

Robot Navigation and Mapping with Vision-Based Obstacle Avoidance

Erik Rufelt

July 20, 2009

Abstract

This report describes solutions to two important problems for mobile robot control, mapping of a static environment and avoidance of dynamic obstacles. There are many possibilities to achieve an appropriate behavior, and this project is based on ultrasound sonars and a webcam, present on the MobileRobots Pioneer P3DX robot used for the project. The operating environment is an indoors office-like environment, and the project is about realizing a working control software, allowing the robot to explore its environment, as well as avoid humans moving in the robot's vicinity while doing so. The end result is a map of the world, and the ability to view it through a web-interface on a wireless connection. The map must correspond to the real world with enough accuracy to be used for pathfinding and orientation.

Sammanfattning

Denna rapport beskriver lösningar på två viktiga problem vid användning av en mobil robot, kartläggning av en fast omgivning och undvikande av rörliga hinder. Det finns många möjligheter att uppnå ett nöjaktigt beteende och detta projekt baseras på ultraljud och en webbkamera, som finns installerade på den robot av typen MobileRobots Pioneer P3DX som används för projektet. Arbetsmiljön är en inomhus kontrosmiljö, och projektets huvudsakliga tema är att implementera en fungerande mjukvara, som låter roboten utforska sin omgivning, samt undvika människor som rör sig i robotens närhet. Slutresultatet är en karta av världen och möjligheten att visa den i en webbläsare, via trådlöst nätverk. Kartan måste stämma tillräckligt överrens med verkligheten för att kunna användas till att hitta vägar i den verkliga omgivningen.

Contents

1	Introduction	4
2	Background	6
2.1	Robot specifications	6
2.2	Ultrasound sonars	7
2.3	Map	8
2.4	Odometry and SLAM	10
2.5	Dynamic obstacles	11
3	Mapping	12
3.1	Sonar mapping from odometry orientation data	12
3.2	Pathfinding in the map	14
3.3	Odometry drift	14
3.4	Sonar problems	16
4	Dynamic obstacles	18
4.1	Sonar	18
4.2	Video	18
5	Experiments and results	21
5.1	Control program and interface	21
5.2	Robot movement	22
5.3	The robot handler	23
5.4	Map results	24
6	Conclusions and future work	26
	References	27
A	User's Guide	28

1 Introduction

Mobile robotics is an interesting field, with a multitude of possibilities, given a sufficiently intelligent control program. There are several problems that need solving to get a functioning robot, which, depending on the goals for the robot, can be more or less complex. The most basic method of movement for a mobile robot is to drive forward until an obstacle is detected, and then turn to avoid hitting the obstacle. To reach a more intelligent and useful behavior there are issues such as mapping and planning to solve.

There are many types of mobile robots, some of which appear in Figure 1. The usage possibilities are vast, already including tasks such as cleaning, fetching, scouting and mapping, among others [8]. Future possibilities include most tasks carried out by humans today, as long as the robot can navigate the environment where the task is to be carried out, and have instruments capable of doing the necessary work.



Images from
<http://www.activrobots.com/robots.html>

Figure 1: Various mobile robots.

The main goals of this thesis is to develop a method capable of good enough control for the robot to safely move around in an office-like environment, have it avoid crashing into humans moving in the operating environment, as well as map the environment it has visited. The robot should also be able to seek out areas it has not yet visited, and explore those areas, rather than only drive around randomly. In addition, it should be possible to order the robot to return to a previously visited area, visible in a human-readable map set up by the robot during exploration. The robot should be controllable through wireless communication, using a web-interface to provide compatibility with different devices, allowing a human to tell it what to do.

Chapter 2 discusses the robot used for the project, and common techniques for achieving the desired behavior for the various events and situations the robot

must be able to handle. Chapter 3 deals with the mapping of the environment, specifically how the map is represented, and how to overcome some of the problems that arise during the mapping process. In chapter 4 the matter of dynamic obstacles is discussed. Chapter 5 describes the control program, robot movement, and results from driving the robot. The last chapter describes the needs and possibilities for improvements in future work.

2 Background

2.1 Robot specifications

The target robot for this project is a Pioneer 3-DX mobile robot (Figure 2) [10], capable of driving forward and backwards, turning, and detecting obstacles using ultrasound sonars, in eight different directions in front of and to the sides of the robot. It also keeps track of its current position relative its startup position, using odometry calculations from wheel movement. In addition, a camera is available and to be used to provide additional detection of humans. The camera is an AXIS 207MW Network Camera [2], mounted on the robot and accessed through a local ethernet connection. It has the capability of providing high-resolution JPEG images of 1280x1024 pixels at 12 frames per second, through an ethernet network connection.

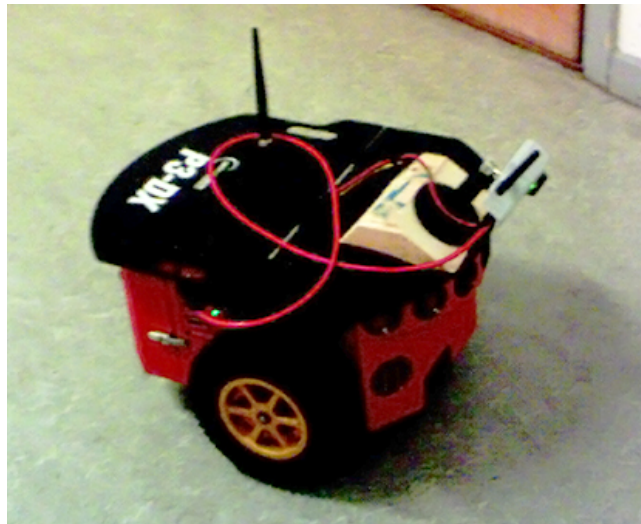


Figure 2: The Pioneer 3-DX robot, with the AXIS 207MW Network Camera mounted on the front.

Sonar input is achieved through bouncing ultrasound off the environment. Each of the eight sonars on the robot continuously return the distance to the closest obstacle, in the direction of that sonar. The directions are static relative the robot. The distance detection range is up to 5000 mm, and the detection rate is about 25 Hz. Figure 3 shows the sonar setup on the robot. Since the used robot only have sonars in front of it, moving backwards can be dangerous, if a dynamic environment is to be expected.

