Technical University of Denmark

Christopher A. Ryther
Ole B. Madsen

Obstacle Detection and Avoidance for Mobile Robots

Bachelor's Thesis June 2009

DTU Electrical Engineering
Department of Electrical Engineering

Christopher A. Ryther
Ole B. Madsen

Obstacle Detection and Avoidance for Mobile Robots

Bachelor's Thesis June 2009

Obstacle Detection and Avoidance for Mobile Robots

Report written by
Christopher A. Ryther
Ole B. Madsen

Advisor(s)
Nils Axel Andersen (naa@elektro.dtu.dk)
Ole Ravn (or@elektro.dtu.dk)

DTU Elektro
Technical University of Denmark
DK-2800 Kgs. Lyngby
Denmark

E-mail: studieadministration@elektro.dtu.dk

Project period:	February - June, 2009
ECTS:	15
Education:	Bachelor of Science in Engineering (BSc)
Field:	Electro technology
Class:	1 (public)
Edition:	1st edition
Remarks:	This report is submitted as partial fulfillment of the requirements for graduation in the above education at the Technical University of Denmark.
Copyrights:	© C. Ryther & O. Madsen, 2009

Abstract

This report details the work performed over five months for a bachelor's thesis at the Technical University of Denmark (DTU). The project presents an obstacle avoidance plug-in module for a laser scanner application. A common wavefront algorithm is used for path planning and a fixed cell size grid map is used for the internal representation of the environment. The fundamental parts of the obstacle avoidance functionality are first tested in simulated environments, and ultimately the entire system is tested on a differential-drive robot mounted with a laser scanner, in three different scenarios. The report concludes that, under the given conditions, the plug-in proves to effectively complete the proposed test missions while avoiding obstacles.

Summary

Mobile robots are complex systems comprised of numerous sub-technologies, many of which still have a lot of potential for improvement. Navigation and mission planning, which are some of the most critical aspects of a mobile robot, are the primary subjects of this thesis.

Using a small differential-drive robot, mounted with a laser scanner, this project seeks to implement an obstacle avoidance system and validate its effectiveness through different test scenarios. For this purpose the two aspects of greatest importance are the internal representation of the surroundings and the search algorithm with which to plan a collision-free route.

The data received from the laser scanner is transformed and mapped into an internal fixed cell size grid map. From this map a modified wavefront algorithm is used to plan a safe path for the robot. Other functionality such as relocating the goal when it is unreachable, and resetting the map for long missions is implemented to increase the number of scenarios the plug-in can handle.

The obstacle avoidance system is tested on different platforms throughout the development. It is first partially simulated and tested in MatLab and then in C. Finally the whole program is implemented as a plug-in for the laser scanner in C++.

The entire system is tested as a plug-in on the mobile robot in three scenarios including a custom one-way maze and a long obstacle-filled hallway. Missions of up to 10 meters have been completed in this way.

In conclusion, the fundamental parts of the plug-in work perfectly under ideal conditions and can navigate a mobile robot through many types of environments. However because the plug-in is sensitive to robot odometry, and because the world can present many more scenarios than tested in the project, there is much potential for future work.

Contents

List of Figures

CHAPTER 1

Introduction

Mobile robots have until recently primarily been used in industrial production and for military operations. Nowadays however the technology is progressing towards so-called service robots, intended to assist humans in everyday life. These type of robots have been in development for years and will soon influence many aspects of our lives.

Autonomous navigation in obstacle filled environments has always presented a considerable challenge. Solving this problem would allow mobile robots to move about and complete tasks such as cleaning rooms, assisting disabled people and many other missions, without the risk of damaging themselves or their environment.

To navigate effectively the robot must be able to perceive and map the surrounding world accurately. It is also absolutely critical for mission success that the robot successfully identifies all obstacles around it. In this project the sensor used for these tasks is a 240° laser scanner mounted on a small differential-drive mobile robot.

1.1 Thesis Statement

The aim of the project is for an autonomous mobile robot to successfully navigate in a completely unknown indoor environment. Success criteria include, but are not limited to

- A forward velocity of at least 0.2 m/s.

- No practical restrictions on how far away the robot's goal can be located in relation to the starting point.
- Real-time stability.
- Plug-in functionality.
- No collisions with obstacles.

Using the data collected from the laser scanner and the robot's odometry, the project seeks to implement an internal grid-based map and use a wavefront path planning algorithm to succesfully avoid all obstacles.

The system in its entirety is to be written in C/C++ and used as a plug-in by the robot's internal laser scanner server, thereby eliminating most of the delay associated with wireless communication.

Also, all fundamental parts of the plug-in are to be designed in modules so that the plug-in can be used as an open library for future use.

1.2 Work by Others

Several of the points outlined in section 1.1 have been addressed by others, notably in (Galve, 2008). The work in this project is based on (Galve, 2008) — notably the theory behind coordinate transformations and map representation methods. However, Galve's project is implemented in MatLab, and as such is not particularly optimized for processing speed. This is evident from the slow movement of the robot during execution.

The actual control of the robot is interfaced by the MRC program, developed by Associate Professor Nils A. Andersen. The communication with the robot is handled by the ulmsserver application, developed by Assistant Professor Jens Christian Andersen.

Chapter 2

The SMR Platform

2.1 The Small Mobile Robot

For the practical implementation of the obstacle avoidance system in this project, the Small Mobile Robot (SMR) platform is used. These robots were developed by DTU, using mostly off-the-shelf components, and are highly customizable and versatile. For real time stability each robot runs a version of Linux installed with RTAI (RealTime Application Interface)[1].

Interfacing the SMR's hardware is done through the MRC and ulmsserver applications. MRC provides a way to control the movement of the wheels and measure the robot's pose in $(x; y; \theta)$-format, along with measurements of the various sensors, which are of less importance to this project.

To control the SMR movement, a language called SMR-CL developed by (Andersen and Ravn, 2004) is used. SMR-CL is an intuitive script language intended for the real-time control of the SMR without sacrificing capability.

A 240° laser scanner is mounted at the front of the robot. A server used to access the laser scanner, the ulmsserver application mentioned before, is run locally on the SMR. It provides measurements from the laser scanner in a XML-based format, each measurement also includes the odometry of the robot at the time the measurement was made.

[1] http://www.rtai.org/

2.2 Laser Scanner

The scanner is a Hokuyo URG-04LX and has a scanning angle of up to 240°. In short, the scanner is both a light source, emitting laser light into a rotating mirror, and a sensor. After leaving the mirror, the laser light is reflected off of objects around the scanner and returned.

The scanner has a maximum measuring range of up to 4 meters. It has an angular resolution of $\Delta\theta = 0.352°$, and a range accuracy of $r = 10\,\text{mm}$ within 1 m. The accuracy is however inversely proportional to the measurement distance, which means that from 1 m–4 m the accuracy is 1% (Hokuyo, July 2005). When a scan is made, it becomes available through the ulmsserver application. An example of a scan is seen in figure 2.1. Note that when a measurement has data where no reflection was measured, due to no obstacles being within maximum measuring range, that data is set to 4 meters.

Although the scanner has a field of view of 240°, only the middle 180° are considered. Using the extra 30° on both sides would in some cases give a slightly better view, but they are often blocked by the front wheels if the robot has been turning. Therefore the extra data is discarded, in order to avoid misleading measurements.

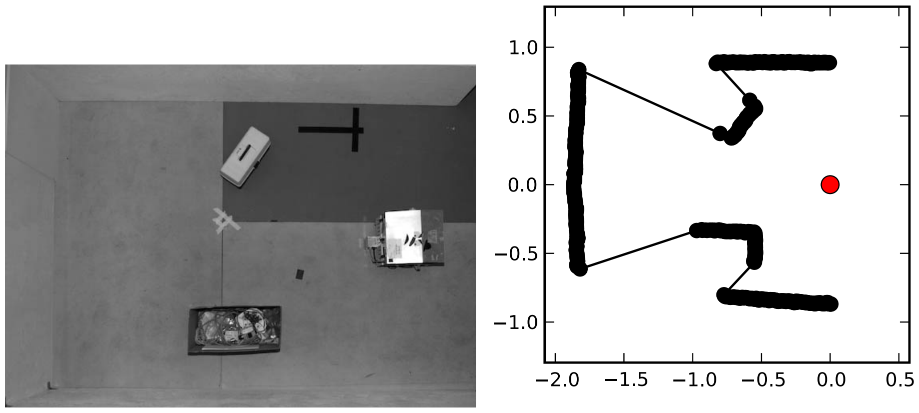

Figure 2.1: Laser scanner example. Left: Photograph of setup. Right: Plot of laserscan data in MatLab.

