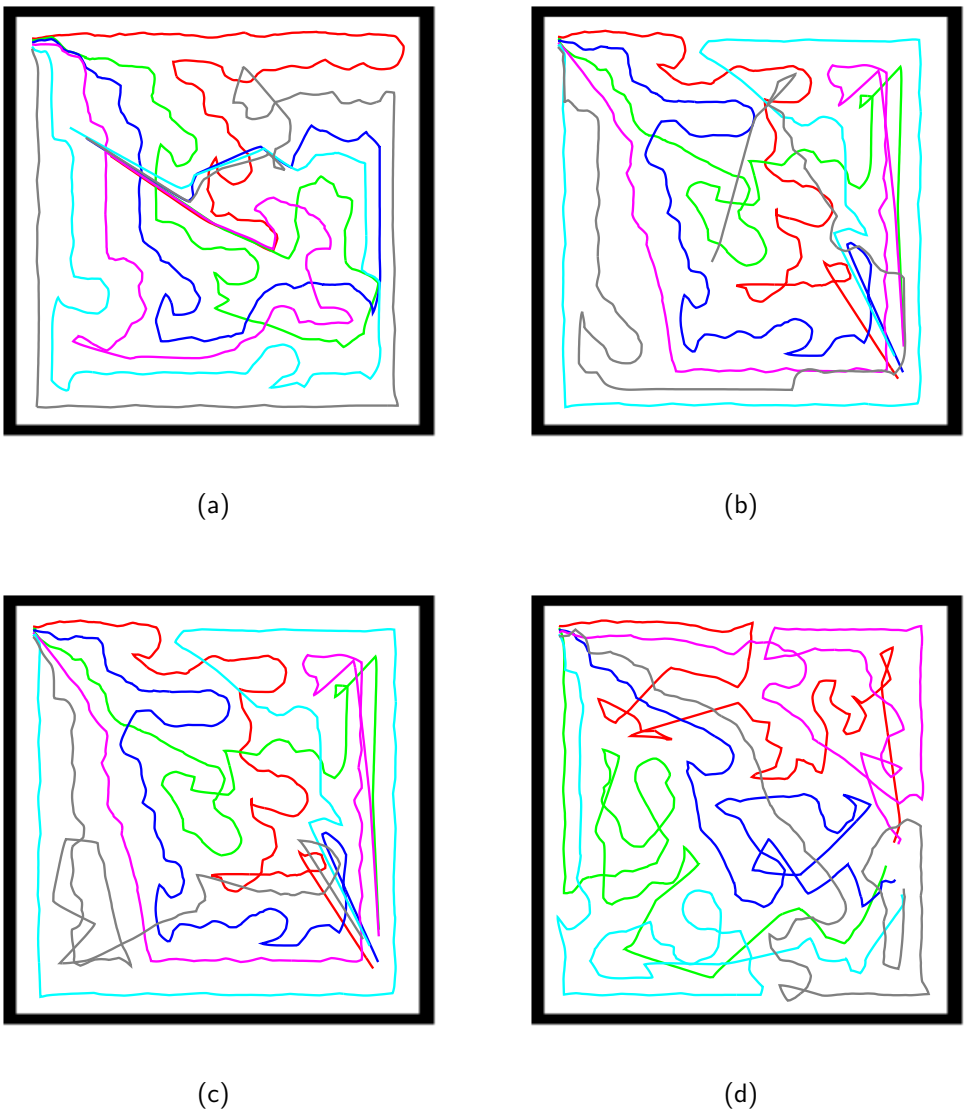


Figure 5.14: Hospital-small map size progress: (a) for 4 robots; (b) for 6 robots; (c) for 8 robots; (d) shows the map built time with the number of vertices.

5.5 Robot paths

Robot paths from the selected experiment are visualized in Figure 5.15. The selected experiment is the exploration with 6 robots on the Empty map for simplicity. The figures show that for K-means strategy robots operate in their regions and when they finish they proceed to the further regions. The other greedy-like strategies have more duplicate paths or crossings. Each strategy is visualized separately.

