

# **Testing cognitive mapping ideas on a mobile robot**

**Thomas Brunner**

A thesis submitted to  
Auckland University of Technology  
in partial fulfilment of the requirements for the degree of  
**Master of Computer and Information Sciences (MCIS)**

July 2011

School of Computing and Mathematical Sciences

Primary Supervisor: Prof. Wai Yeap

# Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>7</b>
<b>2</b>	<b>THE PERCEPTUAL MAPPING PROBLEM.....</b>	<b>9</b>
2.1	GENERAL BACKGROUND ON COGNITIVE MAPS.....	10
2.2	YEAP'S THEORY OF COGNITIVE MAPPING.....	12
2.2.1	<i>Representing local spaces.....</i>	<i>12</i>
2.2.2	<i>Remembering immediate surroundings.....</i>	<i>14</i>
2.3	A NEW THEORY OF PERCEPTUAL MAPPING.....	15
2.4	CONCLUSION .....	20
<b>3</b>	<b>IMPLEMENTATIONS .....</b>	<b>21</b>
3.1	TRADITIONAL ALGORITHM.....	21
3.1.1	<i>Identifying surfaces .....</i>	<i>21</i>
3.1.2	<i>Integrating views by transformation.....</i>	<i>22</i>
3.1.3	<i>A simple error correction method.....</i>	<i>24</i>
3.1.4	<i>Experiments.....</i>	<i>26</i>
3.1.5	<i>Discussion .....</i>	<i>29</i>
3.2	COGNITIVE MAPPING ALGORITHM .....	31
3.2.1	<i>Integrating surfaces into the MFIS .....</i>	<i>31</i>
3.2.2	<i>Building a network of ASRs.....</i>	<i>35</i>
3.2.3	<i>Experimental Results .....</i>	<i>38</i>
3.2.4	<i>Discussion .....</i>	<i>45</i>
3.3	MAKING THE ROBOT GO HOME .....	46
3.3.1	<i>Motivation.....</i>	<i>46</i>
3.3.2	<i>Deciding which way to go.....</i>	<i>47</i>
3.3.3	<i>Using the MFIS while travelling.....</i>	<i>49</i>
3.3.4	<i>Following a known path from memory .....</i>	<i>52</i>
3.3.5	<i>Experimental results.....</i>	<i>55</i>
3.3.6	<i>Discussion .....</i>	<i>61</i>
<b>4</b>	<b>CONCLUSION .....</b>	<b>62</b>
<b>5</b>	<b>REFERENCES.....</b>	<b>64</b>
<b>6</b>	<b>GLOSSARY .....</b>	<b>67</b>

## List of Figures

Figure 2.1: A network of ASRs (Jefferies & Yeap, 2001) .....	13
Figure 2.2: A route in the ASR network .....	13
Figure 2.3: Navigating between exits inside an ASR .....	13
Figure 2.4: Surfaces inside the MFIS .....	14
Figure 2.5: Error propagation during mapping .....	17
Figure 2.6: Errors in a cyclic environment (Thrun, 2003).....	19
Figure 2.7: ASRs after traversing a circuitous route (Jefferies & Yeap, 2001) .....	19
Figure 2.8: MFIS after traversing a circuitous route (Jefferies & Yeap, 2001).....	19
Figure 3.1: Recognizing surfaces .....	22
Figure 3.2: Coordinate transformation between two views .....	23
Figure 3.3: Bad surface placement.....	23
Figure 3.4: Measuring a pair of matching surfaces.....	25
Figure 3.5: Good robot in a small environment.....	27
Figure 3.6: Bad robot in a small environment .....	27
Figure 3.7: Floor plan of the large office environment.....	28
Figure 3.8: Good robot in a large environment .....	29
Figure 3.9: Bad robot in a large environment.....	29
Figure 3.10: Matching surfaces .....	33
Figure 3.11: Surfaces in the current view .....	34
Figure 3.12: MFIS when inserting a new surface .....	34
Figure 3.13: Separating spaces with a narrow passage .....	35
Figure 3.14: Wideness when crossing an exit.....	36
Figure 3.15: Surfaces seen across the exit .....	37
Figure 3.16: Surface placement in the MFIS .....	38
Figure 3.17: A bad reference target.....	39
Figure 3.18: Surface displacement when picking a bad reference .....	40
Figure 3.19: MFIS of large environment A .....	41
Figure 3.20: Floor plan of large environment A.....	41
Figure 3.21: Surfaces detected by laser sensor inside a corridor .....	42
Figure 3.22: MFIS of large environment B .....	43
Figure 3.23: Floor plan of large environment B .....	43

Figure 3.24: Large environment B separated into ASRs .....	44
Figure 3.25: Movement options in a narrow environment .....	47
Figure 3.26: Movement options in a wide environment .....	47
Figure 3.27: Moving away from a dead end .....	48
Figure 3.28: The most promising option.....	49
Figure 3.29: Forgetting old geometry while moving.....	50
Figure 3.30: Robot gets stuck through inaccurate MFIS.....	51
Figure 3.31: Moving away from a dead end .....	52
Figure 3.32: Exploring the dead end again .....	52
Figure 3.33: Waypoints for the path home.....	53
Figure 3.34: Shortcuts in a path .....	54
Figure 3.35: Robot travelling home around a corner .....	56
Figure 3.36: Visibility of MFIS geometry .....	57
Figure 3.37: A valid movement option?.....	57
Figure 3.38: Robot travelling home and taking a simple shortcut.....	58
Figure 3.39: Robot travelling home around obstacles.....	59
Figure 3.40: Dense collection of obstacles .....	60

## **Attestation of Authorship**

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Thomas Brunner, 04/07/2011

A handwritten signature in blue ink that reads "Thomas Brunner". The signature is written in a cursive style and is positioned above a solid horizontal line.

## **Acknowledgements**

I wish to express my sincere thanks and appreciation to Prof. Wai Yeap for his attention, guidance and insight during this research. His support, especially during the preparation of the thesis, has been invaluable.

## **Abstract**

A recent theory of perceptual mapping argues that humans process spatial information in a different way than previously thought. In particular, the theory suggests a process which, unlike SLAM, does not correct perceptual errors and does not need to integrate successive views when computing the map. Such a process could equally be applied for robot mapping.

The purpose of this study is to implement the theory on a mobile robot, and see what maps can be produced. Towards this end, an algorithm based on it has been implemented and tested on a mobile robot equipped with a laser sensor. Its performance has been analyzed in relation to the traditional SLAM approaches, and different experiments have been conducted which cover typical problems of robotic mapping. Additionally, a simple method has been shown which enables a robot to autonomously navigate using the map created.

The results obtained showed that the generated maps do indeed preserve a good layout of the environment, thus supporting the integral claim of the theory. It was also found that basic navigation with the produced maps is possible. The principal conclusion is thus that the theory shows much promise and could be used as a foundation for further research of human cognition and to develop new algorithms for robot mapping.

## 1 Introduction

How do humans perceive their surroundings? How do we see, process and remember our environment? And how do we use this information to navigate? There has been much debate on this topic across various disciplines, from geographers (Lynch, 1960) to neurophysiologists (O'Keefe & Nadel, 1978) and behavioural psychologists (Tolman, 1948). These researchers agree that what is computed is a complex map, which they refer to as a *cognitive* map (Tversky, 1993). A cognitive map is not simply a description of the layout of the environment. Among other things, it consists of one's emotional as well as one's physical experience of the environment. For instance, one feels that an outward journey often takes a longer time than an inward journey. It also depends on how one conceives a place as opposed to one's direct perception of it. Kuipers' (2000) Spatial Semantic Hierarchy model of a cognitive map, for example, consists of 4 complex layers: a control layer, a causal layer, a topological layer and a metrical layer. While this is the most detailed and complex model to date, it is still inadequate to capture the full complexity of a cognitive map.

Yet, the cognitive mapping process must begin from where perception ends. When an autonomous agent, be it a human or a robot, takes a step forward, it must somehow remember the spatial layout of things seen from the previous views. Otherwise, it would have no idea what is immediately behind it. Such an initial map of the environment, which is obtained directly from one's perception, is referred to as a *perceptual* map. The perceptual map then provides a basis from which one's cognitive map emerges. Much research has been conducted by psychologists to investigate what frame of reference is used by humans to compute a perceptual map (for recent work, see Mou, McNamara, Valiquette, & Rump, 2004; Wang & Spelke, 2000). There is little dispute that multiple different frames of reference are used in cognitive maps but for the perceptual map, the question arises as to whether it uses an egocentric frame of reference or an allocentric frame of reference. The former implies that the centre of the co-ordinate system is focused on the viewer; this requires constant updating as one moves. The latter implies that the centre is independent of the viewer and is thus more stable. In general, it was thought that a human's perceptual map utilizes an egocentric frame of reference and the information is then transferred to the cognitive

map which uses an allocentric frame of reference (Burgess, 2006). However, how exactly information is transferred between the two maps has not been explained - if it is simply copied without further processing then the maps are, mathematically speaking, equivalent.