The robot is controlled by an onboard computer, running the software controlling the robot. It has a single-core Pentium III processor, operating at 850 MHz, and 256 MB of RAM, the computation power which must be taken into account when determining what methods are viable to use in controlling the robot. The control program interacts with the robot instruments using the ARIA API [1], Advanced Robotics Interface for Applications, provided with the robot. It is

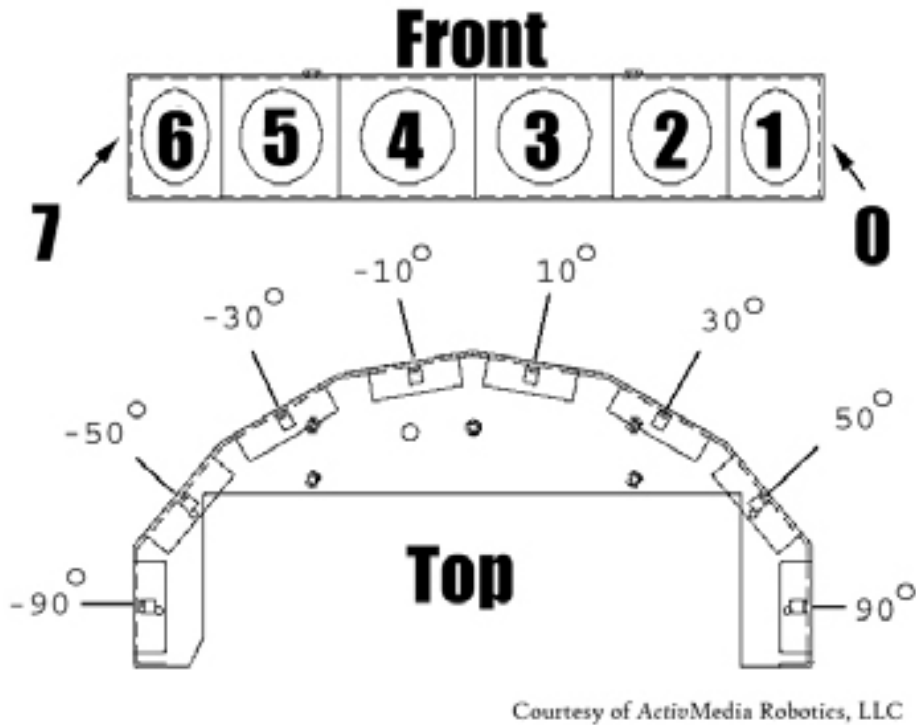


Figure 3: Pioneer 3 sonar array

a C++-based development environment, with a class-based system for controlling the robot, allowing a custom program to poll sensor data and control robot behavior through a higher level interface than the actual physical connections to the robot instruments. It provides cross-platform compatibility for developing on a simulator and on the actual robot, allowing for speedier testing of a program before running it on the robot. The API also provides a networking interface, for use in communication with other devices.

2.2 Ultrasound sonars

The sonars work by emitting ultrasound from the sensor, and listening for a returning sound, determining the distance to the obstacle the sound hit by measuring the time since the sound was emitted [9]. The sound spreads in a cone-shape outward, resulting in less accuracy on the actual obstacle position with increasing distance, as displayed in Figure 4. The exact distance is also not entirely accurate, but it is quite close, if the sound bounces back to the sensor without problems.

There are many known problems with sonars, such as that the sound can be reflected instead of bouncing back. When this happens, the sound bounce at an angle when hitting an obstacle, like a mirror, and does not return to the sonar. Sometimes the sound may bounce twice before returning to the sonar, resulting

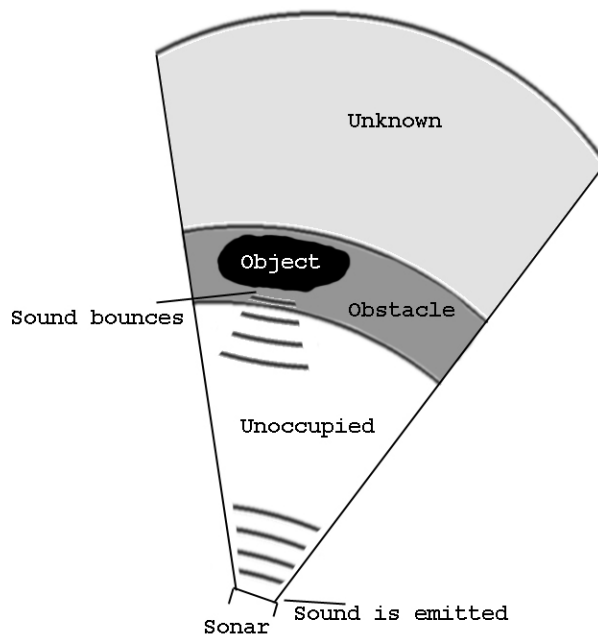


Figure 4: The sonar model. When the sound is returned it can be determined that the white area is free from obstacles, and that somewhere in the darker gray area there is an obstacle. No information is obtained about the lighter gray area.

in an erroneous distance estimate for the obstacle.

2.3 Map

To perform tasks taking into account the world, apart from what is currently seen by the sonars, the robot must have some memory of the areas it has visited. Such tasks include exploration, pathfinding and many more functions. To accomplish these tasks some map structure is required, keeping track of the robot's view of its operating environment. There are several different map structures that may be used for this purpose.

A common map structure is a grid, where each cell represents some area in the physical world. Each cell contains the available data of that particular area, creating a map that can be more or less detailed, depending on the size chosen for the areas each cell represents. The simplest form is a binary map, where each cell is either passable or impassable, and the robot only drives through passable cells. The sonars are used to find obstacles, and the appropriate cells will be marked impassable when such are detected. Figure 5 displays how a few boxes in an environment can be represented in a grid-map, where the grid covers the ground plane. A two-dimensional map is often sufficient, when the robot is used only on planar floors.

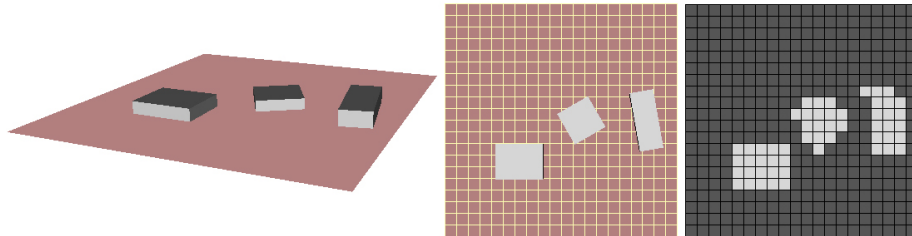


Figure 5: A grid map representation of a simple environment. On the right is the map, where the lighter cells can not be entered by the robot.

It is often hard to say with certainty that a particular cell is passable or impassable, so instead of a binary map a probability map can be used. Each cell contains the estimated probability that the cell is impassable, and the robot stays away from cells with a high enough probability [8]. Whenever a sonar detects an obstacle, the corresponding cells will have their probability increased, while a cell passed by a sonar signal has its probability decreased. This results in a more dynamic map, where false detections or relocated objects are erased from the map when newer information is gathered, and a more robust map is obtained. Figure 6 shows how such a map might look like after the robot has surveyed the environment. The lighter a cell is, the greater the probability that it is impassable.

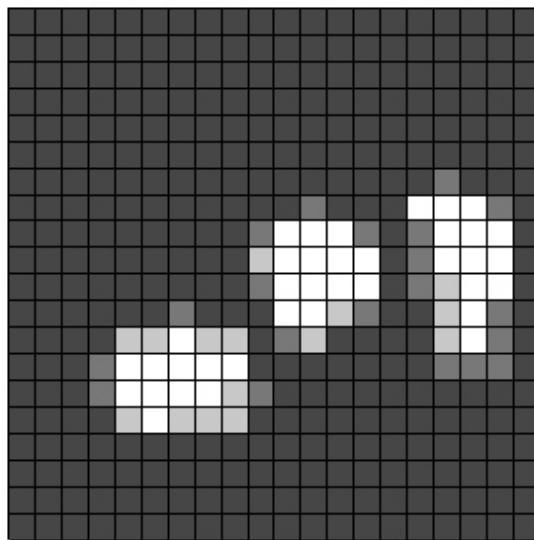


Figure 6: A probability-grid.