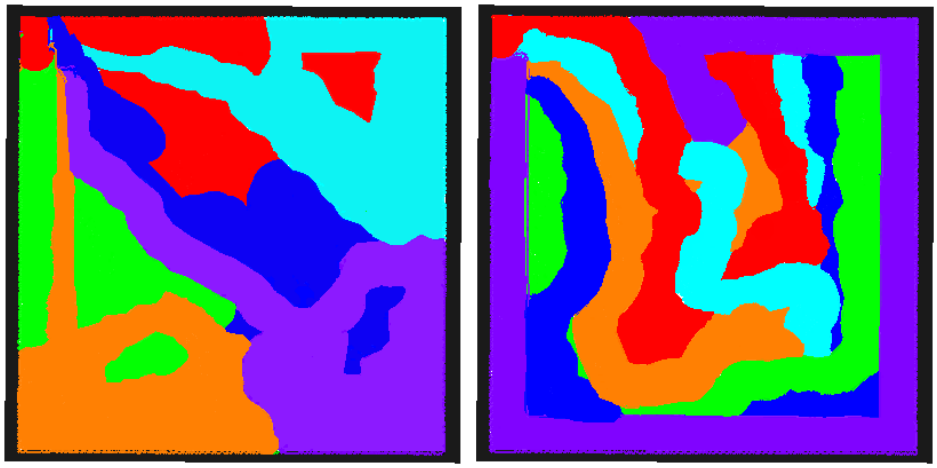


(a) (b)

(c) (d)

Figure 5.15: Robot paths for selected experiments with 6 robots on the Empty map: (a) Greedy strategy; (b) Greedy-ble strategy; (c) Hungarian strategy; (d) K-means strategy.

The areas explored by the individual robots are visualized in Figure 5.16. With the polygonal representation it can be easily done with the Difference clipping operation provided by the clipping library. A new scan is clipped against the already explored map and the result contains the new explored area which is stored, while keeping the information about the author of the scan. The areas distinguished with colours apparently correspond with the robot paths depicted above in Figure 5.15. Only the two strategies has been selected as an example.



(a) (b)

Figure 5.16: Robot explored areas by 6 robots: (a) using the K-means strategy; (b) using the Hungarian strategy.

5.6 Hospital section map experiment

To simulate an extreme case with many robots on a huge map the Hospital-section map is used. The exploration is performed by 10 robots with no other changes against the previous experiments. The starting positions of robots are the same as on the smaller Hospital-section map. This experiment can not be speeded up because of the computational complexity so the number of repetitions is lower, i.e., 10 trials.

Figure 5.18 shows the created map at the end of the exploration. This map occupies an area of 14253 meters squared and contains 1226 vertices. Notice that a map is created during each exploration but it would take a lot of pictures to present them so this map is introduced as an example.

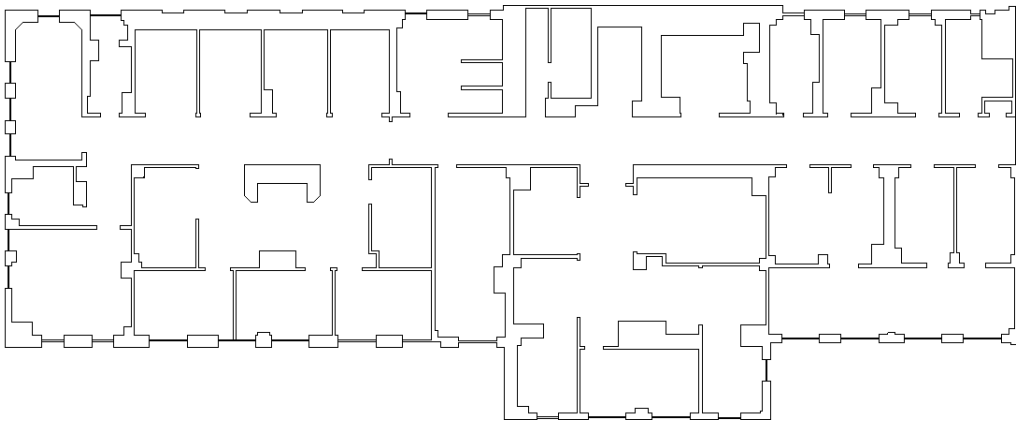


Figure 5.17: Hospital section map used to simulate the extreme case with dimensions 271.5x110.75 m.