Robotics researchers have developed some impressive algorithms to show how a spatial map of one's environment can be built directly from one's perception of it. They do not first compute an egocentric map and then convert it to an allocentric one. Rather, they build a precise metric map for the entire environment experienced. They refer to this process as simultaneous localization and mapping or, in short, SLAM. Their overriding concern is to enable the robot to know where it is. Consequently, their solution is to compute a precise metric map, whereby even the position of the robot in it is known. Humans, in contrast, do not remember their precise location in the cognitive map and they cannot tell the exact co-ordinates of objects in it (Passini, 1984).

So, how do humans compute a perceptual map? What is the solution discovered by nature? One of the earliest works on cognitive maps that pays attention to this problem is Yeap's (1988) computational theory of cognitive mapping. Yeap argued that two representations are most important: a MFIS (a memory for one's immediate surroundings) and an ASR (an absolute space representation). The former is the perceptual map as discussed above and the latter is the information extracted from the perceptual map to become part of one's cognitive map. Recently, Yeap (2011) extended his theory to provide an explanation as to how a perceptual map is computed.

This thesis provides an implementation of Yeap's theory of perceptual mapping using a mobile robot equipped with laser sensors. Chapter 2 describes the perceptual mapping problem in detail and outlines Yeap's theory. Chapter 3 describes the implementation of Yeap's theory and the results obtained and chapter 4 concludes the thesis with a summary and future directions.



## 2 The Perceptual Mapping Problem

The perceptual mapping problem is concerned with how humans compute a spatial layout of their immediate surroundings. Naturally, successive views of the environment need to be integrated to form a map - the details of such an approach have been much studied by roboticists, who refer to this problem as SLAM. This concept has dominated the field of autonomous robotics research for many years (Sünderhauf & Protzel, 2010). SLAM stands for Simultaneous Localization And Mapping and can be described as “the ability to place an autonomous vehicle at an unknown location in an unknown environment and then have it build a map, using only relative observations of the environment, and then to use this map simultaneously to navigate” (Dissanayake, Newman, Clark, Durrant-Whyte, & Csorba, 2001). Great strides have been made towards obtaining solutions for the problem, and the general approach is now well understood by researchers (Bailey & Durrant-Whyte, 2006).

SLAM solutions are characterized by extreme spatial accuracy, as it is typically argued that this is needed to perform correct navigation in an otherwise unknown environment (Sünderhauf & Protzel, 2010; Williams, Dissanayake, & Durrant-Whyte, 2002). In order to achieve such precision, probabilistic error correction has been the de-facto standard for many years (Thrun, 2003). SLAM has met with great success and the power of such algorithms has been clearly demonstrated: complex and large-scale environments have been mapped with great accuracy, and SLAM techniques have been used to successfully complete the DARPA Grand Challenge, enabling a computer-guided car to travel autonomously for over 200 kilometres through difficult terrain (Thrun, et al., 2007).

Yeap (2011) noted that the SLAM solution posed a major problem for computing humans’ perceptual map. Psychologists believe that information made explicit in a perceptual map will eventually be transferred into one’s cognitive map (Mou, et al., 2004; Wang & Spelke, 2000). However, the precision of the information contained in such a map is not compatible with what is made explicit in a cognitive map. Humans work with inaccurate and fragmented memories in their cognitive map (Tunstel, 1995). Note that this does not imply that the cognitive map does not have metric information. Humans can have a good sense of direction even after only a short exposure to a novel

environment (Ishikawa & Montello, 2006). They simply do not maintain an accurate metric map, and consequently they cannot “read off” the exact position of things (including their own position) from it.

Yeap (2011) recently developed a new approach for computing a human perceptual map. This chapter provides an overview of his theory. We begin, in Section 2.1, by highlighting some general questions that researchers often ask about cognitive maps and these are the questions that are also of interests to us. In section 2.2, a brief overview of Yeap’s computational theory of cognitive maps is presented (Yeap, 1988; Yeap & Jefferies, 1999). This provides the background of his work which leads to the development of his (2011) new perceptual theory. The latter is described in section 2.3.

## **2.1 General background on cognitive maps**

A cognitive map is commonly referred to as an internal spatial representation, used as a guide to travel by humans (Golledge, 1999). While it is safe to assume that the cognitive map is not necessarily a metric “map” in a cartographic sense, there is still much debate as to the exact nature of such a map.

Looking at the issue from a functional perspective, Scholkopf and Mallot (1995) argue that the basic computational purpose of a cognitive map is to allow one to perform specific kinds of behaviour, such as exploration and navigation. According to them, it must thus include information which enables place recognition, as well as finding spatial relations between these places. This data in turn is gathered from observations about the environment (Beeson, Modayil, & Kuipers, 2008). But in what form could such information be stored in the map?

A very important issue is accuracy. When moving through an environment, humans usually do not rely on exact measurements. Rather, they have a subjective impression of distance (Tunstel, 1995). A human might say an object is “a couple of metres away”, or even “it’s over there”. But where is over there? It becomes evident that humans can navigate their environment by using only very rough estimates. It has been suggested that cognitive maps can be constructed in several layers (Beeson, et al., 2008). A low-

level map would consist of raw geometric input, e.g. distances and orientations. From this map, a higher-level representation can then be synthesized (Yeap, 1988). In an indoor scenario, for example, a simple low-level map would consist of walls, whereas the high-level map could consist of rooms. This can be seen as an analogy to human categorizing: a human would move from one room to another, and not directly think of the walls constituting these rooms (Kuipers, 2000).

The question is of course: how would one go about separating an environment into disjoint spaces – which then form the high-level map? How can one recognize exits, the splitting points where one logical space ends and another starts (Jefferies, Baker, & Weng, 2008)? While this thesis will mostly focus on low-level geometry, some experiments will be conducted in order to see what kind of high-level maps can be generated, and how they could be used to travel.

The low-level map is often referred to as a perceptual map. One important question is how to combine subsequent views to form it. The question arises whether they are combined into a continuous spatial map (Yeap, 1988), or if they are merely remembered as independent snapshots, with only a logical connection between them (Scholkopf & Mallot, 1995). Assuming that views are spatially connected, how would one add new objects to the existing map, especially considering errors in perception?

When moving through an environment, humans are able to recognize and track familiar objects (Scholl, Pylyshyn, & Feldman, 2001). It seems thus logical that, when encountering a new object, it can simply be placed relative to other known objects which have been tracked over the last views. By employing this strategy, inaccurate perception is less of an issue because any global error in the view affects objects and reference together. Since the position of the reference target in the map is known, the new object can thus be positioned correctly – no matter how great the global error. Naturally this is possible only under the assumption that two successive views always overlap, and that they share common objects. If this is not the case, the individual becomes disoriented and makes mistakes in integrating a new view into the map. This can however be considered quite natural, as humans usually become disoriented when unable to visually track objects, for example when being blindfolded and moved to an

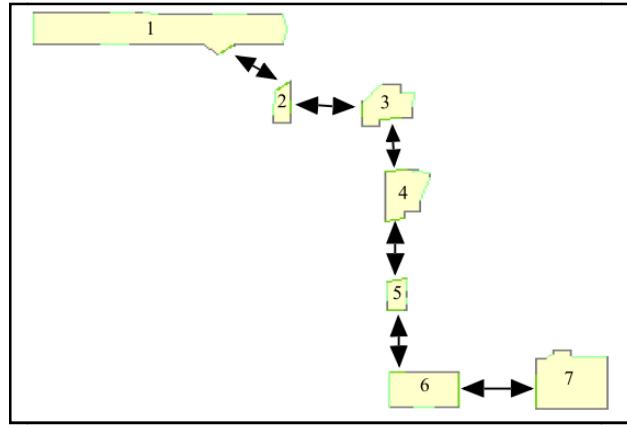
unknown environment (Wang & Spelke, 2000). As will be shown in Section 2.3, Yeap (2011) proposed the use of such a strategy to compute a perceptual map.

## **2.2 Yeap's theory of cognitive mapping**

Yeap (1988) proposed a computational theory of cognitive mapping whereby he argued that the cognitive mapping process must compute a representation of the local space that one is in and a representation of the spatial layout of one's immediate surroundings. The former is referred to as an Absolute Space Representation (ASR) and the latter is referred to as a Memory For one's Immediate Surroundings (MFIS). A network of ASRs thus becomes the basis for one's cognitive map. While the theory was meant to describe human cognitive mapping, it has always been tested using a mobile robot. Thus, without any loss of generality, we will, from here on, discuss the theory from a robot's perspective.