Chapter 3

Map Building

In order to adequately process the environment around the robot, an effective internal representation of the environment is needed. The success criteria for our representation are both a high accuracy and a low execution time. Since the robot at hand only moves in two dimensions, the entire process can be simplified to a representation of the two dimensional planar world around the robot. Because robot technology aims to minimize workload and hardware requirements, numerous solutions to this problem have been developed over the years (Siegwart and Nourbakhsh, 2004), two of which are discussed in this chapter.

3.1 Map Representations

By approximating objects in the real world with square cells in a large map, it is possible to find a balance between workload and accuracy so that execution time is low and the path planned still is safe. A first intuitive idea is to represent a large obstacle with large squares within its boundaries and then use smaller squares to approximate the rest, as seen in figure 3.1.

In programming languages however, it is quite difficult to represent this type of variable cell sized map. A fixed cell size map, on the other hand, can be implemented using a simple 2D matrix of numbers, where each number in the matrix corresponds to a cell of fixed resolution in the real world. With a low computational time, resulting in a high update frequency, and a laser scanner, this method of map representation can even correct errors

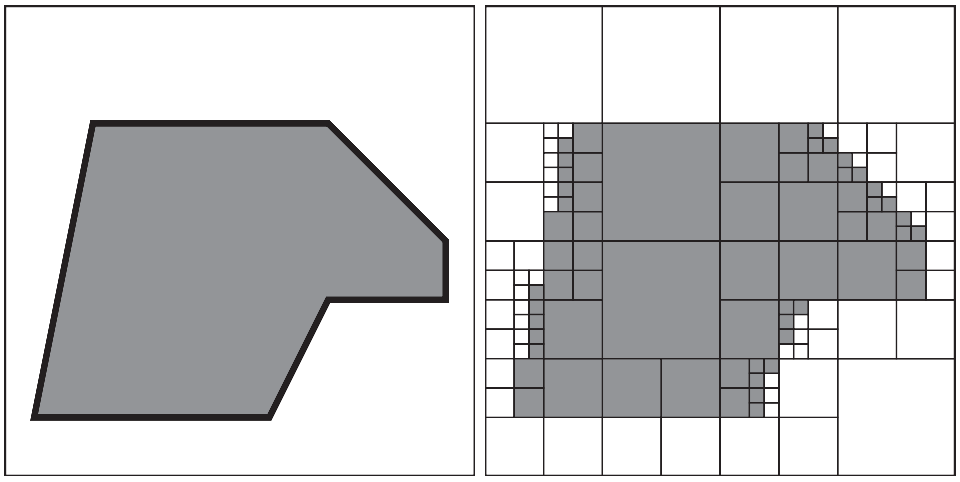
Figure 3.1: Left: Simulated obstacle as would be seen by laser scanner. Right: Varible cell size grid map representation of the obstacle.

made from previous scans. The downside is not being able to map the environment with more precision than given by the map resolution. It also increases memory usage, as the variable map in figure 3.1 has about 120 cells, while the fixed map in figure 3.2 has 1024 cells.

In this project a map resolution of 5 cm is chosen since anything smaller has been seen to result in large odometry errors (Galve, 2008). This means that the largest possible error in the internal map is 2.5 cm.

3.2 Fixed Cell Size Grid Map

Figure 3.2 shows a fixed grid map, where the black cells are objects that the laser scanner has detected, while grey cells indicates a clear area and white cells indicates an unknown area. In order to keep the internal map static with regard to orientation, there is a need for a transformation matrix from laser scanner coordinates to map coordinates (Galve, 2008).

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & O_{x_l} \\ \sin(\theta) & \cos(\theta) & O_{y_l} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_s \\ Y_s \\ 1 \end{bmatrix} = \begin{bmatrix} X_m \\ Y_m \\ 1 \end{bmatrix}. \qquad (3.1)$$

In equation 3.1 θ is the orientation of the scanner in the global coordinate system and $(O_{x_s}; O_{y_s})$ are the coordinates of the scanner's coordinate system in respect to the robot's local coordinate system. By using this transformation it is easy to convert between coordinates expressed in laser coordinates $(X_s; Y_s)$ and map coordinates $(X_m; Y_m)$.

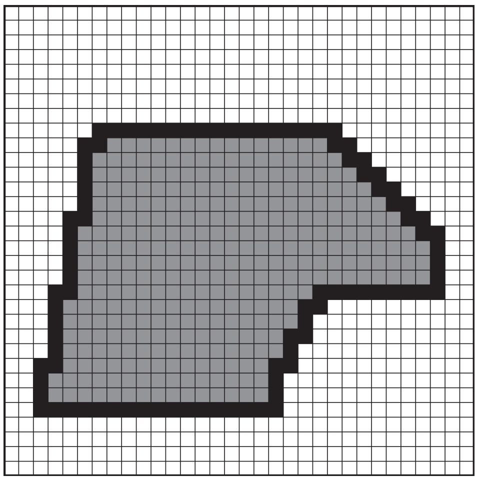

Figure 3.2: Fixed cell size grid map representation of the obstacle in figure 3.1.

3.3 Bresenham Line Algorithm

When a laser scan consisting of angle/distance pairs has been transformed into the correct (x, y) coordinates, the corresponding cells in the internal map are marked to represent obstacles. It is also necessary to mark as clear all the cells between the obstacles and the robot along the measurement path, so that the robot knows these cells are traversable. This is done with the Bresenham Line Algorithm (Bresenham, January, 1965). This algorithm determines which points need to be painted in order to achieve a good approximation to a line between two given points. However, given that the laser scans have a finite angular resolution, there is a tendency to be unpainted areas between each measurement. To avoid this, each point from the Bresenham algorithm is dilated into a 5-cell cross as seen in figure 3.3.

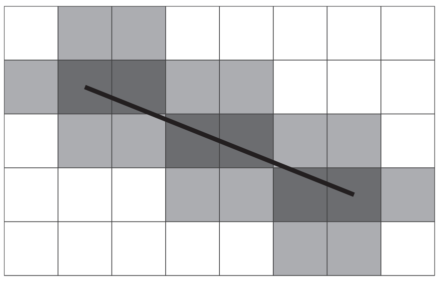

Figure 3.3: Example of two points and the bresenham line that connects them. The lighter grey cells are part of the line due to 5-cell dilation. All gray cells will be painted.

3.4 Map Dilation

When planning a path for a robot to follow, one would intuitively consider every single point containing the robot and assert that none of those points collide with the environment. However, keeping track of all the points at once and planning a route based on them is neither effective nor easy to do. An easy solution to this problem is representing the robot as a single point in the grid map. Safe route planning is then achieved using map dilation (Jazayeri et al., March 2006).

Map dilation is the process in which every single cell classified as part of an obstacle, is dilated with a certain amount so that the robot's size effectively is represented as a portion of the obstacles. This way the path planner need not consider the size of the robot in its calculations.

There are two methods of performing the dilation. The dilation can either be done by adding a square or a circle around each obstacle point. The first method seems intuitively correct since the robot itself is square, but this type of dilation is inaccurate since the robot's orientation cannot be forseen near a given point. Performing the dilation with a circle on the other hand, eliminates all orientation considerations.

There are other advantages to this general approach as well. The most noticeable is the averaging of measurement points. Since every single point is dilated with a relatively large area, the accuracy of the laser scanner becomes less of an issue, because most of the points' dilation overlap. This is seen in figure 3.4, where the distant measurements to the right still produce an even dilated area.

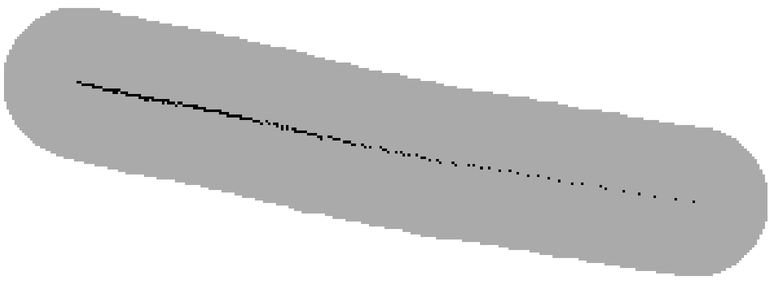

Figure 3.4: Scattered meassuring points from a laser scan (black) and the surrounding dilation cells (grey).