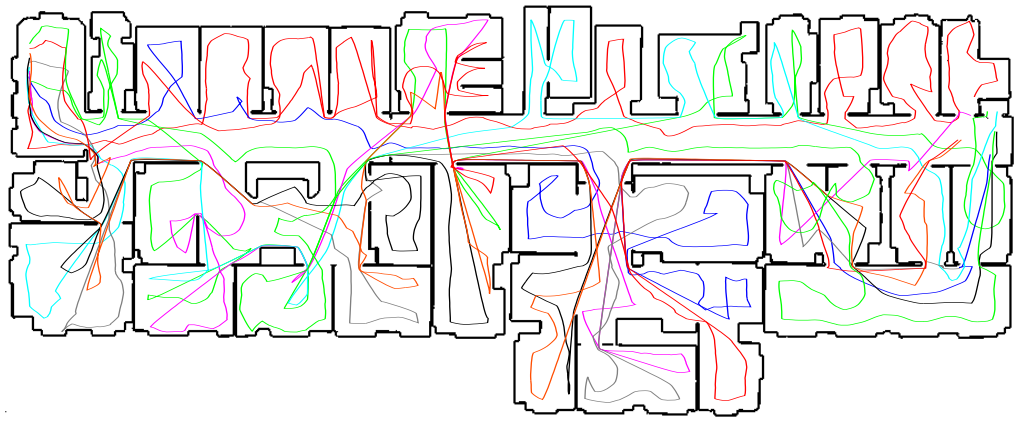


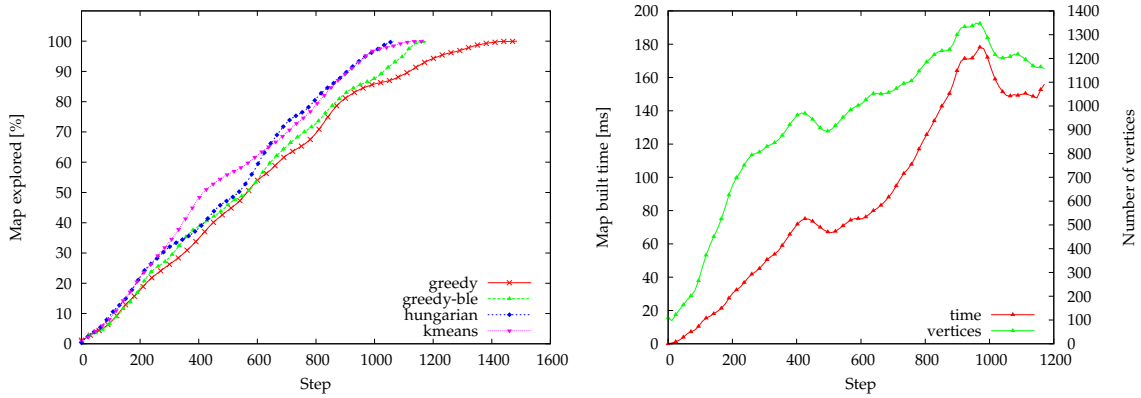
Figure 5.18: Hospital section map explored and visualized using Rviz with paths travelled by 10 robots with the Hungarian strategy.

The results in Table 5.5 show the shortest time needed to find the object of the K-means strategy. Its standard deviation shows that the exploration depends on how the regions are generated. Because of the K-means clustering is a stochastic method and may give different results.

The results of this experiment are in accordance with the results of the previous experiments. The worst and the most varying strategy is the Greedy strategy. The Greedy-ble strategy is better but still not optimal. The best results are achieved by the Hungarian strategy which has also the shortest d_{max}. With this number of robots and the K-means strategy it is possible that a robot visits a lot of goals from different regions until it reaches its region because its assigned region lies on the other side of the map. Such a robot travels a long distance especially when new shorter paths are frequently found.