There is also the possibility of creating geometric or topological maps [11]. In a geometric map the world is represented by some geometric shapes, such as lines or squares. These can also be constructed from sonar input, but often

require more data to reliably map the environment with acceptable precision, so a grid map is often used as a first step, and then a geometric map can be constructed from that data. The same is true for topological maps, where the main map structure is a graph of positions in the world, and the edges are paths for the robot to move. Such a map is perhaps the most beneficial to obtaining optimal movement in a mapped environment, since it can be constructed from a complete grid-map, calculating the safest or fastest routes to travel between two known locations. Figure 7 shows how a few key positions can be chosen in a grid-map, and this data can be used to have the robot travel between them. With such a structure most of the map can be ignored, since only a few points are considered relevant targets.

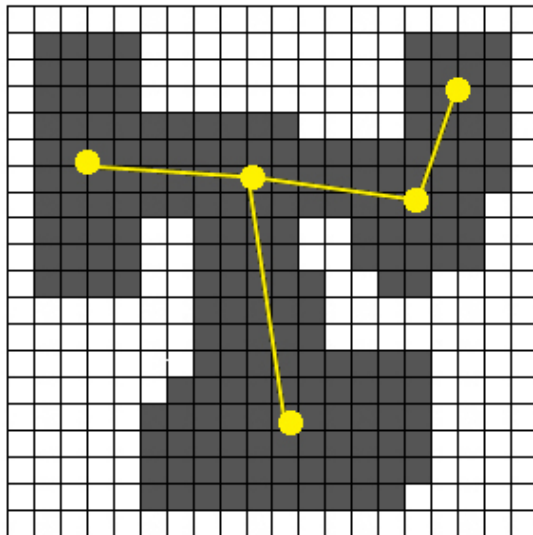


Figure 7: Topological data from a grid-map. The robot can use the information of key positions to only travel chosen routes.

2.4 Odometry and SLAM

Odometry is a method of measuring movement, and works by measuring the rotation of the robot's wheels. By knowing the circumference of the wheels, the distance traveled can be calculated from the rotation, and an estimate of the robot's position in the world is obtained. Odometry is far from perfect, and errors will almost always be present. These errors are not static, but accumulate with time, often resulting in very large differences between where the robot thinks it is located, and where it is truly located [8].

SLAM stands for simultaneous localization and mapping, and is the problem of constructing a map that correctly describes the true environment. As errors in the robot's estimated position grow large, a map constructed using the position estimate as a base will become severely distorted. Correcting this problem can

be very complex, and is an area of much research in mobile robotics. There is no known flawless method, but several methods exist, taking probabilistic approaches to estimating a map true to the environment [12].

2.5 Dynamic obstacles

In addition to static objects being recorded in the map, there can be dynamic objects moving around, such as humans or other robots. These should preferably not be visible in the map, since they are temporary, and can also require additional strategies to avoid. There is no currently known algorithm that perfectly solves the problem of dynamic environments, but several best effort techniques have been tried [11], the most common being to use camera input to distinguish between various object types. With a probability map, as described in section 2.3, dynamic worlds are handled fairly well by sonar, but multiple different inputs can be very beneficial to detecting moving objects. One possibility is to use the camera image to detect humans or other objects known to move, that are indistinguishable from a static environment by the sonar input. This requires some type of image processing algorithm, capable of detecting wanted features in the image, and is often achieved through pattern recognition on the input image. Various features can be recognized in the image, providing more information for the robot, allowing it to avoid an area where for example a human face is located. Other than visual input, high-resolution distance sensors, such as laser range finders, can be used to give better resolution than the sonars. These are however not capable of recognizing objects in the same way as camera image processing.

3 Mapping

This section explains the mapping process, as well as how the map is used for path finding, to allow the robot to navigate in the environment.

3.1 Sonar mapping from odometry orientation data

The method of choice for world representation in this thesis is a simple grid-structure. Each cell in the grid represents some area of the robot's operating environment, and contains the known properties of that particular area. The robot uses the sonars to detect obstacles in the world, and together with the odometry position information it can determine what part of the grid the obstacle resides in. The cells within some distance of that position are then marked as a blocked part of the map, and are to be avoided by the robot when exploring further. When such blocked cells are detected, it is reasonably likely that the cells in between the robot and the detected obstacle are not blocked, since otherwise the sonar would have detected a closer obstacle instead, as discussed in section 2.2.

Every cell represents a 100mm x 100mm area of the world, and contains the probability that an obstacle resides in that area. Everytime a sonar-signal travels through a cell and fails to detect an obstacle there, that probability is decreased slightly. This method removes false detections, or detections of dynamic objects such as humans moving around, so that the robot will eventually cease to avoid an area previously occupied, if it is later vacated. Figure 8 shows how estimated positions of obstacles in a room are converted into a probability-map, used by the robot for navigation.

The method chosen for mapping probabilities is to set cell values to certain obstruction whenever an obstacle is detected, and decrease this probability slowly if later sonar data suggests otherwise. This model was chosen because false detections in open areas proved non-existent, and it leads to a faster change in paths chosen in the map when new obstacles are detected. Negative effects of the method can include unnecessary recalculation of paths when passing humans in the environment leave traces in the map, but it has shown over-all good results during the project. The probability is decreased by slightly less than 1% of maximum everytime a subsequent sonar detection indicates the cell to be open. The map is updated from current sonar data every 40 ms, resulting in about a 5 second investigation time for a cell to be considered completely safe to move through, after it has once been considered occupied. The robot will stay farther away from a cell the higher the probability of it being occupied is, but is allowed to enter cells of less than 80% probability of occupancy, so only one second of investigation is necessary before a previously occupied cell becomes passable again. Section 3.2 on pathfinding discusses the cell avoidance algorithm in more detail.

The algorithm executed at every map update is as follows:

- Gather data from all eight sonars, determining the position of each detected obstacle.