### **2.2.1 Representing local spaces**

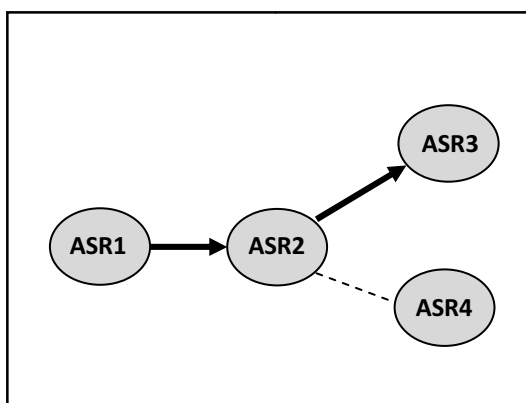
The idea of ASR computation is that one makes explicit a description of the local environment that has been visited. When a robot leaves the current local space and enters a different one, a new ASR is generated and a connection between them is formed. By this method a topological network of ASRs is built as the robot explores its environment - Figure 2.1 shows an example. Of special interest is the question of where one ASR ends and another begins. The robot must judge by itself and recognize when it has entered a new local space. The partition of an environment into these spaces can be quite subjective; a human might for example divide a building into separate rooms, and then use this network of rooms to navigate.



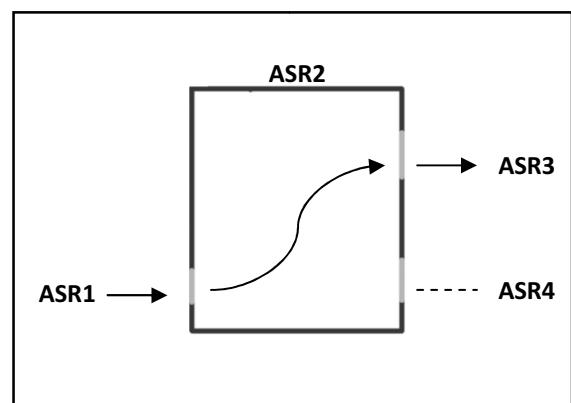
**Figure 2.1:** A network of ASRs (Jefferies & Yeap, 2001)

The part of the ASR boundary that leads to a different local space is called an exit, since the robot uses it to leave the current ASR and enter a new one. A key issue in the algorithm is the detection of such exits, because they are the connecting points of the network (Yeap, 1988). They uphold the spatial relation between ASRs, and thus a robot can only navigate between different local spaces if it is able to identify and cross them.

An ASR must thus contain spatial information about its exits. This is especially important for navigation: consider a robot that has already built a network of ASRs and now wants to travel back to a known location. In order to do so, it has to find a route across the network – this is easily accomplished by a graph search, as shown in Figure 2.2. But when trying to follow this route, the robot still has to navigate from one ASR to the next; thus it must be able to identify the exit which leads to the correct space. Figure 2.3 demonstrates this: in order to find a path from ASR1 to ASR3 the robot must know where the correct exit is located. For this reason, exit positions are part of the ASR.



**Figure 2.2:** A route in the ASR network



**Figure 2.3:** Navigating between exits inside an ASR

### 2.2.2 Remembering immediate surroundings

From what input can an ASR - and its exits - be generated though? Yeap argues that a second, more low-level representation of the environment is needed. He thus proposes the MFIS, or memory for immediate surroundings. It is a limited global memory containing the last few local spaces visited (Jefferies & Yeap, 2001). This MFIS consists of all the objects seen in these spaces in the form of a geometric map. Here the environment is represented by a two-dimensional map which, in turn, contains geometric surfaces that have been recognized by the robot (see for example Figure 2.4).

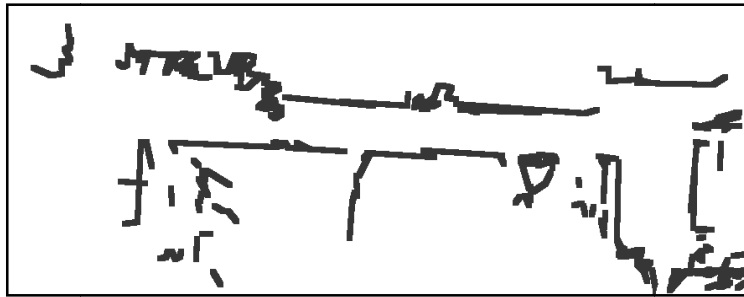


Figure 2.4: Surfaces inside the MFIS

This leads to the question of what frame of reference should be used for surfaces in the MFIS. Are they perceived relative to the individual (egocentric), or would they be seen relative to a different point in the world (allocentric)? Naturally the sensory input is egocentric since the robot scans for surfaces and notes their distance from itself. However would they also be stored in the map in this way? After all, the robot is moving and constantly changing position. If the environment is represented relative to it, then the whole map would have to be changed after every step. This seems quite counterintuitive, since it is the world which is considered fixed and the robot which is moving. A simpler solution would be a map with a fixed coordinate system representing the world, where the robot position is moving relative to it (Yeap, 1988). The MFIS is thus using an allocentric frame of reference, fixed to a certain point in the environment. Jeffries and Yeap (1999) argue that this point need not be chosen arbitrarily and that the entrance of the current ASR could be used as this reference. In this way, the robot would perceive its immediate surroundings relative to the local space it is currently in.

When moving, the robot recognizes new surfaces that it has not seen before. The MFIS is then updated by integrating them into the global map. It is grown after every step, since every movement could potentially yield new information about the environment. It is reasonable to believe that there are overlapping parts between successive views (Yeap, 1988) and thus common surfaces between them can be identified. These surfaces can be used as reference targets for updating the MFIS: with them it is possible to determine where to put a newly encountered surface in the map.

### **2.3 A new theory of perceptual mapping**

One significant problem with Yeap's theory of cognitive mapping is implementability. While Yeap provided significant arguments as to why these representations are computed, there are several problems related to how this is actually done. For example, it was never clearly defined what an ASR is or what the extent of the MFIS should be. If the older parts of the MFIS are simply removed and new parts are simply added, then the MFIS is really a representation of the whole environment visited but with the most recent parts displayed. Such a map would be no different from what robotics researchers compute for their mobile robots using SLAM. Yet, as discussed earlier, SLAM is inappropriate for describing the mapping process of humans.

Finally, this problem was solved and Yeap (2011) developed a solution which does not need constant updating and does not rely on an accurate computation of position information of surfaces/objects in view. The solution is based upon observing that one's view provides a good description of the local environment which one is about to explore. He used the example of wandering down a corridor. As one enters it, one's view down the corridor provides a good description of the local space that one is about to explore. As one moves inside of it, one does not need to update that description. However, at some point down the corridor (e.g. when one is about to move out of it), one needs to update the description with a new view.

One problem remains with this idea. That is, how do we add a new view into the map if we do not constantly update the map with successive views? The solution came from asking how humans recognise where they are – they recognise familiar objects in the environment. Using the corridor example again, how does one know where one is

when one is half-way down the corridor? If one is able to recognise that the exit at the end is the same exit which has been observed earlier and remembered in the map, then one would be able to triangulate one's position in the map. This method requires tracking of known objects across successive views, which is a task that humans are good at. If no known objects are to be found in the next view, it is time to update the map with information from the current view. New surfaces in the current view are added to the map using the same triangulation method.

This way of updating removes the need for error correction inside the map. A mapping error is generally considered to be the insertion of objects at a wrong place in the map. If we assume that our algorithm is working as intended, such errors can only come in the form of wrong measurements from the robot's sensors. The typical robot for which the algorithm is designed uses just two sensors: a laser rangefinder to detect surfaces, and an odometer to keep track of robot movement. Errors in the former only affect individual surfaces and are thus not critical to overall mapping success. MFIS updating is not impeded by a small number of faulty surfaces and can safely continue as long as at least one reference target can still be identified. The other source of errors, a wrongly measured robot movement, is far more dangerous. Traditional robot mapping algorithms often depend on this information – it is used as a mathematical transform which projects the newly seen surfaces into the coordinate space of the map. While the idea might seem intuitive, it is problematic because even the slightest error in the measured movement can have disastrous consequences. A wrongly measured rotation, for example, will offset all surfaces on the map together, and objects far away from the robot will drastically change position. But worse still, in the next step another such error is added to the map. Eventually the accumulated error becomes so great that the entire map is rendered useless. Consider Figure 2.5: the surface shown has been rotated by the small error  $\epsilon_1$ . The mapping process is continued, and in the next step it is rotated again by another small error,  $\epsilon_2$ . But since the errors accumulate, the global error between the first and last steps becomes very large.