3.5 Map Reset

Because the internal map has a finite size and the universe does not, a problem arises whenever the position of the goal is outside the boundaries of the internal map. There are a few solutions to this problem: Either create multiple connected maps, dynamically expand the internal map or reset the internal map during the mission. With the two first solutions the robot would need to allocate relatively large amounts of memory during execution and that would be against real-time guidelines. Instead, the last option is chosen. Every time the robot is within 0.5 m of any of the edges of the internal map, the map is reset and the robot is placed in the middle of the new map. This ensures that the robot is never out of bounds and that the current provisional goal can be placed on the current internal map. In this manner, the robot's travelling distance is not limited by programming.

Chapter 4

Path Planning

4.1 Types of Path Planning

There are several ways to plan a movement path from a fixed-cell grid map, but among the simplest algorithms are the wavefront planner and the A* search algorithm (Galve, 2008). A* is computationally faster, but has a more complex implementation. For the sake of this project, the simpler wavefront planner is used. This can be justified by the fact that the searches can be executed extremely fast in C. While execution time was a major factor in Galve's selection of A*, this project is more focused on the implementation as a plug-in, and can later be expanded with a more thorough or faster path planning algorithm.

The path planning is implemented as two seperate functions, one that maps the distance associated with each cell, and one that calculates the actual path from this map. This modular structure allows for easy modification or replacement of the two algorithms independently of each other, but the primary advantage is in code maintainability.

4.2 Goal Location

Before the actual wavefront expansion can begin, as described in section 4.3 it is necessary to consider the location of the goal. At the start of a run, and every time the map is reset, the user defined goal (from here on referred to as the absolute goal) is examined. If it is inside the the borders of the current grid map, a provisional goal is placed at the exact same coordinates, and the planning begins. If however the absolute goal is outside the map, the provisional goal is placed at the edge of the map, as close as possible to the absolute goal.

Until the next reset of the map occurs, the absolute goal is only considered when evaluating if the robot has reached the goal or not. For the rest of the planning, the provisional goal is used. This ensures that the goal used in the planning process is always located on the map.

The first location of the goal after a map reset, is calculated regardless of whether the goal is in an unknown or cleared area, as the entire map is unknown at this point.

If however this provisional goal is located in a cell that is marked as an obstacle on the dilated map, it will be necessary to relocate the provisional goal. If the absolute goal is located outside the map, a primary concern is that this new provisional goal is located at the border, so the map can be reset properly, as described in section 3.5.

Regardless of whether the absolute goal is outside the map, the new provisional goal should also be as close to the previous provisional goal as possible. It should also be placed in a cell that is known to be cleared, for two reasons: To make sure that the new goal can be reached physically, and to hopefully prevent the necessesity of relocating the goal again, thus saving some computation time.

If the new provisional goal is allowed to be located in any cell that is not obstructed, there is a considerable risk that this new goal is located on the far side of the obstacle, resulting in a route that is impossible to reach. Forcing it to be in a cleared area ensures that the SMR has had this point in its line of sight earlier.

If the new goal is placed in an unknown area, there is a risk of it needing to be relocated again, namely if the SMR discovers that this new area is also obstructed. This could go on for several iterations of the run and result in a higher execution time.

Note that it is not a problem that the first provisional goal is in an unknown area, as the wavefront expansion happens in both known and unknown areas.

4.3 The Wavefront Planner

The wavefront planner (also known as grassfire) is a simple and effective way to plan the shortest possible path on a grid map (Andersen, 2008). By marking every cell's distance to the goal, the shortest path between the goal and the robot can be found.

The algorithm is simple to implement. It starts at the goal cell and marks each adjacent cell with the distance to the current goal. Using 8-point connectivity, each cell has up to eight adjacent cells, with the diagonal cells having a distance of $\sqrt{2} \approx 1.4$ to the curent cell. The remaining cells each have a distance of 1. This process is then repeated for each cell, continously marking neighbouring cells, until the robot position has been reached. Cells defined as obstacles[1] after dilation are ignored.

The algorithm has been implemented in such a way that it avoids any cell that is marked as an obstacle in the dilated grid map. It is free to plan in all remaining cells, known and unknown alike. This makes it easier to plan the route, as it is no longer necessary to place the provisional goal at the edge of known areas for every new plan.

If the wavefront expansion runs out of neighbouring cells before reaching the robot, it means that the marked goal and the robot are completely seperated by obstacles. However, there is another possibility: That the robot is in a dilated area. This can happen if the robot drives too close to a dilated obstacle or if a newly discovered obstacle is dilated into the robot's position. To remedy this situation, the immediate area around the robot[2] will be cleared in the dilated grid map for the purpose of the current path planning process. This clearing will however not permanently influence the internal map. A demonstration of this is shown in figure 4.1.

While this circle may seem like a rather large area to clear, it serves several purposes. Only clearing a rectangle with size and pose matching the robot, would ensure that a true obstacle never gets cleared. This is not necessary since the current radius is not large enough to clear both sides of an obstacle such as a wall. Furthermore the circular clearing results in a route that always will be planned away from the obstacle, into the cleared area. Using a larger circle helps ensure that this path is smooth and that the robot will have a way out, in the forward direction, instead of back the way it came.

However, an unwanted situation can arise if the goal and the robot are near the same obstacle. If the area around the robot were to be indiscriminately cleared, it is quite possible that a direct line between the two would be seen, with the unfortunate result that the planned path would have robot drive towards the goal, without regard for the obstacles. Therefore, the area is not cleared if the robot is too close to the goal.

There is an obscure situation, where the goal and the robot are close to

[1] Or outside the map entirely.
[2] A circle with a diameter of 55 cm.

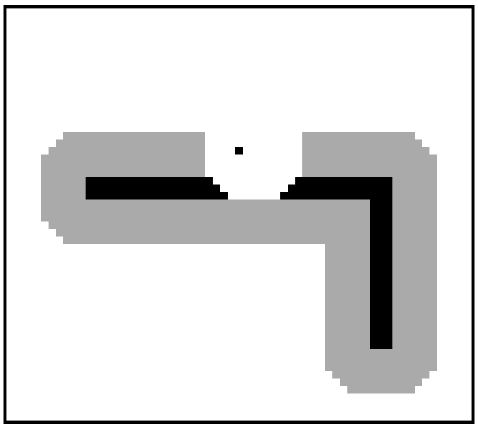

Figure 4.1: An example of how the area is cleared, where the robot actually was inside the dilated area. The black dot marks the position of the wheels, while the actual clearing is done from the physical center of the robot. Fortunately, there are no circumstances where the path planner will have the robot move closer to the dilated obstacle.

each other[3], with a wall between them and with the robot in the dilated area. In this case, the area around the robot will not be cleared, and then no path can be found to the robot. This will make the program stop, even though there may be a way to get closer to the goal. This scenario is not one that has been experienced in this project, and therefore it has not been dealt with in the program. A possible solution is to find the provisional goal before clearing the area.

4.4 Planning The Route

After a distance map has been made, it is possible to find the shortest route. Starting at the robot, every adjacent cell is examined, and the cell with the lowest distance value will be a part of the correct path. Following this algorithm until the goal is reached will result in a list of cells, given in map coordinates, each being a part of the final route.

However, if the next cell — in the path from the robot to the goal — takes the robot into an unknown area, there could be an undiscovered obstacle blocking the path. In order to avoid potentionally hazardous situations, the movement stops when such a situation arises, forcing an update of the environment when the robot has reached the edge of the unknown area.

[3]Within 27.5 cm — the clearing radius.

Every time the internal map is reset[4] the robot has to perform an initial scan of the surroundings. In this first scan it cannot see any potential obstacles behind itself. Because the map is reset, the path planner is unaware that the robot may already have travelled there, or that a path behind the robot might be blocked. To solve this problem the reset map function can insert a hollow circular artificial obstacle, 1 m in diameter, around the robot in the internal grid map.

Upon conducting its first scan the mapping function will clear the artificial obstacle where it is confirmed to not exist. As long as there are no real obstacles co-located with the artificial one, the first scan would thus clear the artificial obstacle in the entire 180° scan area while keeping the half-circle behind the robot intact.

This way the path planner is forced to believe that the path behind itself is blocked. This is particularly useful in large one-way maze setups where the robot should not backtrack. An example can be seen in figure 4.2.