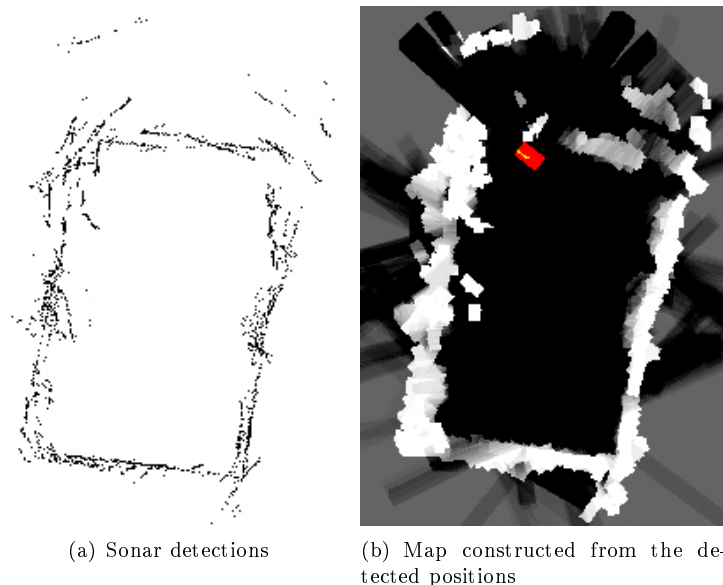


Figure 8: Estimated positions of sonar-detected obstacles are used to construct a probability-map. Lighter colors means a higher probability of the area being obstructed.

- Decrease obstruction probability in cells within an area between the robot and a detected obstacle.
- Mark cells in proximity of each obstacle position as occupied.

Subsequent detections by the same sonar are used to obtain a better approximation of cell occupancy. If two updates has the same sonar detecting obstacles in close proximity, the area in between these obstacles is stretched in the direction of each detection and considered as a larger obstacle. This helps in avoiding gaps in walls when the robot turns quickly, while allowing a shorter safety distance in a direction likely to be away from the wall.

Accuracy can very well become a problem when using sonar data, depending on what detail is desired in the map, since the actual obstacle position is only determined to within some area. Though reasonable accuracy is usually obtained, the sonars are far from perfect, and cases of false detections and obstacles being missed by the signals must also be taken into consideration. Depending on the robot's movement speed at the time of detection, calculated obstacle positions may also be slightly out of place. Some of these problems are visible in Figure 8, and will be discussed further in section 3.4.

Once the robot has moved around a bit, an approximation of the world is obtained from the sonar signals, and can be used to plan the next move. For exploration, this is to find the closest cell not yet classified, and drive to a point where it can be investigated. This is accomplished by following a path towards the nearest cell, a path which can be calculated with a simple graph-search from

the cell in the grid where the robot is currently located. Once the robot is on the path to its target, its map might very well change quite drastically, so the path will need continuous recomputation, to make sure the robot keeps going in the correct direction, or abandons its target if upon closer inspection it is found to be blocked.

3.2 Pathfinding in the map

When in exploration mode, the robot continuously search for the area still unknown to it with the lowest cost, according to a cost function for the path to that area, and proceeds by driving along the chosen path. The method of finding a path was designed to keep the robot far away from obstacles as much as possible, abandoning a shorter path for a safer one, while still allowing movement through narrow passages if necessary. The behavior is achieved through deciding cell costs as the total probability of obstruction in cells closer than 500mm, with a weight multiplier decreasing quadratically with distance. The actual path is calculated with Dijkstra's algorithm [6], beginning on the cell the robot is currently occupying, stopping when an unknown cell is found as the closest target. In addition, a cell with a probability of 80% or higher of being occupied is never entered, even if there is no other path to explore.

Pathfinding is handled separately from actual movement. Once a path is found, the robot's movement program is instructed to move along the path. Local obstacle avoidance from both sonars and the camera is used when carrying out the actual movement. This works by applying a basic local movement method on top of the path, where the robot is told to drive forward, and turns away if it would ever get too close to an obstacle, even if the object is not visible in the map.

3.3 Odometry drift

Keeping track of the robot orientation through wheel movement is not an exact science. Slippery floors, or slight differences between the wheels, can make the robot's perceived orientation vary significantly from its true orientation. For example, if the robot drives in a straight line, its odometry program might believe it has turned slightly. Though the robot will still function correctly in the place its currently at, these errors accumulate to unacceptable distortions in the map, if the robot is to be able to find its way back to a previously visited area.

As can be seen in Figure 9, where sonar detections of a rectangular room the robot explored are plotted, the perceived walls are nowhere near their true places after a while. At the end of the run, the whole image has turned 90 degrees, in just the couple of minutes it took the robot to wander a couple of laps around the room. This obviously makes a correct map of larger areas impossible to obtain without means of correcting this error.

Compensating for drift in rotation can be achieved by searching for walls that are not seen at the angle they should be. For arbitrary environments this is a hard problem, since there might very well in some environment be a wall that

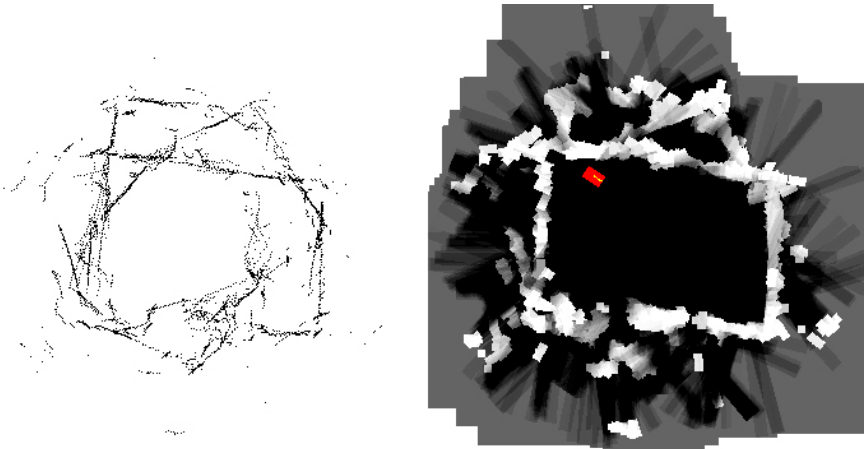


Figure 9: Map distortion from odometry drift.

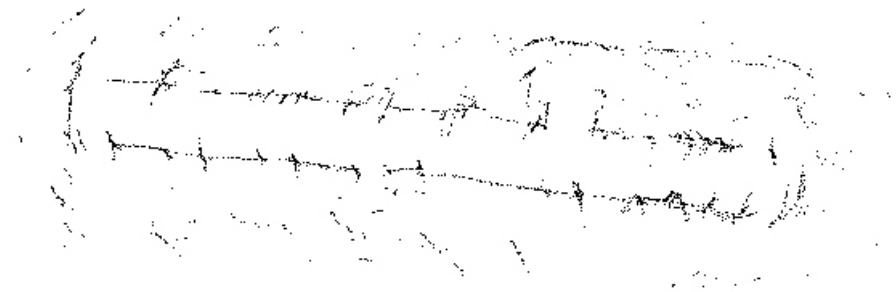
is curved. If there are known attributes of the environment however, the problem is easier to solve. In a normal indoors environment, most walls intersect at 90 degree corners, and do not bend, something that can be exploited to keep the robot on track. By inspecting recent sonar hits, looking for points forming straight lines, walls can be identified and their angle in the map be determined. Such lines are located using the Hough transform [7]. When a wall is found at an angle close to but not perfectly vertical or horizontal, it is likely that the robot has drifted slightly in its orientation. The control program searches for straight lines every 2 seconds, stores information about the most prominent line, and compares it to the results from the last 4 searches before it. If 5 lines are found to be bent in the same direction, no more than 20 degrees from straight horizontal or vertical, then these lines are estimated to be 90-degree walls in the real world, and the robot is considered to have drifted. The robot's internal direction estimate is corrected at this point, to align with the true world, as estimated from the detected lines.