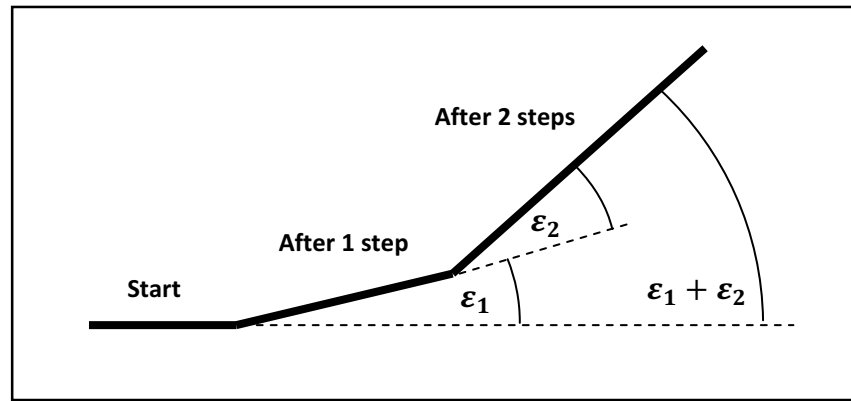


Figure 2.5: Error propagation during mapping

Here lies the key advantage of Yeap's approach: since surfaces are inserted relative to a reference target, the measured robot movement is not needed at all. The MFIS can therefore be updated without introducing these errors. As a side effect, the robot keeps no account of where exactly it has travelled; it orients itself based on its surroundings instead. The downside to this is obviously the strong dependence on an accurate reference target. In the case that no common surfaces between two views can be found, the robot would be completely lost. This however happens very rarely and only in special situations, as experiments in chapter 3 will show.

Now that a global map with all surfaces exists, the geometry in the MFIS can be used to construct ASRs. Since ASRs are connected via exits, these have to be identified before the network can be built. But how can the robot detect where one local space ends and another begins? Yeap and Jeffries (1999) define an exit as gap in the boundary of a local space. They argue that these boundaries are the surfaces which constitute the overall shape of the space, and that any passage between them is automatically an exit. The robot must recognize this whenever it reaches such a location. Yeap and Jeffries further argue that depth information should be used to determine which surfaces are boundaries and which are not, but this is beyond the scope of this thesis. Since all our experiments have been performed in indoor environments we can greatly simplify the problem: if we assume that local spaces represent rooms in such environments, then exits will represent doorways. Since a doorway is an opening in the wall of a room, it cannot be wider than the room itself. Consequently, an exit must always be narrower than the local spaces which are connected by it. Following this assumption, the robot can easily determine how

narrow its current surroundings are – for example by measuring the distance of surfaces to its immediate left and right, similar to a human using peripheral vision. Whenever such a narrow passage is crossed, the robot can thus deduce it has crossed an exit. Once that happens, the spatial connection between the individual ASRs can be defined and the network can be constructed.

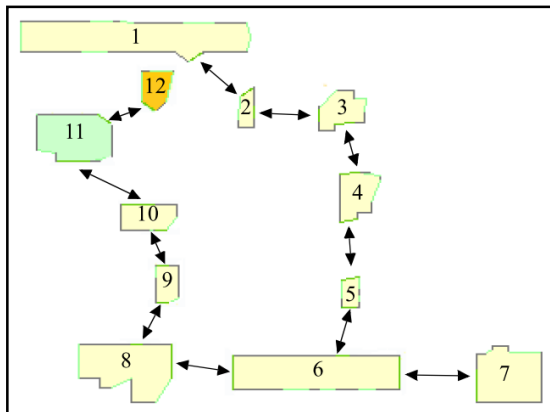
Using ASRs and the MFIS, a mobile robot possesses both a rich spatial map and a topological network of places. By combining this information it can then navigate its surroundings and recognize places it has already visited, instead of creating new ASRs for them (Jefferies & Yeap, 2001). Here it does not matter if the sensors are inaccurate, because only a rough spatial connection between the ASRs is remembered. If there are indeed errors in perception, resulting in skewed geometry, these errors will not be propagated for the rest of the journey. After all, once a new ASR has been entered its geometry is not directly connected with surfaces from the previous space – there is only the logical connection in the network. The layout of every new ASR is instead built from scratch, resetting any propagated errors.

In this manner some of the traditional issues in robot cognition can be circumvented. The loop closing problem, for example, is considered a typical challenge to robotic mapping algorithms (Thrun, 2003). It describes a situation in which a robot, after having travelled some distance, comes back to an area previously visited. If one uses a precise metric map, its ends would then need to connect. But if only the slightest sensor error is encountered, this will not be the case and the robot will get lost. Figure 2.6 below shows a metric map of a cyclic environment which has been traversed twice. From the picture one can see that the robot suffered errors in sensory input, probably rotational, and thus mapped the same geometry twice.

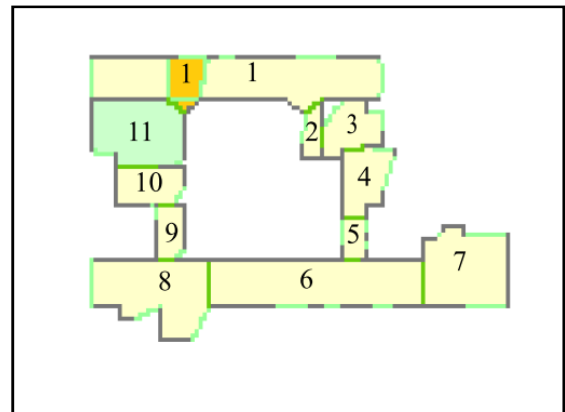


**Figure 2.6:** Errors in a cyclic environment (Thrun, 2003)

Combining ASR and MFIS information can solve this problem, since it enables the algorithm to recognize local spaces that have already been visited. When trying to build a new ASR for such a place, the robot can see from the MFIS that its position overlaps with another ASR, which is already known, and simply connect them together. Consider the images below: they show the ASR network and corresponding MFIS of 12 local spaces traversed. When coming back to the first space, a new ASR-12 is created. However from the MFIS it is clearly evident that ASR-12 is part of ASR-1. Now ASR-1 replaces ASR-12 and it is connected to ASR-12, closing the loop.



**Figure 2.7:** ASRs after traversing a circuitous route (Jefferies & Yeap, 2001)



**Figure 2.8:** MFIS after traversing a circuitous route (Jefferies & Yeap, 2001)

In this manner, the robot can solve the problem by simply reasoning that ASR1 and ASR12 are one and the same. Figure 2.8 shows the MFIS perfectly connecting at the end of the loop; here it is a coincidence and actually not a requirement for the algorithm to work. In fact, it is highly probable that the ends do not connect so accurately, since the MFIS is a global map without error correction. However, even if the ASRs in question have been displaced because of errors, they will still be in rough

vicinity of each other. From this the robot can then deduce that they are one and the same, and connect the loop in the ASR network. It must however be said that a “rough vicinity” is only given if the errors encountered are sufficiently small. In a case where the ends of the loop are not even remotely close to each other, there need to be additional methods for ASR identification. The robot could for example compare their size and shape. As can be seen in Figure 2.8, ASR12 fits smoothly into ASR1, so it seems probable that they represent indeed the same local space.

Maintaining ASRs and MFIS at the same can therefore be expected to produce a useful layout of the environment, even without error correction. Now that the functionality of the algorithm has been described, we turn towards its implementation.

## 2.4 Conclusion

A cognitive map is a complex representation that holds one's total knowledge about the environment. For humans, this includes one's conceptual view in addition to the perceptual view of the environment experienced. The "map" computed from one's perceptual view is the perceptual map or what Yeap (1988) referred to as an MFIS, a memory for one's immediate surroundings. Furthermore, Yeap (1988) argued that a network of local spaces or ASRs (absolute space representations) is then abstracted from one's perceptual map. Such a network provides an initial abstraction of what is perceived. Until recently, how the MFIS and ASRs are computed remained unclear.

Yeap (2011) provided an algorithm for computing an MFIS. In an abstract form, the algorithm is described as follows:

1. Execute movement instruction and get new view
2. Compare the current view with the previous and identify reference objects
3. If the new view contains objects that were not seen before, determine their position relative to the references, and add them to the MFIS. Go to step 1.

### 3 Implementations

This chapter describes the implementation of Yeap's perceptual theory on a mobile robot. The goal is to investigate whether the map produced is useful. This is judged subjectively by looking at the shape of the maps produced for different environments and by computing a network of ASRs for the robot to find its way home. If the maps maintain the shape of the environment traversed and also allow the robot to perform basic navigation tasks, then the implementation is claimed to be successful. However, Section 3.1 begins with a straightforward implementation of the standard transformation approach for robot mapping. This will allow us to have some comparison of the new approach with the standard approach. Section 3.2 describes the implementation of Yeap's algorithm to produce the MFIS and ASRs. Section 3.3 shows how these representations are used to find one's home.

All of these implementations are designed for a mobile robot which uses a two-dimensional laser rangefinder. Laser data is easy to work with since the sensor can quickly generate a fairly accurate outline of the robot's surroundings - thus not much effort is needed to capture the environment in spatial terms. The disadvantage with this, however, is an inability to perform complex object recognition and feature detection techniques, since laser data only consists of positions and distance measures. Naturally the sensor is thus only able to generate a two-dimensional image which shows the distance of objects from to the robot.