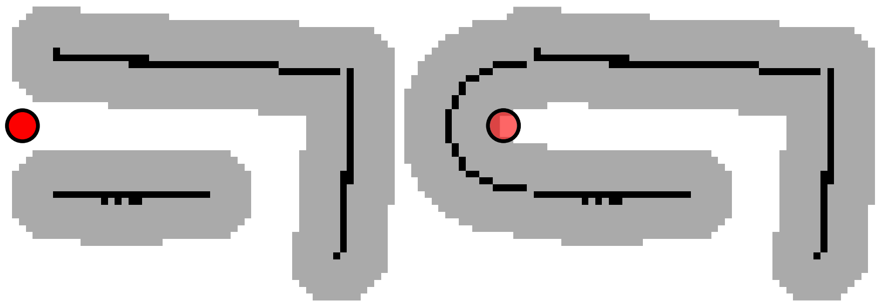

Figure 4.2: Left: A scan of the beginning of a maze as it would be if no artificial circle were added. The path planner would perceive an optional path behind the robot. Right: The same maze but this time a circle has been placed around the robot first, effectively blocking the area behind it. The path planner perceives a blocked route behind the robot.

4.5 Line Finder

Sending a long list of cell coordinates that the SMR should travel, will result in bad performance for two reasons. First of all, with the chosen communication method, it is only possible to send at most two coordinate pairs at a time. As a path can easily contain several dozen points, this will result in a lot of overhead. The second reason is that the SMR drives more smoothly if it receives a coordinate located farther away from the

[4]Which is at the beginning of every new mission and every time the robot is within 0.5 meters of the edges of the internal map.

robot, thereby giving the internal regulator more time to adjust. This was especially evident when the robot and grid map were tested with a cell size less than 5 cm. This made the internal regulator in the SMR unstable, and made it impossible to control the robot.

To alleviate this, a method to “compress” all the points on a single line was implemented. Taking the difference between the path points results in a list of small increments, of up to one cell width in each direction. This makes it easy to find any line segments along the path, which can then be added together, resulting in a node at the end of every segment.

Consider a goal position lying 2 meters in front of the robot, with an internal map resolution of 0.05 m/cell, resulting in 40 points. As all the points lie on the same line, the path can be compressed into a single $(40; 0)$ coordinate. The same idea can be used on a more varied path, as seen in table 4.1.

Path coordinates	Difference	New path
$(38; 43)$		
$(38; 44)$	$(0; 1)$	
$(38; 45)$	$(0; 1)$	
$(38; 46)$	$(0; 1)$	$(0; 3)$
$(39; 47)$	$(1; 1)$	
$(40; 48)$	$(1; 1)$	$(2; 2)$

Table 4.1: Example of how the line finder compresses six coordinate pairs into two. $(38; 43)$ is the robot’s current position.

As the robot only performs a new mapping of the surroundings at every node, it was decided to limit the lines to a maximum length of 2 m.

Note that when using relative coordinates, it is necessary to also communicate the pose from which the path was planned, as the internal odometry may have changed in the meantime. Not doing this would result in an offset error, unless the robot is brought to a complete stop before planning every path.

Chapter 5

Program Structure

5.1 Program Structure

The most common architecture of mobile robots divides mission control into several phases. Traditionally these phases are perception, planning and execution, as seen in figure 5.1, (Siegwart and Nourbakhsh, 2004). In the perception phase, the robot gathers data about the environment with its sensors and stores it. Based on the data collected, as well as any previous data, the robot will make decisions in the planning phase. In the execution phase the robot will execute these plans and complete its mission in the end.

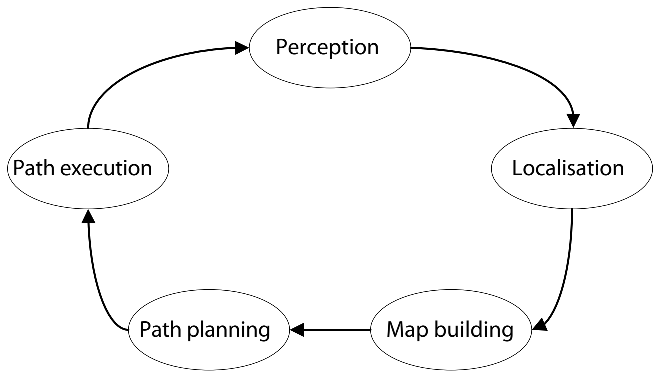

Figure 5.1: General program architecture used for control of the SMR.

The object of this project is the development of the path planning plug-in module, named *Laseravoid*, for the laser scanner server and an SMR-CL script for MRC. The laser server manages all laser scans and all plug-ins for the robot. The path planning plug-in uses a large open source library associated with the laser server, and only communicates with MRC through a set of XML variables.

A brief flowchart of how the plug-in interacts and performs with the SMR-CL script is seen in figure 5.3.

The plug-in is a passive element and is controlled entirely by the SMR-CL script. All perception, localisation, map building and path planning is handled by the plug-in.

As seen by the temporal diagram in figure 5.2, the different parts of the navigation system run at different speeds. To keep the execution time as low as possible the plug-in has an initialisation functionality that pre-allocates all significant memory needed to run. In this way additional delays are avoided during the runs. Even the types of data structures have been chosen to maximise performance.

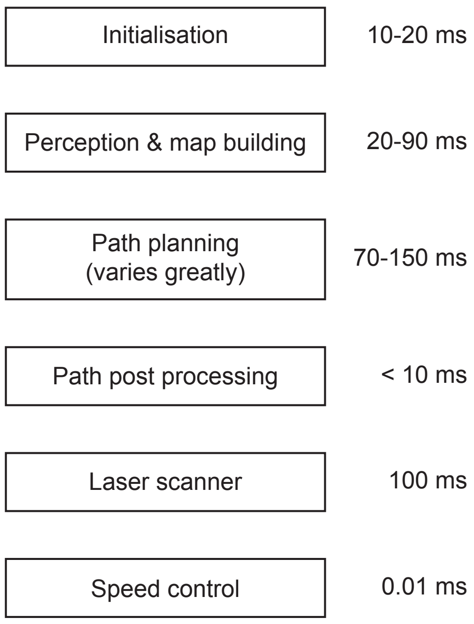

Figure 5.2: Temporal diagram.

By implementing the plug-in in C instead of MatLab, it is possible to run the program directly on the robot hardware instead of having to communicate wirelessly with a host computer running MatLab, as seen in other projects like (Galve, 2008).