Figure 10 displays the difference when using the correction algorithms to avoid the error, when exploring a corridor. The real corridor is completely straight, but the odometry position suffers from an increasing error, making the perceived world severely distorted without correction.

If it takes some time to determine a good enough estimation of the drift, the robot can already have moved some distance in a direction other than what the data suggests, offsetting its position estimate. This can make the map distorted, even though the direction is kept correct. To correct for this behavior, the robot must also be repositioned to account for the incorrect movement data. It is assumed that once the robot is ordered to move, it will have an approximately constant drift in perceived relative real orientation, and therefore the robot's perceived movement path is stored. Once a drift is detected and adjusted, the stored perceived path is re-calculated, each part of the path redirected to counteract the total error, finding the likely real movement path. Figure 11 shows how the path is corrected, if the correction algorithm detects that the robot has



(a) Without correction.



(b) Corrected map.

Figure 10: By using map data for localization the robot attempts to compensate for the odometry error.

moved straight, when the odometry indicates that it has turned.

3.4 Sonar problems

As can be seen in Figure 12 showing a map of the rectangular room, the odometry correction algorithm keeps the room reasonably straight, and the robot can find its way back to a previously explored position with good accuracy, but other problems are visible. Though there are no openings in the walls of the room, there are several sonar detections far behind them, which cause the robot to believe that the world has changed, modifying the map to include areas not really there. During the experiments in this project no wall was ever found to always give false detections, so the robot's avoidance program will still keep it away from the wall, even though there are enough false detections to cause the map to show a very low probability of an obstacle there. The local movement program avoids an area if there is currently any obstacle detected in the way, even if it is only a single detection.

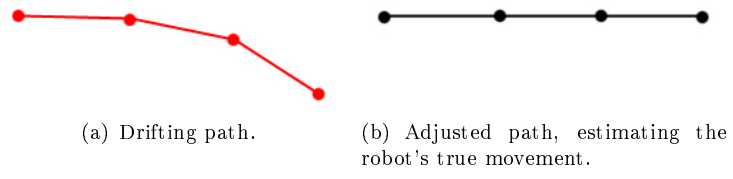


Figure 11: Corrected path.

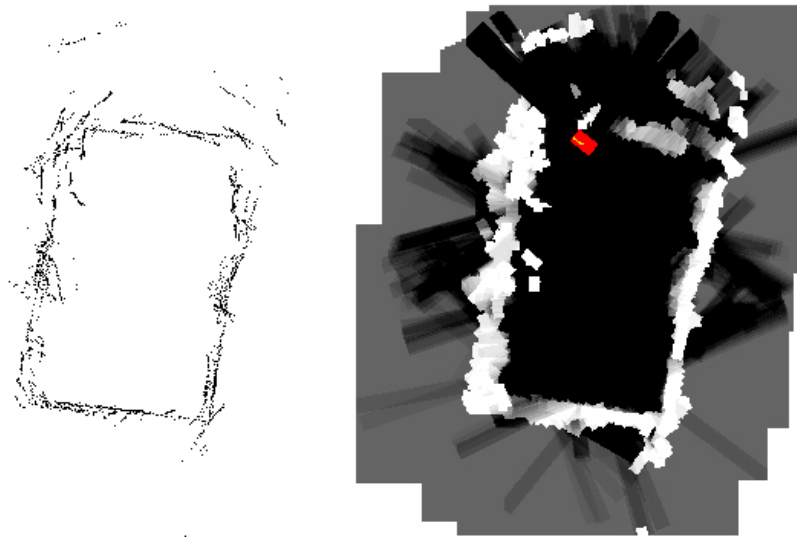


Figure 12: In the map on the right it can be clearly seen how the sonars return faulty distance information.

4 Dynamic obstacles

Avoiding humans is often done automatically by the sonars, treating them as just another obstacle in the environment. The sonars are not fail-safe however, since the sonar directions might be far enough apart for small objects to slip between them. This will also pose problems for other thin obstacles, such as table-legs and similar kinds of objects. Image-processing algorithms can be used on the camera image to detect objects of known types, avoiding them separately in addition to sonar-detected obstacles. It can also be desirable that the safety-distance to a human be greater than to a wall or an inanimate object.

4.1 Sonar

Since the sonar will detect humans close to the robot, it will avoid them with the same strategies as avoiding the rest of the world. This way humans will also leave traces in the map, though once relocated the robot will correct this with subsequent sonar detections of the same area. Everytime a sonar measurement indicates that no obstacle is visible the map probability will be decreased slightly. Experimentally a time of 5 seconds has been chosen for a detected obstacle to be completely removed from the map, if the sonars sees the area as unobstructed.

4.2 Video

A camera is used to image the environment in front of the robot, and the image is analyzed to find humans present in the image. There are many methods that can be used to detect relevant features in the image, for example human face and gesture recognition [3, 5]. The detection algorithm chosen here is a simpler one, and works by finding close to vertical edges from human legs in the image. This method is not too good, and can often give false positive detections, but it is reasonably cheap in processing time and detects humans with a good enough probability to allow the system to function. To easily achieve a reasonable estimate of whether there is a human in front of the robot, and at what distance, the camera is inclined to image the floor close to the robot, as shown in Figure 13.

The algorithm first detects possible vertical edges by searching for horizontally adjacent pixels where the difference in color or luminance is large, as in the Canny edge detection algorithm [4]. These edge pixels are then traced, searching for vertical lines, allowing some noise and bend in the line, to find lines that might be the edges of human legs. This can be done for example by using the Hough transform, as in [11], but since only near-vertical lines are interesting an easier trivial approach is taken. A loop checks edge pixels from top to bottom, counting how many occur adjacent to each other, and if a vertical line of significant length is found it is stored as a probable edge in the image. After all such edges are found, they are inspected for pairs at an appropriate distance from each other, in which case a human is considered present at that position in the image. The image resolution and framerate chosen for the camera can be an

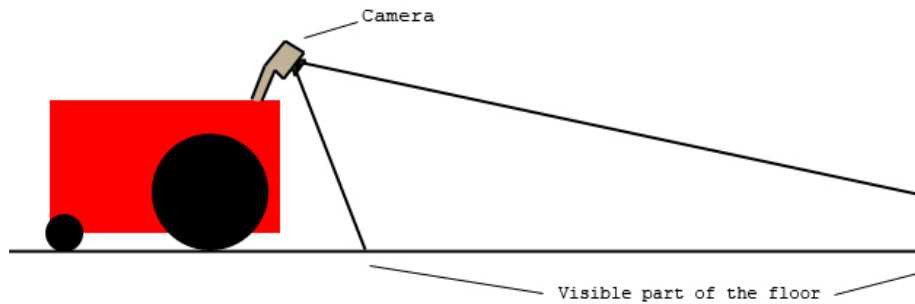


Figure 13: The camera is inclined to image the floor in front of the robot.

issue, requiring balance between image detail, reaction time, and performance. A resolution of 480x360 and an update frequency of 5 frames per second was found to work well, in all these regards.