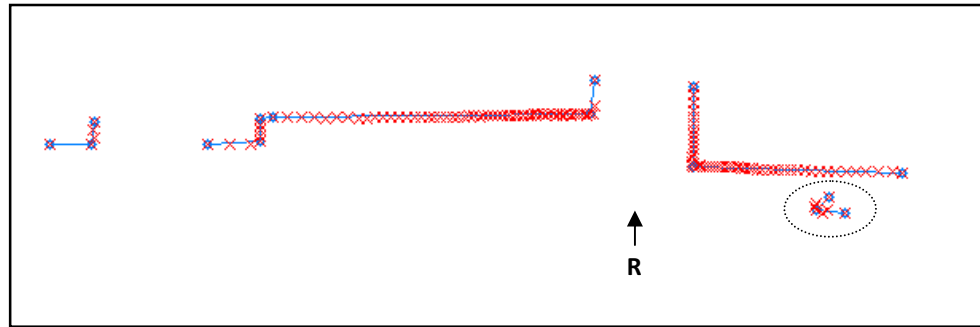
#### 3.1 Traditional algorithm

This section outlines the design of a traditional, error-correcting algorithm and describes the steps necessary for its implementation.

##### 3.1.1 Identifying surfaces

The only kinds of objects easily recognized from laser data are geometrical surfaces, such as walls limiting a room; therefore they will constitute the robot's representation of its environment. The simplest form of surface is of course a straight line, and thus it

will be used as basic unit of this implementation. Naturally, before the mapping process itself can begin, these surfaces must be synthesized from the laser input first.



**Figure 3.1:** Recognizing surfaces

**Legend:**

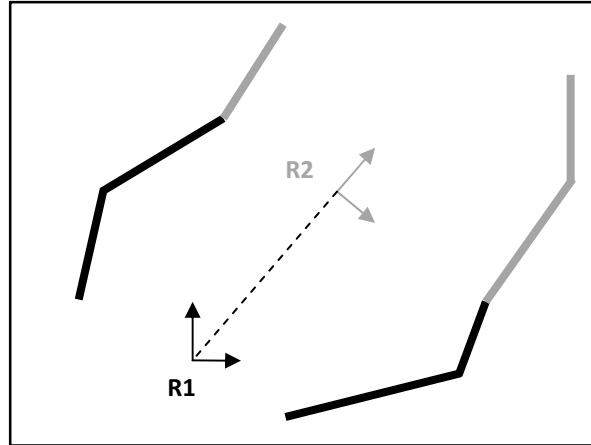
<b>R</b>	Robot position
<b>x</b>	Point detected by laser range finder
<b>—</b>	Surface to be used by the algorithm

Figure 3.1 shows a sample laser scan, grouped into surfaces. It can be seen that this generalization works well with long surfaces, which in the picture are representing walls, and not so much with smaller objects – see for example the encircled surfaces to the right. In this particular example, the sensor has scanned the legs of a chair. This however is not apparent in the picture; the scan only shows a cluster of points closely huddled together. It is clearly not enough information to identify the object, and for this reason it will simply be treated as debris: an obstacle of arbitrary form, to be avoided by the robot. Longer surfaces can be considered more important, since they typically are the defining geometrical features of the environment in which our experiments are carried out. However, in this implementation all surfaces detected are entered into the map. If it becomes necessary later, they can then be interpreted further.

### 3.1.2 Integrating views by transformation

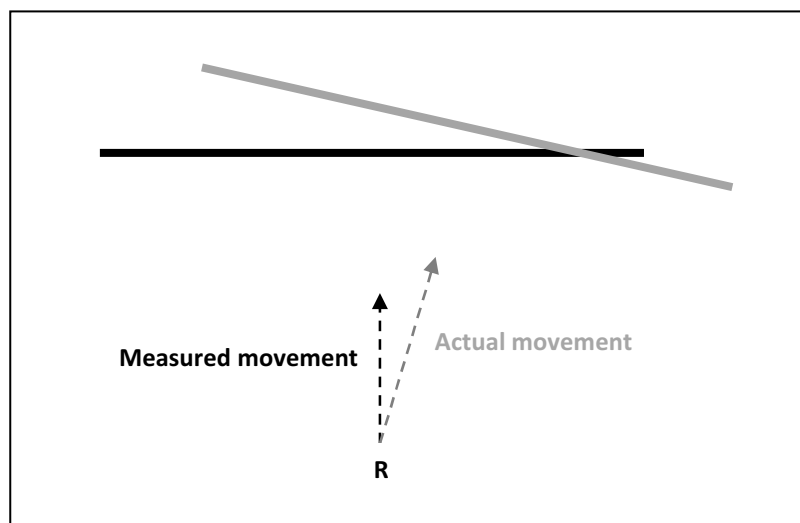
In order to generate a continuous map, the algorithm has to combine successive views of the environment. For this traditional metric map a simple coordinate transformation is used: the robot turns, moves, and then scans its environment. From its odometer, the robot knows where it has travelled; the map in memory is then rotated and

translated so it fits on top of the current view. Such an approach seems intuitive: we move to a new place and then offset everything we see in the map by the distance and direction travelled. Theoretically all geometry should then fit together, as shown in Figure 3.2, and the map would be precise as a result.



**Figure 3.2:** Coordinate transformation between two views

The problem is of course: the surfaces do not fit together in practice. How would the robot know exactly how far it has travelled? What if its odometer is not precise? What if it slips, or travels on uneven surfaces? It is easy to see that this measure is not accurate; and as a result, new geometry will be placed in the wrong location, thereby quickly rendering the map useless. Consider Figure 3.3 for example: if there is only a slight error in estimating the distance and angle moved, then a new surface would be placed at a wrong position.



**Figure 3.3:** Bad surface placement

From this, one can deduce that error correction methods are needed which negate such inaccuracies and - by doing so - enable the robot to fit one view on top of another correctly. In chapter 2 it has been argued that this reasoning might be flawed and that human-like cognition may not necessarily use such correction. However, since it has been an integral part of many approaches in the past (Thrun, 2003), it will be used for the traditional, metric mapping implementation described here.

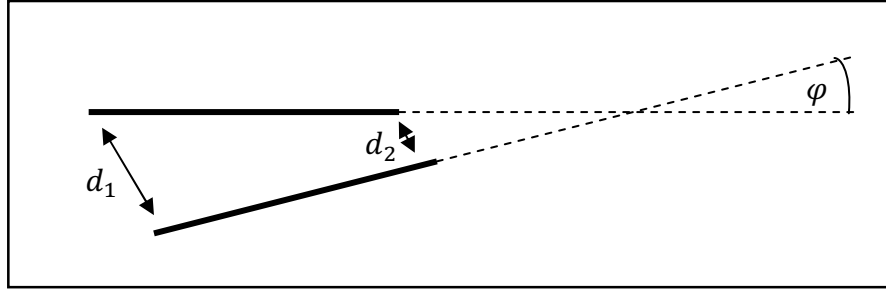
### 3.1.3 A simple error correction method

In traditional robot mapping algorithms, very complex methods are often used in order to correct errors, such as statistical optimization with Kalman filters (Thrun, 2003). However, since the traditional algorithm described here is not the main focus of this thesis, a relatively simple solution has been implemented. When exploring its environment, at every step the robot turns a certain angle and then moves a certain distance. The measured values for both variables are affected by movement errors, and as such the correct values (the actual angle and distance travelled) have to be determined. The goal here is to find a positioning where the detected surfaces of the current view fit as closely as possible on top of the previous view. This would then be the optimal positioning for updating the map. In order to find this optimal solution a simple heuristic search has been implemented. Heuristic optimization can be defined as a technique which iteratively tries to improve a candidate solution, while little or no knowledge about the actual problem is required (Reeves, 1993). Thus, without needing to analyze exactly how the errors are generated, we can conveniently use the technique to correct them. The candidate solution here is the imprecise measurement obtained from the odometer, and the heuristic is used to optimize it. For this, the issue has to be formulated as a search problem:

Let  $\theta$  be the angle turned and  $x$  the distance which the robot thinks it has travelled. They form a coordinate transformation  $T(\theta, x)$ , leading from one view to the next. As a result of this movement, let  $\bar{\epsilon}(\theta, x)$  be the error observed when integrating the current view into the map. To correct the error, the heuristic will try to minimize this function.  $\bar{\epsilon}$  describes how accurately the map fits on top of the current view of the environment, formulated as the average error of all matching surfaces. A pair of



matching surfaces, as shown in Figure 3.4, is characterized by a low endpoint distance and a low difference in orientation:



**Figure 3.4:** Measuring a pair of matching surfaces

A greater error thus results from a greater distance and rotation. Let  $\varepsilon$  be the matching error between surfaces  $S_1$  and  $S_2$ :

$$\varepsilon(S_1, S_2) = d_1(S_1, S_2) \cdot d_2(S_1, S_2) \cdot \varphi(S_1, S_2)$$

By multiplying the values, the error  $\varepsilon$  rises with increasing difference. Here it has to be said that the actual formula used in the implementation is slightly more complex, in order to account for special cases (for example when  $\varphi$  is exactly 0). The overall result however is similar: the matching error increases with surface distance and rotation.