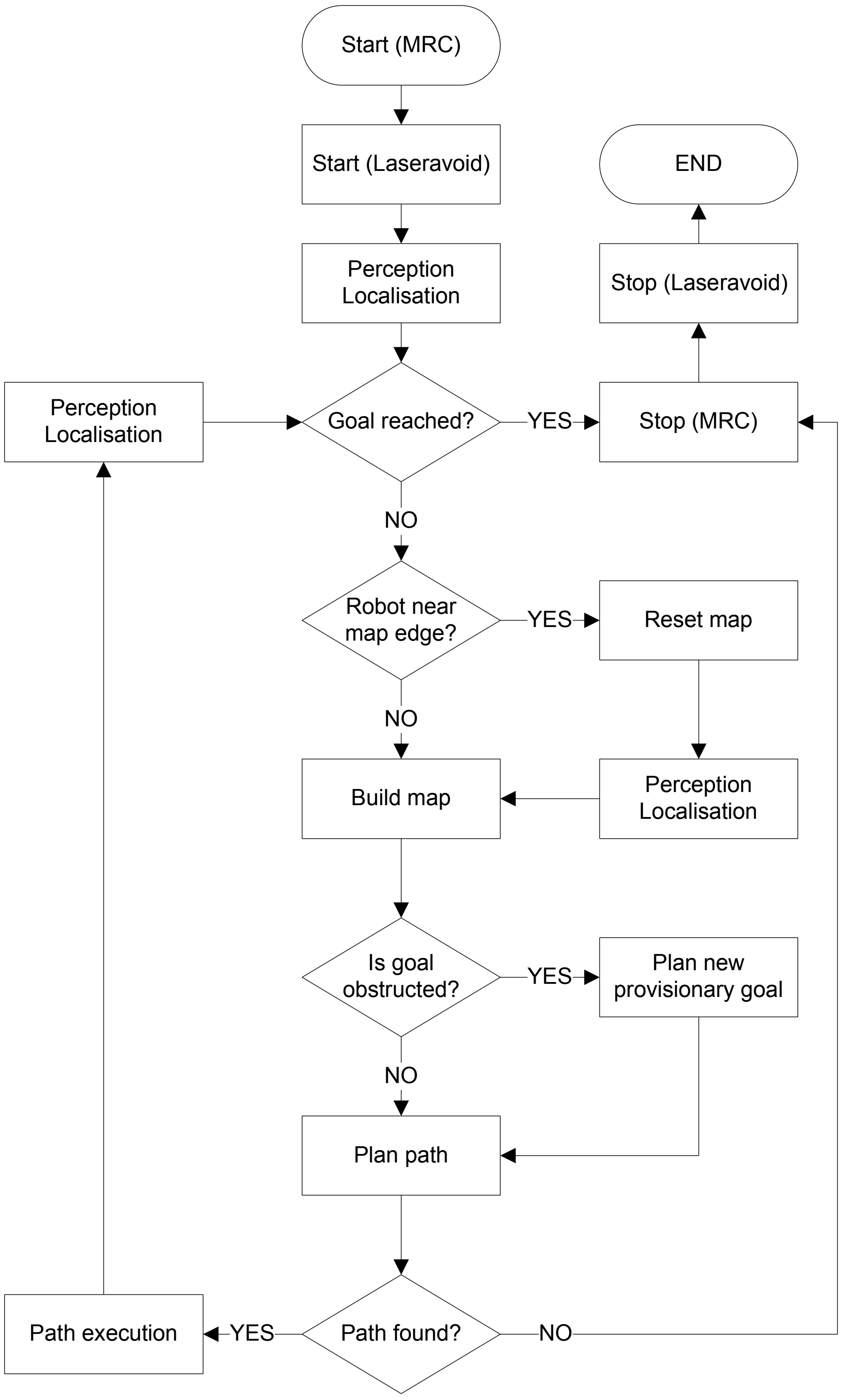

Figure 5.3: Flowchart diagram of the obstacle avoidance plug-in working together with the SMR-CL script.

5.2 SMRCL Drive

As it is not possible to control the movement of the robot from ulmsserver, it is necessary to have an SMR-CL script that handles all movement, and tells the server when to perform path planning, when to scan, and so on.

This will necessarily involve two separate applications, and therefore some issues with synchronisation of data exchange can arise. In order to prevent these issues, a counter flag has been added. When a command with data output is executed in the ulmsserver plug-in, the counter is incremented by one. The script receiving the data then performs a check, and delays further execution until it has verified that the counter has been incremented. This increment implies that new data has been received.

When the data is received and verified, path execution can begin. When the robot arrives a node, a new laser scan is made[1] in order to get a more accurate map of the surroundings.

The robot's move command is implemented with a controller described briefly in (Andersen and Ravn, 2004). Given a line defined by a point $(x; y)$ and a direction θ, the controller will move the robot along the line until the robot has reached a line through $(x; y)$ perpendicurlar to θ.

The script used for the development of the plugin can be seen in appendix C, and a further usage guide can be seen in appendix B.

[1] And mapped to the gridmap, but not dilated.

Chapter 6

Results

In the following chapter all the implemented functionality is tested in both simulated environments and in real scenarios with the SMR. The first two simulations demonstrate that the two fundamental parts of the obstacle avoidance plug-in work when given simulated data. Then, the two same parts are tested given real data from the laser scanner in a simple setup. The three next tests are example scenarios that serve to describe the strengths and weaknesses of the chosen solutions. Finally a series of repeated runs are performed and the results are compared.

All runs are performed with the desired forward velocity of 0.2 m/s.

6.1 Simulations

6.1.1 Testing the Grid Map and Dilated Map

In order to test that the dilation function works properly, a test case can be made where various obstacles are placed on the grid map, and then dilated. First of all this will prove that the implementation works with default parameters, and that all sides of an obstructed point are dilated by an equal (and adequate) amount. Figure 6.1 shows an example of this.

Along the vertical parts of the border, the dilated area extends by 6 cells to either side. With the default resolution of 0.05 m/cell, this is equivalent to 30 cm. This is as expected, as it is slightly larger than the required 27.5 cm. The difference is due to rounding, but neglible. The same tendency can be

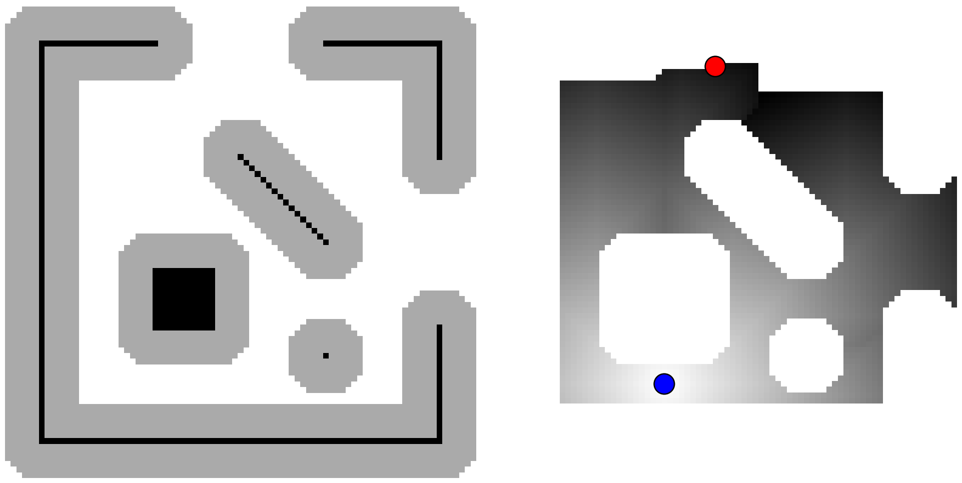

Figure 6.1: Simulated environment mapped into a grid map with a dilated map as an overlay. The black cells are the actual obstacles, the grey cells show the dilated areas and the white cells show the known cleared area. The figure to the right show the resulting wavefront expansion. Robot position is the red marker and goal is the blue marker. The darker the cells are, the farther away from the goal position they are.

seen at the horizontal parts of the border, and along the edges of the large box.

Since circular dilation is used, the corners of all dilated points will be rounded. With these examples the dilation only results in a small (and aliased) rounding of the corners, due to the low resolution used with the SMR. As expected, the farthest dilated point is at least 6.4 cells away away from an obstacle's corner.

The openings in the borders demonstrate that line endings are handled properly, meaning the dilation radius is correct. Finally, the large box shows that the dilation algorithm can handle neighbouring obstructed cells properly, without overwriting existing obstacles.

6.1.2 Testing the Wavefront Planner

The second simulation is to show that the wavefront algorithm works as expected. By taking the dilated map from the previous simulation and performing the wavefront expansion, the resulting wave map is as seen to the right in figure 6.1.

As expected, the wavefront algorithm starts at the goal position, avoids all dilated areas, and succesfully reaches the robot's position.

These results prove that the two most fundamental parts of the laser-based object avoidance system function under ideal conditions.

6.2 Practical Tests

6.2.1 Laser Scan to Grid Map

Figure 6.2 shows the relationship between raw measurement data from the laser scanner, and the internal grid map generated by the program. The robot was at $(x; y; \theta) = (0; 0; 0)$ as indicated on the raw plot on the left.

As can be seen, the algorithm plots the data fairly accurately, although measurement noise from the laser scanner and rounding errors from the grid map transformation detract a bit of the "neatness". Some of these artifacts stem from the cross-painting done by the map updater as described in section 3.3.

When measuring the number of cells from the edge of the visible area there are 16 cells(0.8 m) to the middle obstacle, and 30 cells(1.5 m) to the back wall. Both these measurements match the distances reported by the laser scanner, as seen on the axes of the MatLab plot. The entire grid map is 61 cells from top to bottom, this is equal to 3.05 m and also matches the scan.

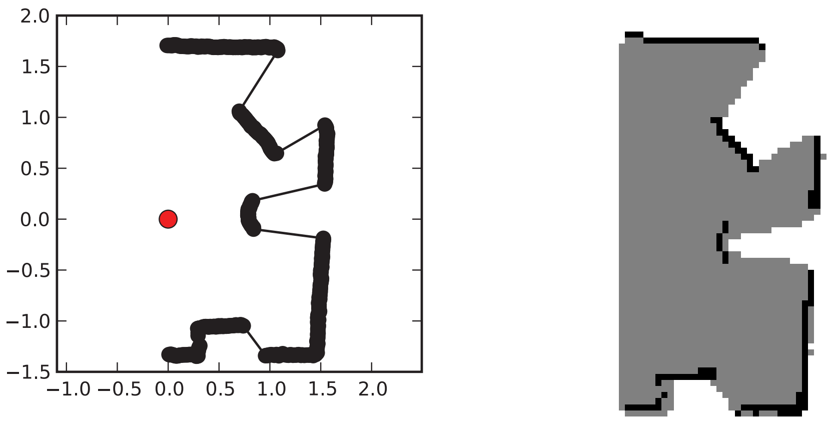

Figure 6.2: Left: A polar plot of a laserscan from the robot. Right: The same laserscan data transformed and mapped in a grid map. Note that in this case, the grey areas represent cells that are known to be clear, while white cells represent the unknown area.