The camera is statically placed on the robot, so each pixel imaging the ground will always correspond to some fixed distance from the robot. These distances need to be mapped at some acceptable resolution, and then each detected object's lowest point in the image is used to decide at what position that object is located. Since the robot only moves on the floor this method works well, as long as the floor is not too reflective, so that the point where the object meets the floor is properly detected.

Figure 14 shows how an image is projected onto the ground-plane, and how the distance vectors to the detected object is determined. The grid size is 20 cm. As can be seen, precision is much better closer to the robot, since many more pixels will be mapped to a smaller area.

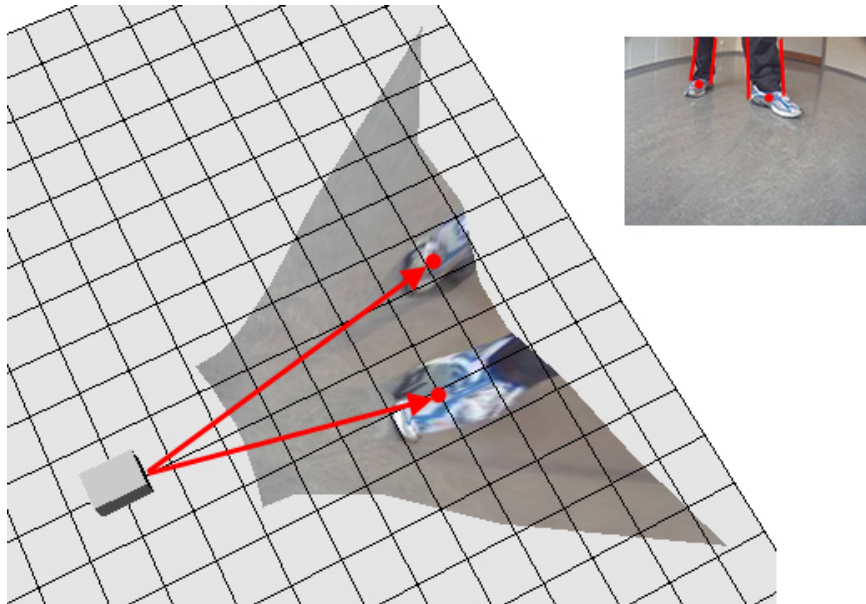


Figure 14: A camera image projected on a ground-plane, allowing distances to be calculated.

5 Experiments and results

5.1 Control program and interface

On the onboard computer, the actual robot sensor control is handled by the ARIA API in the background using threading. The API is controlled by adding action class objects that are called by the API, when the robot wants information on how to behave. Because of the single-core CPU, there is rarely anything to gain from threading computations or control, but it can lead to performance penalties in synchronization, so the software does as much as possible in a single thread. Each cycle the current sonar input is checked, the map updated, and if needed path-finding and changed orders are handled to determine the desired robot behavior. The appropriate actions are then activated to give the robot instruments instructions via ARIA. In addition, the network is checked regularly for incoming data, and requests for map updates and other information by connected clients are handled.

Figure 16 shows the main parts of the program. The logic is handled in RobotHandler, in the update cycle run every 40 ms. This part of the program interfaces with the ARIA API for robot control, and with the map for updates and path-finding. The localization map is given the same input as the probability map, but uses it to find walls that can be used to correct odometry orientation errors. The image provider part is connected to the camera, and continuously receive the latest video image seen by it. The image is queried by the robot handler, and searched for humans in vicinity of the robot, which are then remembered and avoided when moving further. The HTTP server accepts incoming connections from a web-browser, and allows monitoring and control of the robot. When connecting to the robot a page with an image of the current map is visible, along with information on the robot's current orders and intentions. Changes to these can be initiated from the web-page, such as telling the robot where to travel next. Figure 15 shows the web-page.

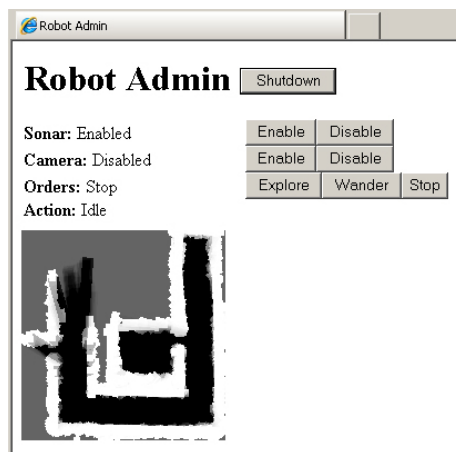


Figure 15: The web-page for controlling and monitoring the robot.

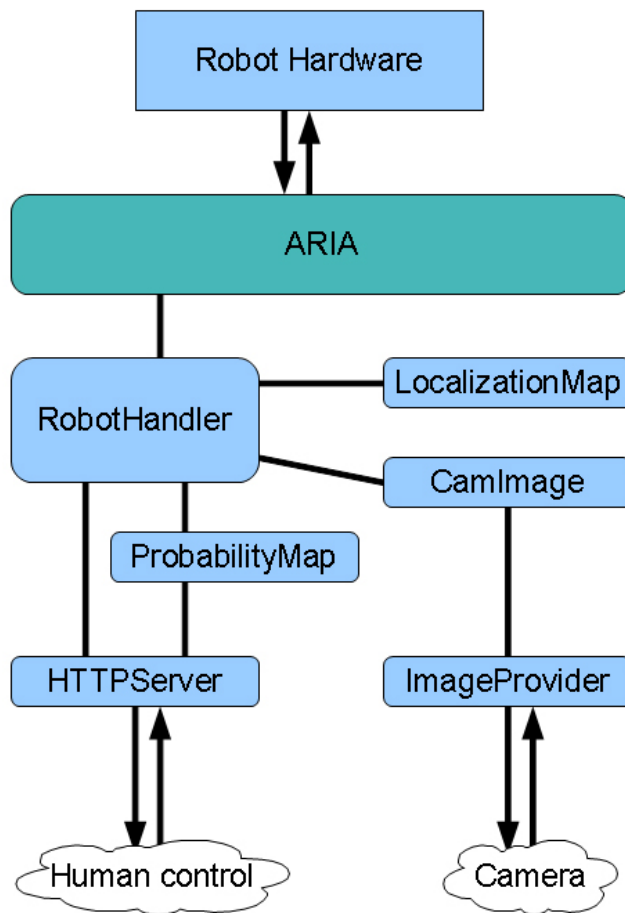


Figure 16: The main parts of the robot control software.