Robots	Strategy	t_{exp}	t_{min}	t_{max}	$\sigma_{t_{exp}}$	d_{max}	$\sigma_{d_{max}}$	T_M	σ_{T_M}
10	greedy	1365.7	1195	1507	158.05	710.13	81.54	607	13.58
	greedy-ble	1221.7	1148	1294	40.17	697.28	35.22	572	16.91
	hungarian	1098.8	1051	1168	33.27	625.72	24.71	520	17.27
	kmeans	1222.7	1142	1325	48.95	691.16	26.91	515	15.04

Table 5.5: Map hospital section: Comparison



(a) (b)

Figure 5.19: Hospital section map size progress: (a) for 10 robots; (b) shows the map built time with the number of vertices.

5.7 Discussion

The selected strategies have been tested in series of experiments on different environments. As expected, the worst results on every map produces the Greedy strategy. It is a naive strategy where a goal can be explored by many robots with no coordination. It is common that robots form a group in which the robots explore the same room which is not effective. It serves for a comparison of the improvements included in the other strategies. The effect of the preventing selection of the same goal in the Greedy-ble strategy is significant. This strategy outperforms the Greedy strategy in all aspects. But the fact that it can produce a suboptimal assignment causes that it is not so good as the Hungarian strategy. The Hungarian strategy yields the best results in the most of cases. The K-means strategy is full of contradictions. In general it is more efficient for lower number of robots and less complex environments. In majority of cases, the maximal travelled distance is small because the robots focus on their regions and their operating areas are limited.

The reason why the K-means strategy is not so good as expected lies in several facts. First, the regions may be badly generated partially spreading over many separated rooms. Second, a robot heading to its region may select a new different path found by any other robot which is shorter due to the penalization. There is a possibility that the robot will turn back and travel trough the already explored environment in this case. Third, the regions contain a map border and obstacles that affect the positions of the centers of the regions.

The experiments have been performed for a different number of robots. The difference between the performance of the exploration with 4 and 6 robots is obvious. The higher number of robots leads to a noticeable overlap between robot scans. If the size of the environment is big enough, e.g., Hospital map, the additional 2 robots improve the performance of all the strategies.

This work primarily focuses on the implementation of the framework for the multi-robot exploration. The real experiment requires an additional effort to adapt the framework interface to the Syrotek system but thanks to the ROS system modularity it is not a big problem. This has not been realized because of the uselessness of running the experiments in the Syrotek arena because of several reasons. Firstly, the Syrotek arena has dimensions only 3.5x3.8 m and it contains only 3 robots equipped with a laser range finder which is not sufficient for the multi-robot exploration. This thesis also assumes precise odometry data but the real arena realizes the localization using a camera tracking the robots. The real environment does not allow to compare the selected strategies as in the simulator. It can be expected that the framework will work in the real arena with problems typical for the real experiments, i.e., inaccurate localization, data noise, etc.

Chapter 6

Conclusion

This thesis introduces a polygonal approach in the mobile robot exploration. Several exploration strategies were presented in Chapter 2 and implemented in the framework. The Clipper library has been selected for the polygon clipping. The library and its necessary modifications were described in Chapter 3. The framework described in Chapter 4 has been implemented using ROS in C++ language. The framework functionality was tested using some existing approaches. The tests with results are presented in Chapter 5.

A considerable effort was devoted to learn ROS framework. It requires completely different approach against the previous work with the Player/Stage. The hardest task was to modify the clipping library in order to work with the frontiers. After several unsuccessful attempts to change the clipping algorithm a compromise solution was found in the edge matching. In the future will be hopefully found a library capable of work with the edge attributes. Another external libraries were used in this thesis, i.e., for visibility graph, path planning, triangulation, Hungarian method, SND driver, and incorporated into the framework.

The implemented strategies were compared according to various criteria. The best results were achieved by the Hungarian strategy. The K-means strategy was expected to dominate among the other strategies especially in the mean effort parameter. Unfortunately, this criterion does not distinguish locations in the map in which the exploration happens and therefore the difference against the other strategies is not so significant.

The polygonal approach proved to be a great method for the map representation. With a quite low number of points it is possible to represent really big environments. On the other hand, the polygons are sensitive to the correctness of measurement because the explored areas are directly added to the map. All the parts of exploration were successfully adapted to the polygonal representation which was proved by the experiments.