To calculate the average error, all surfaces in the view have to be compared: let  $V_{Curr}$  be the current view, containing all surfaces the robot is seeing after moving, and  $V_{Prev}$  the surfaces from the previous step which are taken from the map. Rotated by  $\theta$  and translated by  $x$ ,  $V_{Prev}$  is transformed to  $V'_{Prev}$ :

$$V'_{Prev} = T(\theta, x) \cdot V_{Prev}$$

Between these views, the surfaces are compared individually. For each surface  $S_{Curr} \in V_{Curr}$  the closest matching surface  $S'_{Prev} \in V'_{Prev}$  is picked; that is, for which the error  $\varepsilon(S_{Curr}, S'_{Prev})$  is minimal. For the sake of simplicity we assume here that every surface can only match a single partner from another view. From all resulting pairs  $(S_{Curr}, S'_{Prev})$  the average surface-matching error  $\bar{\varepsilon}$  can then be calculated, with  $n$  being the number of matches:

$$\bar{\varepsilon} = \frac{\sum \varepsilon(S_{Curr}, S'_{Prev})}{n}$$

We are thus able to determine the overall error  $\bar{\epsilon}$ . The goal is now to correct the transformation, or rather to find the optimal combination of  $\theta$  and  $x$  for which  $\bar{\epsilon}(\theta, x)$  is minimal.

The heuristic search technique employed is a simple Annealing search: starting with the values measured by the robot,  $\theta$  and  $x$  are changed by adding and subtracting random numbers. If the resulting error  $\bar{\epsilon}$  is less, then this solution replaces the previous one. Again the variables are modified, albeit by an increasingly smaller margin. Since most searches will start near an optimum (views are almost matching), the solution is likely to converge towards it. While such a search technique can be considered quite basic, it is also easy to implement and thus has been used for this simple algorithm.

Now that the optimal combination ( $\theta_{opt}$  and  $x_{opt}$ ) with the smallest error has been found, it is used to transform the map and estimate the robot's movement. Finally, the new surfaces from  $V_{Curr}$  can be added. With the map now updated, the robot can continue its travel. But how strong is this approach? Can it really account for all errors, and will it generate a clean and accurate map? To find out, the algorithm must be tested.

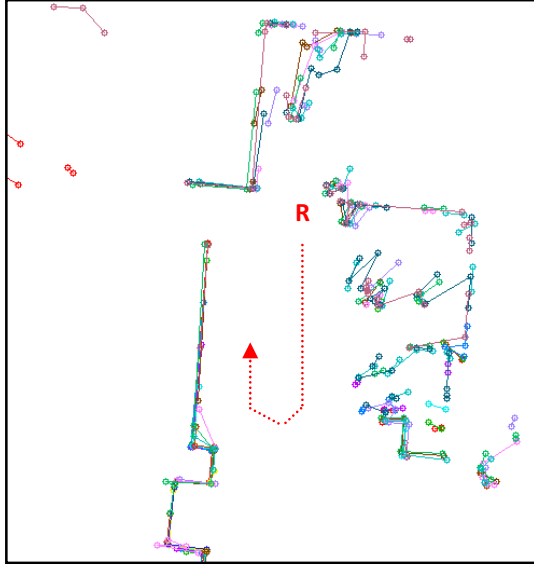
### 3.1.4 Experiments

In order to see what maps it produces, the algorithm has been tested on two different laboratory robots which are instructed to map their environment. Both are Pioneer 3-DX research robots equipped with a SICK laser rangefinder; one however is older and has been found to move with significantly lower accuracy. Because of this difference in precision, the robots are henceforth referred to as "good" and "bad". Their movement is controlled by user input but the mapping process is completely automatic. Also they have no prior knowledge about the environment or their own position.

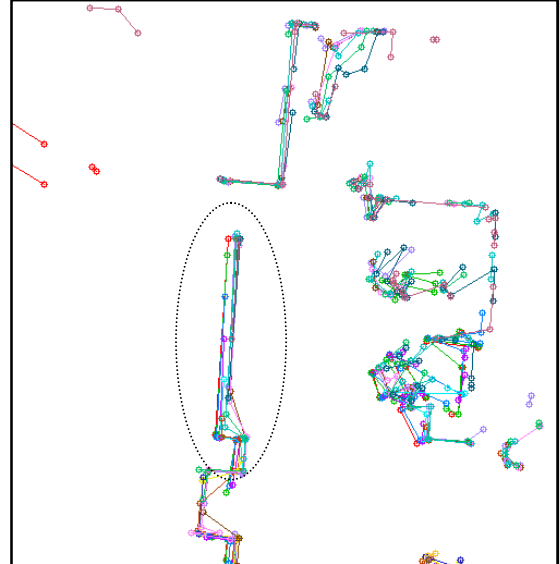
#### A small environment

In the first experiment, the robots have been run through a comparatively small office space. The route travelled is approx. 6 meters long and both robots have traversed it in

8 movement steps. A small environment has been chosen in order to observe the placement of individual surfaces – if the algorithm performs well, then surfaces which are seen several times can be expected to overlap. If it does not, however, then identical surfaces will be placed in different locations, duplicating them on the map.



**Figure 3.5:** Good robot in a small environment



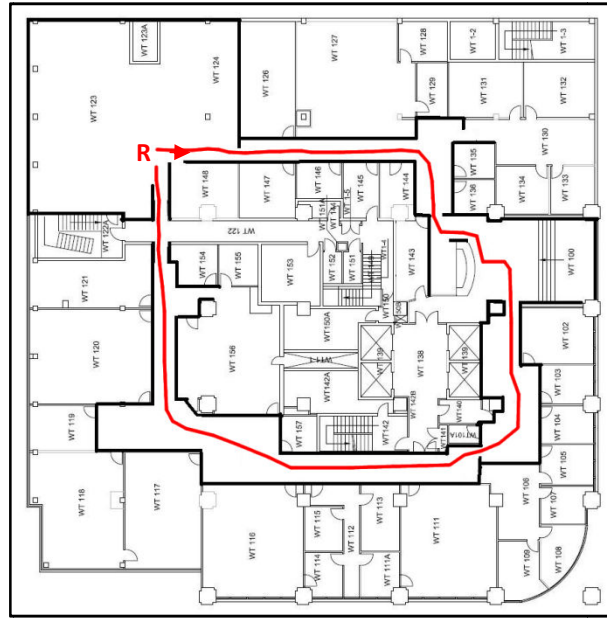
**Figure 3.6:** Bad robot in a small environment

The above figures show the resulting maps which have been calculated by the robots, with the travelled route highlighted in the left picture. The surfaces are marked with a different colour for each view, thus it is easy to spot placement errors. In both figures, most surfaces are placed fairly accurately. Especially the long surfaces mapped by the good robot seem very precise. There are, however, some displaced lines which have been added as duplicate surfaces, such as the ones encircled in Figure 3.6.

In Figure 3.6, one can also see several surfaces to the right that seemingly do not match at all, but occupy the same space. This is a side effect of the surface recognition technique described earlier: small objects cannot be properly represented by lines, and thus they are assigned short surfaces which can change orientation when viewed from a different angle. In this example, the legs of several tables have been scanned. Such debris on the map however is not our primary concern; much more important are long and straight surfaces. Since these have been mapped quite precisely, it can still be said that the algorithm has performed well. Both images look very similar, and even the bad robot has placed the long surfaces accurately, thus the error correction seems to work well for such a small-scale map.

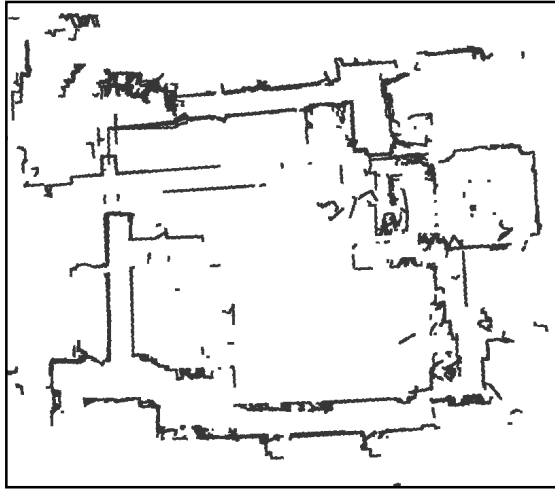
## A large environment

Errors from sensory input can be small at times, and barely noticeable. However, big problems can still arise from error propagation: since at every step a new transformation is added to the map, the errors accumulate with the distance travelled. In order to see the impact of small errors on a bigger map, the robots have been steered through a considerably larger office environment, shown in Figure 3.7.