6.2.2 Plan Path from Dilated Map

To ensure that the path planning algorithms also work on real data and not just on simulated walls, a scan was made and dilated, and is shown in figure 6.3. As expected, the obstacles are dilated normally. Even though the robot is within line of sight of the goal point, the dilated obstacles result in a necessary “slalom”. The planned path has also been added to this map, and it shows that the path planning algorithm prefers to make turns as soon as possible, instead of driving forward.

On the right half of the figure, a view of the corresponding wavefront map is shown. This case is a somewhat inefficient one, where the algorithm explores the area above and below the obstacles before finally reaching the robot’s position. This behaviour is an artifact of the order in which the neighbouring points are visited.

As with the simulated results, the bright spot is the goal position, and the path line matches the shortest route possible, as reported by the wavefront map. In reality, there are several other possible equidistant routes, such as having the robot drive forward before turning.

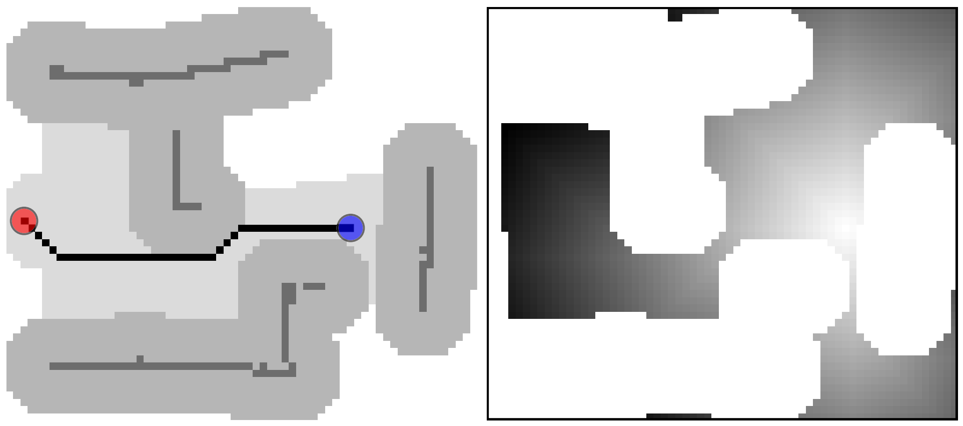
Figure 6.3: Left: A dilated gridmap with path Right: A wavefront expansion performed on the same map.

6.2.3 Scenario: Simple Avoidance with a Few Boxes

The first real test of the object avoidance is done in the simplest of all possible relevant scenarios. The robot has to navigate diagonally across an enclosed square area populated by 3 objects of various shapes and sizes, as seen on figure 6.4.

The area’s dimensions are roughly 2 by 2 meters and it has underlying

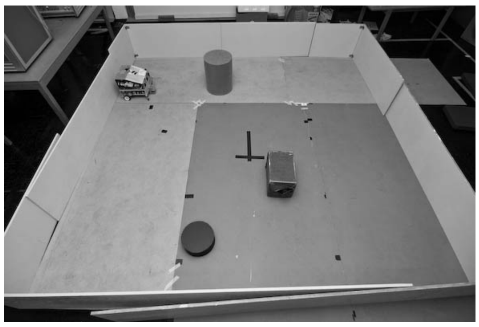

Figure 6.4: Photograph of the simple avoidance setup. For the first test the robot started in the opposite corner of where it is on the above photograph.

sheets of hardboard to minimize the robot's wheel spin. After completing the mission, the internal grid map, dilated map as well as the actual followed path were drawn in figure 6.5.

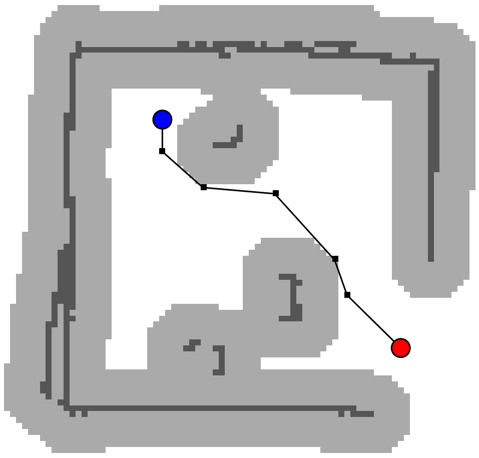

Figure 6.5: Grid map with dilated map as an overlay together with the SMR's followed path. The dots represent nodes where laser scans were made.

As expected the robot avoids all obstacles while planning the shortest route possible. Unfortunately round obstacles with a relatively small diameter have a tendency to be mapped with lower accuracy due to the size of the cells.

Also, straight lines parallel to either the internal x- or y-axis will always be mapped more precisely due to the square shaped cells in the map representation. This is best seen in figure 6.6 where the same course is run with the robot's initial orientation not being orthogonal to any of the course's sides.

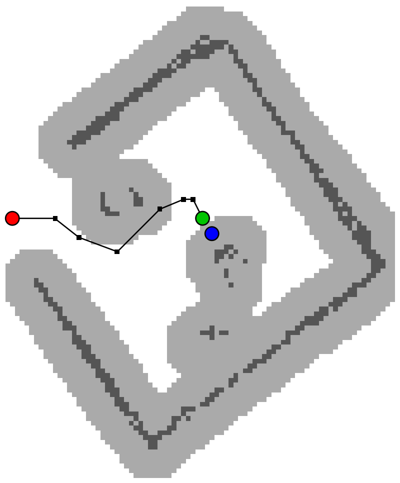

Figure 6.6: Grid map and dilated map of setup with non-orthogonal orientation relative to surrounding walls.

Another important difference between the first run and second run is that the second time, the goal is intentionally placed inside the dilation radius of an obstacle. As anticipated, the wavefront planner acknowledges the problem and changes the goal to a reachable point as close to the original goal as possible, as seen in figure 6.6.

6.2.4 Scenario: Maze

The second test performed is done in a classical one-directional maze as seen in figure 6.7.

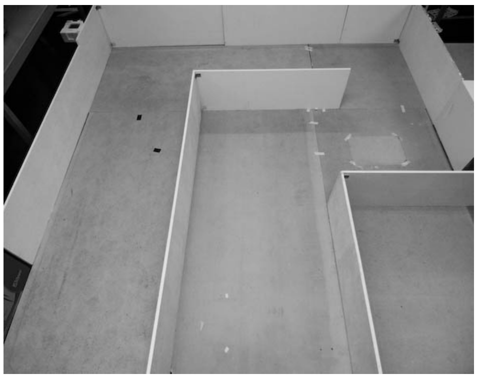

Figure 6.7: Photograph of maze setup. The beginning is in the bottom left-hand corner and the finish is in the lower middle part.

To reach its goal, the robot has to plan a path that initially leads it away from the end of the maze, but as expected it still reaches the goal in the end. The final grid map with the dilated map as an overlay, is shown in figure 6.8.

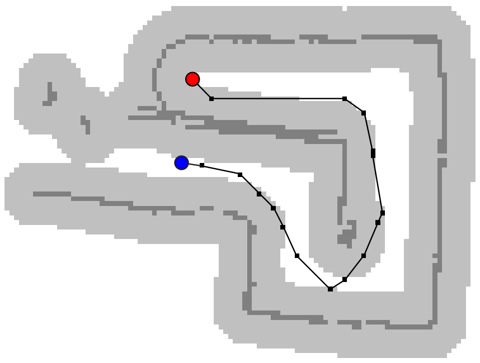

Figure 6.8: Combined grid map and dilated map from a test run through maze.

As long as the internal grid map is accurate as above, it is possible for the robot to navigate narrower mazes, as seen by the ample amount of cleared space on the map. However due to the way the robot movement controller works[1], it is hard to implement 180° turns from a path planning point of view. This means that although it is possible to plan a route if a dead end is found, it requires a lot of space to turn around.

Unfortunately in setups such as this maze, where there can be many small turns, the internal map becomes very sensitive to odometry inaccuracies. As seen on the grid map from the test run, the further along the maze the robot is, the more crooked the walls are. In this case it is due to accumulated odometry errors, primarily associated with small turns, possibly because they are not completed entirely. Errors such as these can result in the robot driving into walls when trying to navigate as close to them as possible.