In the future, the framework will be extended by new exploration strategies. Improvements will be made in order to speed up the modified clipping and handle collinearity problem in the visibility graph algorithm. It would be also useful to consider a sensor model when updating the map together with a focus on localization.

Bibliography

[1] W. Burgard, M. Moors, C. Stachniss, and F. Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376–378, 2005.

[2] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proc. of IEEE International Conferenceon Robotics and Automation (ICRA)*, 2000.

[3] Cyrill Stachniss, Oscar Martinez Mozos, and Wolfram Burgard. Efficient exploration of unknown indoor environments using a team of mobile robots. *Annals of Mathematics and Artificial Intelligence (AMAI)*, 52(2-4):205–227, 2008.

[4] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, S. Moors, M.and Thrun, and Younes H. Coordination for multi-robot exploration and mapping. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2000.

[5] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[6] Robot operating system. `http://www.ros.org`, 2012.

[7] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[8] Francesco Amigoni. Experimental evaluation of some exploration strategies for mobile robots. In *ICRA*, pages 2818–2823, 2008.

[9] Brian Yamauchi. Frontier-based exploration using multiple robots. In *Proc. of the Second International Conference on Autonomous Agents*, pages 47–53, 1998.

[10] Barry Brian Werger and Maja J. Mataric. Broadcast of local eligibility for multi-target observation. In *Distributed Autonomous Robotic Systems 4*, pages 347–356. Springer-Verlag, 2001.

[11] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[12] K.M. Wurm, C. Stachniss, and W. Burgard. Coordinated multi-robot exploration using a segmentation of the environment. `http://www.informatik.uni-freiburg.de/~stachnis/pdf/wurm08iros.pdf`, 2008.

[13] Agusti Solanas and Miguel Angel Garcia. Coordinated multi-robot exploration through unsupervised clustering of unknown space. In *International Conference on Intelligent Robots and Systems*, 2004.

[14] D. Puig, M.A. Garcia, and L. Wu. A new global optimization strategy for coordinated multi-robot exploration: Development and comparative evaluation. `http://www.elsevier.com/locate/robot`, 2011.

[15] Angus Johnson. Clipper - an open source freeware polygon clipping library. `http://www.angusj.com/delphi/clipper.php`, 2012.

[16] Bala R. Vatti. A generic solution to polygon clipping. *Communications of the ACM*, 35:56–63, 1992.

[17] Rogue Modron. Polygon clipping: a wrapper, a benchmark. `http://rogue-modron.blogspot.cz/2011/04/polygon-clipping-wrapper-benchmark.html`, 2011.

[18] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[19] Richard T. Vaughan. Massively multiple robot simulations in stage. *Swarm Intelligence*, 2(2-4):189–208, 2008.

[20] J. W. Durham and F. Bullo. Smooth nearness-diagram navigation. In *iros*, pages 690–695, Nice, France, 2008.

[21] Cyrill Stachniss. C implementation of the hungarian method. `http://www.informatik.uni-freiburg.de/~stachnis/misc/libhungarian-v0.1.2.tgz`, 2004.

[22] Mark H. Overmars and Emo Welzl. New methods for computing visibility graphs, 1988.

[23] J. Kitzinger. The visibility graph among polygonal obstacles: a comparison of algorithms. `www.cs.unm.edu/~moore/tr/03-05/Kitzingerthesis.pdf`, 2003.

[24] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, 1996.

[25] Alejandro Sarmiento, Rafael Murrieta-Cid, and Seth Hutchinson. An efficient motion strategy to compute expected-time locally optimal continuous search paths in known environments. *Advanced Robotics*, 23(12-13):1533–1560, 2009.

Appendix A

CD Content

The attached CD contains the thesis in PDF format, source codes of the thesis in LaTeX format and source codes of the framework. In table A.1 are listed names of all root directories and files on CD.

Directory name	Description
thesis.pdf	The Diploma Thesis in PDF format.
doc	The source codes of the thesis in LaTeX format.
eapd	The source codes of the framework.
lib	The external libraries.

Table A.1: CD Content