5.2 Robot movement

When the program tells the robot to actually move, it does so by instructing ARIA with what velocity and in what direction the robot should drive. This is achieved by using objects called actions, added to the ARIA active action group. Each action has its execution method, called by ARIA when the robot instruments want instructions, and a priority depending on how important it is. It is the control software's responsibility to set the appropriate priorities, telling ARIA which action is most important. The actions themselves can be custom-written, if input is needed from the map or another construct of the control software, or some of the built-in actions to move or stop that only use the current input from the robot's instruments. The highest priority actions are those stopping the robot from crashing into the environment, and their execution methods works by checking if there is an obstacle in the way, in which case they instruct the robot to stop or turn. There are a few of these actions, turning for obstacles some distance in front of the robot and stopping for those very close, both for sonar and camera detected obstacles. The action with slightly

lower priority is the movement action, telling the robot to drive forward or turn, depending on where it is instructed to go. This method guarantees that even if the robot is ordered to follow a path, a higher priority safety action will stop it if an obstacle appears in the way. Figure 17 illustrates how this works.

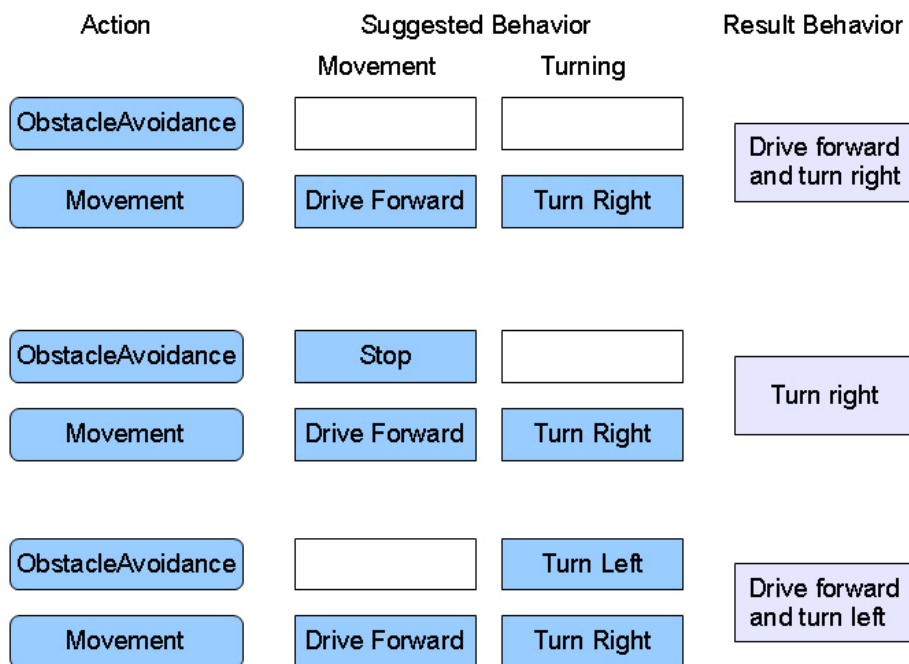


Figure 17: ARIA requests behavior suggestions from all actions, and uses the highest priority suggestion to determine robot behavior. This figure shows three examples, where the ObstacleAvoidance action has higher priority than the Movement action. The Movement action only considers where it wants the robot to go when suggesting an action, and the ObstacleAvoidance action only considers if and how the robot needs to act to avoid crashing.

5.3 The robot handler

The RobotHandler is the main component of the control software, directing input and output from the other components, and making decisions on what the robot should do next. It mainly interfaces with the ProbabilityMap, the ImageProvider, through obtaining camera images from it, the LocalizationMap, and the underlying ARIA API. The purpose of the LocalizationMap is to find errors in the robot's perceived position, as described in Section 3.3. It is given the sonar input every update, and if it finds indication of an error it reports back with this information, which is used by the RobotHandler to correct the error. The ImageProvider connects to the camera, continuously receiving video frames from it, and the RobotHandler requests the latest available image whenever dynamic obstacles should be detected in it. The HTTPServer communicates with the RobotHandler whenever it is necessary to serve a request for information, or when orders for the robot have been received. The main update sequence

run by the RobotHandler follows:

- Gather data from all sonars, and input to the ProbabilityMap.
- Request the latest video image, and try to detect humans in the image. This step is not run every frame, since the image is not updated that often.
- Input the sonar data to the LocalizationMap, and handle positional errors if present.
- Handle current orders, updating robot actions if necessary.
- Calculate movement paths from the ProbabilityMap, when necessary.

5.4 Map results

The map is kept aligned with the world quite well, but the simple correction algorithm used to compensate for odometry errors is not perfect. Since it does not actually determine the robot's position by comparing sonar data to the map, but rather simply adjusts the robot orientation depending on the estimated drift from wall angles, there is often slight accumulating errors. During exploration of a larger area the map can suffer from errors, when the true robot position is not obtained with good enough precision from correcting the perceived movement path based on the direction found from wall angles. Depending on sonar precision, and movement during the correction process, detected walls may not align exactly with their true counterparts, which can also result in the robot aligning itself to a faulty perceived environment. The result is that though the map is kept well aligned angle-wise, the robot can experience positional offsets when returning to a previously explored area, as visible in Figure 19. Though there are problems, the map is usually kept reasonably close to the true world. Figure 18 shows how the robot has moved a longer distance through a corridor.

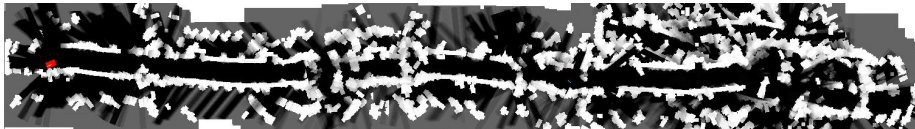


Figure 18: In this image the robot started out in the right end, first exploring with the odometry correction algorithm disabled. This led to significant errors, but after the correction was enabled the map was kept reasonably straight. The robot then drove from right to left through a corridor, passing a few humans that can be seen to have left tracks in the map at certain positions.