In conclusion the robot confirmed its ability to perform independently and proved that the robot would be able to navigate any kind of one-directional maze without any prior knowledge of its layout.

6.2.5 Scenario: Hallway

The third and final test is performed in the institute's hallway, cluttered with obstacles as seen in figure 6.9.

In this setup the robot is planned to move 10 meters in the hallway, and

[1]As explained in section 5.2.

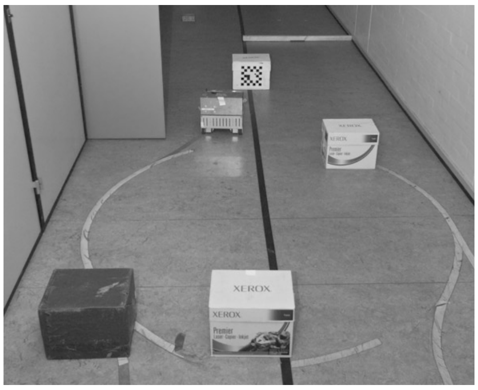

Figure 6.9: Photograph of hallway setup seen from beginning position during a run.

given the dimensions of the internal map, this means that the map will have to reset itself at least three times. This course is therefore a good indicator of how the robot handles long distance missions.

As with the maze example, the robot's performance is sensitive to initial orientation as well as odometry errors. Even more so because the goal is given in absolute coordinates relative to the start position. However given relatively accurate odometry, the robot completed the course and avoided all obstacles on the way. The three internal maps are shown in figure 6.10.

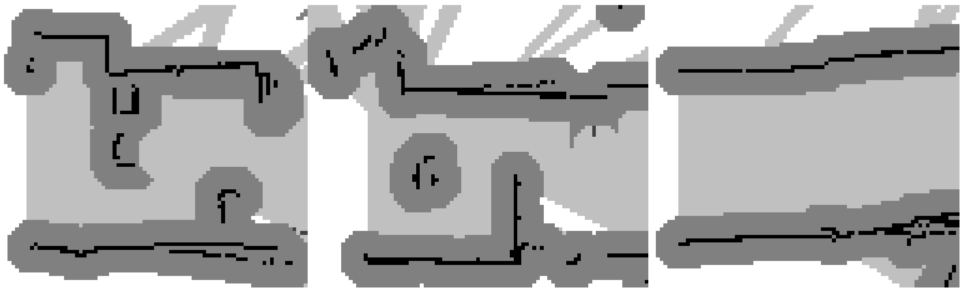

Figure 6.10: Three generated internal maps from driving down a hallway. Note how on the first two maps the clearing of dilated area around the robot is visible as round indentations near some obstacles. Above the upper wall several misreadings are visible as cleared areas behind the wall.

The maps clearly illustrate the ability to automatically reset the internal map when needed while retaining track of both odometry and the position of the goal.

The walls of the hallway are unfortunately built in such a way that the laserscanner gets misleading reads more often than with other types of walls. There are false negativs, i.e. “missed targets”, as well as false positives, “false targets”, on all three maps. Together with the accumulation of odometry errors, this can lead to a situation where the goal internally is located in a wall. When this happens the natural reaction is to relocate the goal to the nearest known point to the original goal, as described in section 4.2. A problem arises when the laser scanner has made an errornous reading through a wall, leading the path planner to believing that it can “see” behind a wall. In this case the planner will place the new goal behind the wall. As there is no reliable way to differentiate between misreadings and actual readings, this very specific problem remains unsolved.

6.2.6 Repeated Runs

To examine how robust the obstacle avoidance system is, a series of runs are made on the two last scenarios above. The results are shown in table 6.1.

Scenario	Total Runs	Completed Runs	Failed Runs	% Completed
Maze	45	36	9	80%
Hallway	25	21	4	80%

Table 6.1: Results of repeated runs in different scenarios.

A few more details on the runs can be found in appendix D. Successful completion is in both cases defined as the robot reaching the set goal. Given this criterion the maze setup proved to be the hardest because both scenarios had an average success rate of 80%, but in the maze there were significantly more brief grazings.

During the runs in the maze the robot showed a tendency to navigate too close to the walls and would some times briefly graze a wall while turning. However, when the robot was intially placed at an angle, relative to the maze’s walls, the robot ran the course 10 times without once grazing the walls. This can be due to the walls being approximated internally by jagged lines. The wavefront planner will avoid some of the “indents” in the walls and thereby create an additional small margin between the robot and the walls. This indicates a general problem regarding the obstacles’ dilation radius not being large enough.

As mentioned earlier the robot was very sensitive to its inital pose in the hallway example. Placing a goal 10 meters ahead of the robot meant that pointing the robot in the right direction was crucial to its success. Two of the failed attempts at navigating the hallway were due to misreadings of the wall, as mentioned earlier. Two more were due to motor errors where the robot did not turn much as expected (or not at all). Taking those factors into consideration, the plug-in is capable of navigating the hallway with a high success rate.

In both scenarios expanding the dilation radius seems like a plausible solution to some of the problems encountered. Expanding the dilation radius would however create further requirements for how large obstacle-free areas the robot would need to be able to navigate. For this project the smallest theoretical necessary dilation radius has been chosen to enable path planning in narrow spaces. An alternative solution, implementing a gradient map, is discussed in chapter 7.1.

Chapter 7

Conclusion

In this thesis an object avoidance system designed for, and implemented on, a small differential robot was presented. For accurate perception of the environment, the robot was mounted with a 240° laser scanner.

A fixed cell size grid map was used to represent the environment as accurately as possible without sacrificing computational performance. Using a wavefront algorithm, together with intuitive goal tracking and relocation, enabled the path planner to successfully generate collision free routes. Both modules were successfully tested individually in simulated environments.

By choosing appropriate data types in the final implementation and refraining from allocating memory during missions, the program ended up following the real time stability guidelines as required.

The two modules were partially tested together on different platforms, and finally tested as a plug-in for the laser scanner in C++. Given three test scenarios, a simple obstacle filled room, a maze and a long obstacle filled hallway, the capabilities of the implemented solutions were demonstrated.

The obstacle filled room proved the basic functionality of the obstacle avoidance, by showing that the robot could navigate an open area without collisions. In the maze the system proved capable of navigating through a completely unknown course and finding the goal. Finally the hallway tested how well the robot handled courses where the internal map needed to be reset in order to cover long distances.

The robot could complete all three scenarios with the required forward velocity, and proved that the obstacle avoidance is not limited by goal location or length of the course.

With a successrate of 80% for both the maze and the hallway scenario the plug-in proved to work with a relatively high success rate. There is however room for improvement, especially in regards to which routes to prioritize when navigating.

The laser based avoidance was however found to be very sensitive to odometry inaccuracies and performed much worse when navigating an uncalibrated mobile robot.

The successful use of the plug-in by the small mobile robot has shown the usabilty and versatility of the implemented system. That being said the subject of navigation and obstacle avoidance clearly remains material for research in the next many years.

7.1 Future Work

In this project a number of source blocks have been designed in the area of navigation and obstacle avoidance. There are several ways these blocks could be used or improved. Some of these are discussed in the following section.

7.1.1 Implementation in Autonomous Tractor

Assistant Professor Jens Christian Andersen has expressed interest in using an obstacle avoidance system for use in an autonomous robot in an orchard. The robot navigates with the help of GPS localisation through rows of trees in the field. GPS loclisation contains no information about obstacles, which is why an obstacle avoidance system could assist the robot in moving properly, without colliding with objects. Refer to (Hansen et al., 2009) for relevant information.

7.1.2 Gradient Map

In order to achieve a more robust route through for example, a maze, the dilation functionality can be expanded. Instead of just using a binary mapping of obstacles and their dilation width, a gradient dilation could be added. With this type of map it would be possible to define zones that the robot absolutely must not drive into, and another type of area that the robot should avoid, e.g. close to the wall, but could move through without problems.

This approach would tend to make the robot drive centered between the maze's walls instead of trying to plan a path as close to the walls as possible. Driving farther from the walls would result in a route with less risk of accidentally driving into an obstacle. This is in contrast to the current path planner, which prefers to stay as close to a straight line towards the goal as possible.

7.1.3 Odometry Correction Using Pattern Recognition

A major problem in the hallway scenario is the fact that a relatively small drift in odometry quickly can result in a rotational error. This error can result in the path planner placing the original goal inside a wall in the internal map. Likewise, a navigation system that only relies on surrounding walls for navigation, may have problems avoiding random obstacles in its path. Combining the two, or creating a secondary system that makes use of landmarks to correct odometry, could improve the robustness of the navigation.