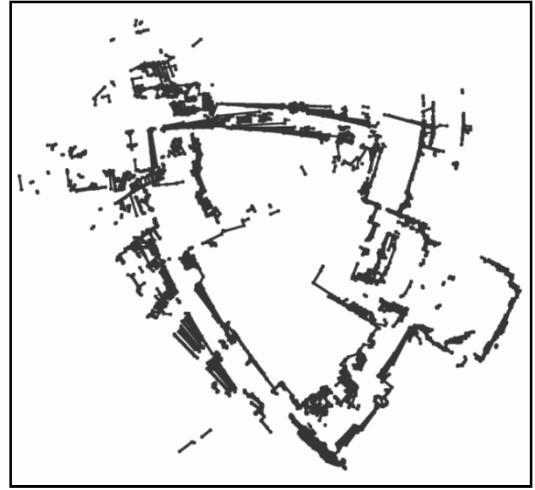


**Figure 3.7:** Floor plan of the large office environment

The route travelled in this experiment is approx. 90 metres long, and both robots have traversed it in roughly 100 movement steps. It should also be noted that the route is circular, meaning the robots arrived at the same place from which they started. Thus, if high precision is achieved, it can be expected that the ends of the generated map will connect.



**Figure 3.8:** Good robot in a large environment



**Figure 3.9:** Bad robot in a large environment

As can be seen from the figures above, the maps produced by the robots differ significantly. While neither map connects accurately at the end, the overall shape of the map in the left picture is relatively good. Even though the loop is not closed, the map still seems usable over long stretches, and there is a clear resemblance to the floor plan. Not so with the bad robot: the shape of its map is extremely skewed and the route takes the form of a triangle instead of roughly a rectangle. This of course can be attributed to extreme error propagation - by the time the robot has arrived at its starting position, the rotational error seems to be almost  $90^\circ$ . It is obvious that such a map is unsuitable for navigation, since the robot's estimate of its global position is completely wrong. Also the map is cluttered with duplicated of the same surfaces, completely blocking the robot's path at times.

It becomes thus apparent that the correction method described can only handle very small errors, and even then does not achieve a perfect result. When confronted with greater errors from the bad robot, the map is rendered useless very quickly.

### 3.1.5 Discussion

The experiments show that the maps created by the good robot are relatively accurate; however this is only the case because its sensors are quite precise. From the results obtained by the bad robot it becomes apparent that, even though the movement error has been mitigated, the maps produced are not accurate at all. Worse still: at many points there are surface duplicates, often deviating by a great distance.

Finally, even with error correction, the map of a larger environment is skewed and the ends of the loop do not connect. Of course, the implemented optimization is quite crude and greater accuracy could be obtained by error correction techniques which are more powerful. With them it might be possible to generate highly accurate metric maps, but is it really necessary to use them? Do we really need to go to such lengths in order to achieve maximum precision? Or can we achieve similar – or better – results without error correction, using a human-inspired algorithm?

This implementation was done as an initial exercise to see how well a simple robot mapping strategy would work. It turns out that the algorithm works well for a new robot and not so well for an older robot. This is interesting and suggests that an older robot should be used to test the new algorithm in the next chapter, since an important claim of the cognitive mapping approach is that a reasonable map can be computed without error handling. As such, we have to ensure that the robot does produce many errors in its sensing.

## 3.2 Cognitive mapping algorithm

This section describes the implementation of the MFIS (see Section 3.2.1) and ASR (see Section 3.2.2) using the new algorithm described in Chapter 2. Section 3.2.3 shows the results obtained and Section 3.2.4 concludes with a brief discussion. Similar to the traditional approach, the robot has to rely on geometric surfaces, since they are the only objects which can be properly recognized from the laser input.

### 3.2.1 Integrating surfaces into the MFIS

The MFIS is a global metric map which contains all recognized surfaces and exits. As such it has to be grown and updated when new data is available, that is whenever the robot has recognized new surfaces. A “new” surface here describes one that has not been seen before and does not yet exist in the map. Every time the robot moves it can potentially find such objects: the platform might move into an unexplored area or turn and see its surroundings from a different angle. According to the theory, such an update does not necessarily need to happen after every step, but for the implementation described here, all new surfaces will immediately be added to the MFIS, so as to not miss any information. The robot, unlike humans, can remember everything it sees.

The robot starts with the first view of its environment – a collection of surfaces. Then it turns and moves, and after this step scans its surroundings again. This is similar to the traditional approach, but the key difference is the way in which the new information is processed. For the MFIS, common surfaces are tracked across subsequent views and, based on them, newly seen surfaces are added.

#### Identifying common surfaces

The first step is to compare the current view of the environment with the previous one (which already exists in the map). Common surfaces can be recognized by using the same matching as described in section 3.1: surfaces that are roughly in the same spot and have a similar orientation are considered matching. One could thus argue that they are one and the same, just seen from a different viewpoint. This is the key to

updating: why keep two records of the same object? Since the surface already exists in the MFIS, nothing has to be done and the newly detected one can simply be ignored. This fits with the concept of a stable world: old information in memory is not updated, even if the newer surface might be more precise. Consider a human: when we track an object across several views, we do not update its shape or position unless it has significantly changed. (Intraub, 1997). Why then should the robot?

One issue remains though: the algorithm has to recognize matching surfaces in order to track them. Since they are represented as two-dimensional lines they carry little information that would make them unique. The only way to identify a match is by spatial proximity and similar orientation – but how can this proximity be measured though? The absolute positioning of surfaces is unknown after all. The robot could still use its odometer to determine where it has moved between two views, but this would re-introduce the errors found in the traditional approach!

The problem can be solved by taking the middle way: by using a rough estimate of how far it turned and travelled between them, the robot can project two views on top of each other. Similarly to this, humans usually know roughly where they are moving. Of course this could also be called a transformation, however the major difference to the traditional approach is that it is only used to identify common objects, but not for the placement of new surfaces. Once the matching surfaces are recognized the transformation can be discarded - and the new surfaces are processed without it. This matching technique can thus be simply considered a crutch for the robot, since it is the only way surfaces can be tracked with the laser sensor. In future implementations a camera could be used in order to reliably identify unique objects, removing the need for such a transformation completely. Potential errors from this transformation will only hamper the algorithm if matching surfaces cannot be detected at all between two views. As long as at least one such common object is identified though, no error will be added to the map. Consider Figure 3.10: even though the surfaces are not exactly overlapping, because of their small distance and similar orientation the robot can conclude that they are the same.



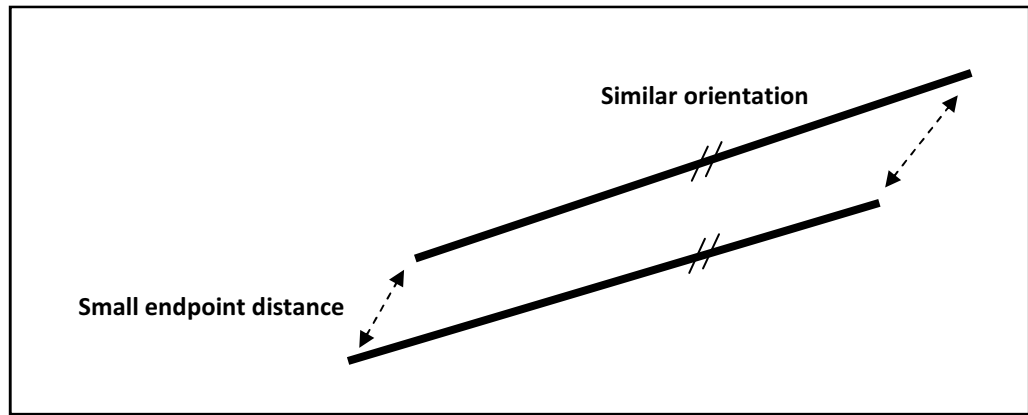


Figure 3.10: Matching surfaces

### Adding new surfaces

Surfaces which match old ones are thus not added into the MFIS. They do however have a very special use: since they have been tracked across several views, the robot can use them as references to orient itself. This holds true especially for adding new surfaces: consider a completely new surface which has not been seen before. What position does it occupy? How does it relate to our frame of reference, the coordinate system? Since the robot has been moving, it does not know its accurate position in metric terms. The only things it has kept track of are the common surfaces, and thus they can be used for adding new geometry.

The figures below demonstrate the concept. Figure 3.11 shows the current view of a robot's environment and all surfaces in it. Figure 3.12 shows the MFIS after updating. The algorithm has identified a common surface (between the view and the MFIS) and – based on this information - inserted a new one into the MFIS. In order to do this, the distance and angle relative to this reference target were determined and, by transferring the measurements to the MFIS, the new surface could be placed correctly.