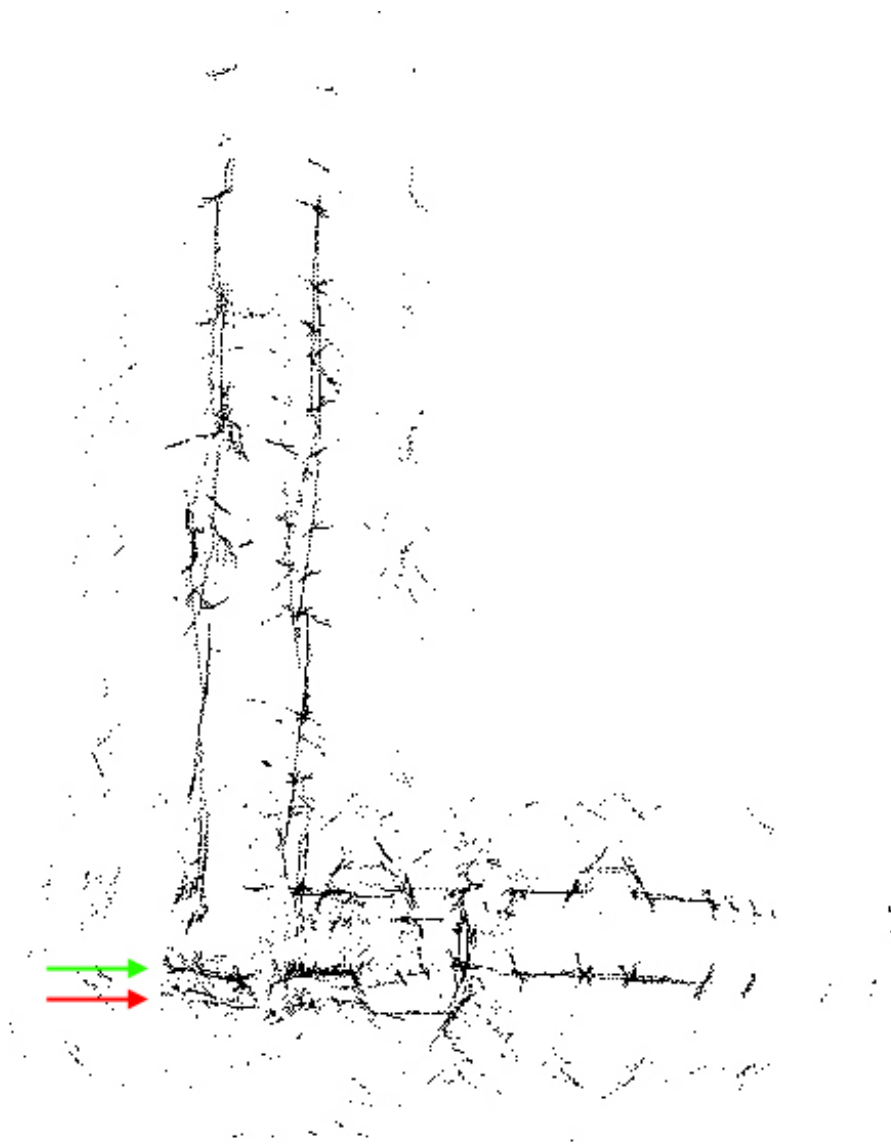


Figure 19: Plotted in this figure are sonar detections as the robot has traveled from the lower right corner, up to the top, then back down. Drift is compensated well horizontally, but a vertical offset error has occurred.

6 Conclusions and future work

In areas such as this there is always room for improvement, since the instruments rarely give perfect data. Fine-tuning implementation and algorithms might gain significant improvement, and calibration of variables is an important part of achieving good results. Apart from this fine-tuning, there are many algorithmic improvements and additions that can give the robot a better overall behavior.

The robot usually moves well and safe in its operating environment, but there is room for much improvement. When moving through narrow passages the robot will often pause and move very slowly trying to keep away from the walls, continuously recalculating the best path. Using a more advanced map structure, such as a topological map, could lead to better paths being calculated quicker, allowing for faster and safer movement through previously mapped environment. This can also enable the robot to move faster, and avoid excessive calculations during movement, saving both time and processing power.

During the planning of the project odometry drift was not taken into account, but it quickly proved itself necessary to correct, to achieve acceptable behavior. The correction for the drift in this project is very basic, and assumes a lot about the operating environment, and true SLAM could significantly extend the possibilities for the robot to map complex environments. It is often impossible to say beforehand how the robot will drift, and in order to achieve a reliable behavior in difficult situations a better localization algorithm is needed, one less reliant on odometry. In more complex usage scenarios, where the robot might even get influenced or moved by external forces, localization by examination of the environment around it would be a necessity.

Dynamic objects are handled fairly well separately from the map, but as discussed in section 4.1 they still leave traces in the map. Avoiding this by disregarding sonar-detected obstacles known to be human can be a good idea. The algorithm used in this project is very basic, and can easily give false positives, which is not acceptable if obstacles are to be removed from the map. A more accurate algorithm, detecting humans with a very high probability, can make such a method viable, if there is enough processing power for it to run.

References

- [1] Mobilerobots support site, <http://robots.mobilerobots.com/> (verified 2009-06-22).
- [2] *AXIS 207W / AXIS 207MW Network Camera User's Manual*, 2007.
- [3] H.-J. Boehme, U.-D. Braumann, A. Brakensiek, A. Corradini, M. Krabbes, and H.-M. Gross. User Localisation for Visually-based Human–Machine–Interaction. In *Proceedings of the Third IEEE International Conference on Automatic Face and Gesture Recognition*, pages 486–491, 1998.
- [4] J Canny. *Trans. Pattern Analysis and Machine Intelligence* 8(6). IEEE, 1988.
- [5] H.I. Christensen, D. Kragic, and F. Sandberg. Vision for Interaction. In Hager, Christensen, Bunke, and Klein, editors, *Lecture Notes in Computer Science*, volume 2238, pages 51–73. Springer, October 2002.
- [6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (2)*. MIT Press and McGraw-Hill.
- [7] David A Forsyth and Jean Ponce. *Computer Vision A Modern Approach*. Prentice Hall, 2003.
- [8] Robin R. Murphy. *Introduction to AI Robotics*. 2000.
- [9] Ulrich Nehmzow. *Mobile Robotics: A Practical Introduction*. Springer, 2000.
- [10] *Pioneer 3 Operations Manual, Mobile Robotics Inc.*, 2007.
- [11] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. 2004.
- [12] Sebastian Thrun. Robotic mapping: A survey. 2002.

A User's Guide

Using the robot program is fairly straight-forward. First a wireless connection must be set up between the robot and the device used to control it, and then the RobotAI program must be started from a terminal. Once the program is up and running it will listen for incoming HTTP connections on port 80, and any web-browser can be used to control the robot, as seen in Figure 20. The page displays the robot's current orders and the action it is presently performing, as well as whether the camera and sonars are enabled. By pressing the available buttons the sonars and the camera can be enabled and disabled, and the robot can be told to either explore, wander, or stop. It is not recommended to disable the sonars unless the robot is ordered to stop, or if there are no obstacles in the vicinity, since it will not be able to detect and avoid obstacles without them. The image on the page shows the map as the robot currently sees it, and it is automatically reloaded to reflect changes every two seconds. In addition to the order-buttons, the map image can be clicked to order the robot to move to the point clicked.

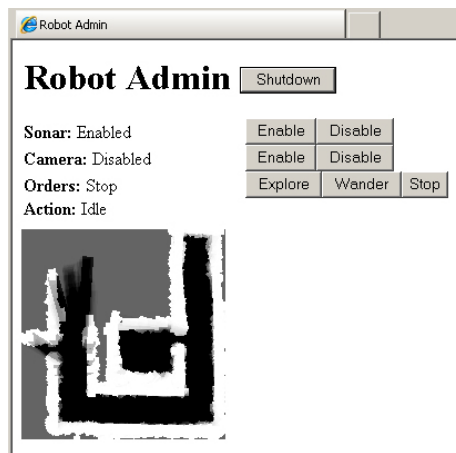


Figure 20: When connecting to the robot using a web-browser, the control page can be used to give the robot orders.

The source code for the program is available at <http://fileadmin.cs.lth.se/ai/xj/ErikRufelt/RobotAI.zip>.