7.1.4 Obstacle Tracking

Implementing obstacle tracking would achieve a better representation of the environment by refining the internal map using previous scan data, instead of simply overwriting the old data. A more accurate and refined internal map could be utilised to achieve the following improvements:

- Accuracy of obstacle location.
- Elimination of misreadings from the laserscan.
- Provide data for odometry correction.
- Allow avoidance of moving obstacles.

Bibliography

Niels Axel Andersen and Ole Ravn. SMR-CL, a real-time control language for mobile robots. In *2004 CIGR International Conference*, 2004.

Nils Axel Andersen. DTU Course 31388 advanced autonomous robots, exercise 11, 2008.

Jack E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, Vol. 4, No.1:25–30, January, 1965.

Rafael Olmos Galve. Laser-based obstacle avoidance for mobile robots. Bachelor's thesis, Technical University of Denmark, May 2008.

Søren Hansen, Mogens Blanke, and Jens Christian Andersen. Autonomous tractor navigation in orchard - diagnosis and supervision for enhanced availability. In *Procedings of 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, 2009.

Hokuyo. Range-finder type laser scanner URG-04LX specifications. Technical report, July 2005.

A. Jazayeri, A. Fatehi, and H. Taghirad. Mobile robot navigation in an unknown environment. In *9th IEEE International Workshop on Advanced Motion Control, pages 295-299*, March 2006.

Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, 2004.

Appendix A

Command Reference for SMR-CL

Command	**laseravoid start**
Required parameters	*double* gx *double* gy
Optional parameters	*double* res = 0.05 *int* mapx = 8 *int* mapy = 8 *int* maze = 0
Return value	*void*
Description	Initialises arrays for maps and so on. gx and gy refer to the goal position, relative to the current position (not pose). $gx = 5$ and $gy = 0$ means that the goal is 5 meters in front of the robot, compared to the original reference grid. Map size is in meters, resolution is in meters per cell.

Command	**laseravoid stop**
Required parameters	*void*
Optional parameters	*void*
Return value	*void*
Description	Stops all calculations and frees up memory taken by maps. Used as a final command to clean up.

Command	**laseravoid updatemap**
Required parameters	*void*
Optional parameters	*void*
Return value	*void*
Description	Updates the internal map and odometry. Never resets the map, because it does not have to (a path will never be planned outside the map).

Command	**laseravoid planpath**
Required parameters	*void*
Optional parameters	*void*
Return value	*void*
Description	Plans a new path, and saves it in an internal queue.

Command	**laseravoid getnext**
Required parameters	*double* gx *double* gy
Optional parameters	*int* n = 2
Return values	l0 *int* atGoal / Counter — -1 if at goal, else loop counter l1 *int* Number of coordinates sent [0;2] l2-l4 *double* $(x; y; \theta)$ – The robot pose the route is planned from (l5-l6) *double* $(x1; y2)$ – The first set of coordinates to move to (l7-l8) *double* $(x2; y2)$ – The second set of coordinates to move to
Description	Returns up to two sets of coordinates $(x; y; \theta)$, along with a counter value, so the script can synchronise.

Appendix B

User Manual

This is a short section that describes the general intended usage of the plug-in as well as how to setup the SMR for runs. All code can be found in the CD attached to the report.

To work with the SMR the user has to start ulmsserver locally on the robot and load the shared object file “aulaseravoid.so.0”. To test single commands to the plug-in the user can write “laser *command*” in the ulmsserver window directly. Refer to apendix A for details on all the commands for the plug-in. In order for the MRC application to communicate with ulmsserver it is important that the server port is set to 24919.

To succesfully make the SMR use the plug-in during runs, the SMR-CL script should be built up like drive.smrcl on the CD. Like the ulmsserver application, the SMR-CL script should also be run locally on the SMR.

In general it is important to issue the start command to intialize the plug-in and the stop command to stop it – regardless of usage. Afterwards the updatemap should be used frequently to keep an up-to-date version of the robot’s surroundings in the internal map.

Appendix C

Example SMR-CL script

Save the script to a folder on the SMR, and run `mrc scriptname`.

```
% Specify the goal and run type
goalx = 2
goaly = 0
maze = 0

% Initialize variables
x=$odox
y=$odoy
th=$odoth
$l0=0
$l1=0
$l2=0
$l3=0
$l4=0
$l5=0
$l6=0
$l7=0
$l8=0
$l9=0

counter = 1
stepno = 0
array "dx" 5
array "dy" 5

% Send start command to plugin
stringcat "<laseravoid start gx='" goalx "' gy='" goaly "' maze='
    " maze "'/>"
laser "$string"
```

```

% Main loop starts here
label "loop"
laser "<laseravoid planpath/>"
laser "<laseravoid getnext/>"

% Recieve data
label "counter"
wait 0.05
if ($10 < -0.05) "finished" % If £10 (atGoal) is -1, we're done.
if (counter > $10) "counter" % As long as our counter is higher than the plugin's, wait some more for an update
    counter = $10 + 0.5 % Increase our counter. Add 0.5 to ensure that it is > £10.
    if ($11 < -0.05) "loop" % If coord count is negative, start over.

    % Load coordinates
    x=$12
    y=$13
    th=$14
    dx[1] = $15
    dy[1] = $16
    dx[2] = $17
    dy[2] = $18
    stepno = 0

    if ($11 < 0.005) "loop" % If number coordinates is 0 or negative, start over

    label "driveloop"
        call "move" % Drive to the next coordinates
        laser "<laseravoid updatemap/>" % Update the map with the latest scan
        wait 0.05
    if (stepno < $11) "driveloop" % Repeat the last few steps if there are more coordinates
    stop % Make sure the robot halts between calculations. This is hardly noticeable, and improves performance.
goto "loop"

label "move" % Function to actually move
    stepno = stepno + 1
    x = x + dx[stepno]
    y = y + dy[stepno]
    th = atan2(dy[stepno], dx[stepno])
    ignoreobstacles
    drive x y th "rad": ($targetdist < 0)
return

label "finished"
laser "<laseravoid stop/>" % Stop the plugin after a run
stop
```

Appendix D

Results from repeated runs

Type	Completed	Failed	Grazed Obstacles	Success Rate
Maze Normal Initial Pose	25	9	Avg. 1 per run	74.3%
Maze Angled Initial Pose	10	0	None	100%
Hallway	21	4	Two in total	80%

Table D.1: Results from repeated runs of the scenarios.

Run #	Completed	Notes
1	✓	
2	✓	
3	✘	Drove too close to wall after first turn and got stuck.
4	✓	
5	✓	
6	✓	
7	✓	
8	✓	
9	✓	
10	✘	Drove too close to wall after first turn and got stuck.
11	✓	
12	✓	
13	✓	
14	✘	Drove too close to last wall before last turn and got stuck.
15	✓	
16	✘	Drove too close to wall after first turn and got stuck.
17	✓	
18	✓	
19	✘	Drove too close to last wall before last turn and got stuck.
20	✘	Drove too close to last wall before last turn and got stuck.
21	✓	
22	✓	
23	✘	Drove too close to wall after first turn and got stuck.
24	✘	Drove too close to last wall before last turn and got stuck.
25	✓	
26	✓	
27	✓	
28	✓	
29	✓	
30	✓	
31	✓	
32	✘	Drove too close to wall after first turn and got stuck.
33	✓	
34	✓	
35	✓	

Table D.2: Results from repeated runs of the maze with initial pose of the robot being orthogonal to the walls.

Run #	Completed	Notes
1	✓	
2	✓	
3	✓	
4	✓	
5	✓	
6	✓	
7	✓	
8	✓	
9	✓	
10	✓	None of the runs grazed the walls or failed.

Table D.3: Results from repeated runs of the maze with initial pose of the robot being at an angle relative to the walls.

Run #	Completed	Notes
1	✓	
2	✓	
3	✓	
4	✓	
5	✓	
6	✓	
7	✓	
8	✗	Misreading through wall caused goal location failure.
9	✓	
10	✓	
11	✓	
12	✓	
13	✓	
14	✗	Incomplete turn resulted in collision with wall.
15	✓	
16	✓	
17	✓	Incomplete turn resulted in collision with wall.
18	✓	
19	✓	
20	✓	
21	✗	Misreading through wall caused goal location failure.
22	✓	
23	✓	
24	✓	
25	✗	Collision with obstacle.

Table D.4: Results from repeated runs of the hallway with initial pose of the robot being orthogonal to the walls.