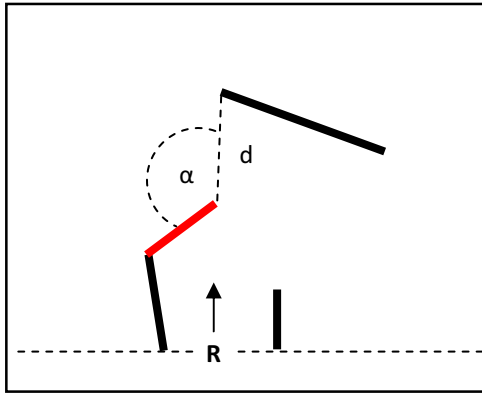


Figure 3.11: Surfaces in the current view

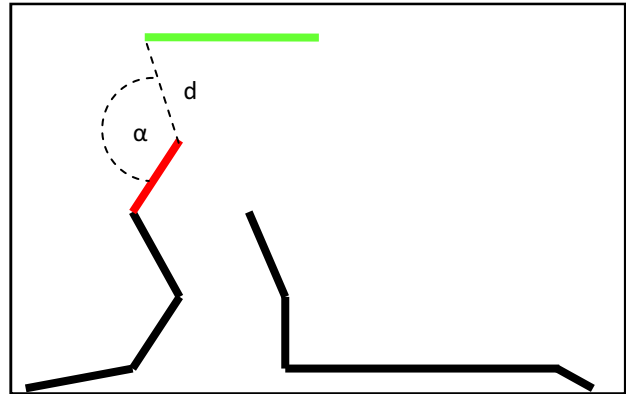





Figure 3.12: MFIS when inserting a new surface

**Legend:**

 Surfaces in the view / MFIS	 New surface inserted into MFIS
 Common surface (reference target)	<b>R</b> Robot position

In this way new surfaces can be added to the MFIS without having to use a mathematical transformation that emulates robot movement. Section 3.1 has found this transformation to be the major source of errors when updating, and by not using it such problems are avoided from the start. As a result no correction is needed.

Greatly simplified, the implemented algorithm performs these major steps:

1. Turn and move (controlled by the user).
2. Do a laser scan.
3. Generate surfaces from the laser scan.
4. Save all surfaces into the MFIS (first view).
5. Turn, move and scan again, generate new surfaces.
6. Identify common surfaces between current and previous view. From these surfaces, the closest-matching pair will be the reference target for inserting new surfaces.
7. Place new surfaces into MFIS, relative to the reference target.
8. Repeat from 5, until stopped by the user.

Now that the robot possesses a continuously updated global map, the ASR network can be created.

### 3.2.2 Building a network of ASRs

From the surfaces contained in the MFIS the network of ASRs is generated. An ASR describes the local space in which the robot currently resides. As described by Yeap's theory of cognitive mapping, it contains the surfaces of the current space along with exit information. An exit is a passage in the boundary which enables the robot to move to the next space and is thus the point where ASRs are connected to each other. In order to know where one ASR starts and the other ends, this point must naturally be recognized.

Once the exits have been identified, some unique information about the ASRs must be gathered. Since a local space is defined by the objects inside of it, an ASR can be defined by the surfaces inside of it. Subsequently, the algorithm must take all surfaces from the MFIS and decide which ASRs they belong to. Thus there are two major tasks when building ASRs: one has to find the exits which separate them, and distribute the recognized surfaces among them.

#### Detecting exits

An exit can be considered a gap in the boundary between two spaces. When the robot passes such a gap it enters a different local space, for which a new ASR must be generated. For this implementation it is assumed that an exit will take the form of a relatively narrow passage between two wide spaces. Consider Figure 3.13 below: it seems obvious to the human eye that the space on the left should be partitioned into two. The one on the right however does not have a narrow passage and, for this reason, no partition seems necessary.

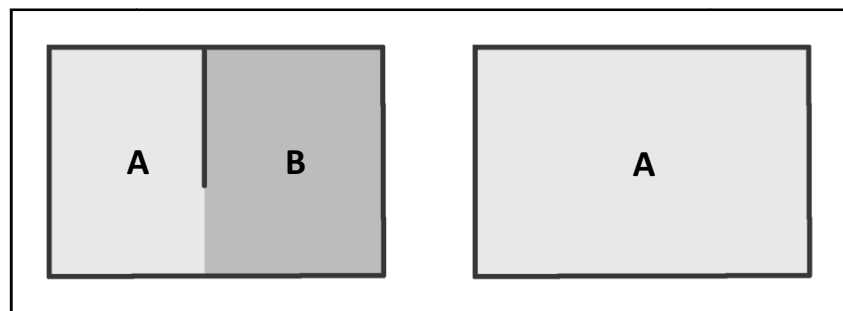
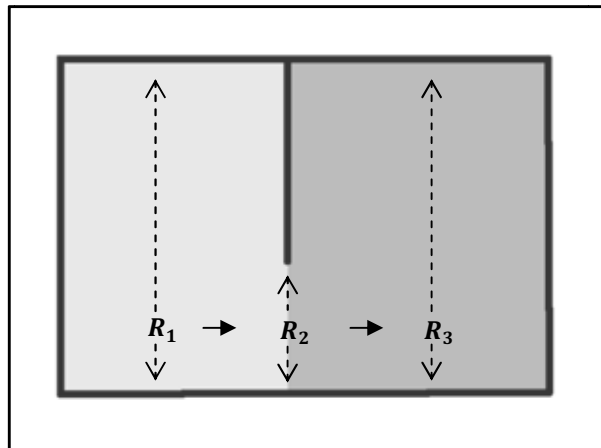


Figure 3.13: Separating spaces with a narrow passage

It can thus be assumed that an exit is located at a local minimum of “wideness” in the robot’s immediate surroundings. When following this notion it becomes quite easy to detect such passages: as the robot travels it keeps a measure of this wideness, which can be as simple as the distance of geometry to its left and right. When travelling it will notice a local minimum in the function of this measure and thus deduce it has crossed an exit. An example can be seen in Figure 3.14 below: the robot starts in a wide space, travels through a narrow passage and arrives in a wide space again. Because of this, it will reason that it has crossed an exit.



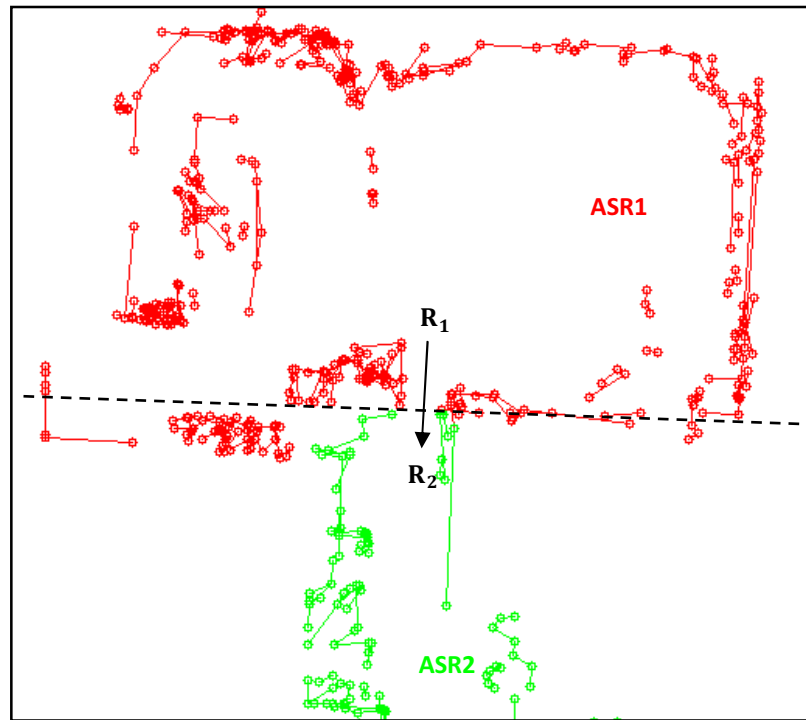
**Figure 3.14:** Wideness when crossing an exit

The obvious drawback of this method is that exits can only be recognized after they have been crossed. There is evidence to suggest that humans can identify exits long before crossing, and actually use them as targets for navigation (Yeap & Jefferies, 1999). However, the method is very simple to implement and, as later experiments will show, also quite reliable in finding exits on the robot’s route, and for these reasons it has been chosen for this implementation.

### Separating the local spaces

After an exit has been crossed, the robot has entered a new local space. Subsequently, a new ASR is generated and connected to the previous one. However, if we think back to the MFIS, which surfaces belong to which ASR? Intuitively, newly seen surfaces would be in the new ASR and old ones in the previous. But this ignores the fact that

some surfaces from the new local space were already visible before the robot entered it. Consider the figure below:



**Figure 3.15:** Surfaces seen across the exit

The picture shows the surfaces of two ASRs, separated by an exit. It is apparent that, at position  $R_1$ , the robot can already see some of the surfaces which will later form ASR2. When arriving at position  $R_2$  however, after crossing the exit, it has to transfer the surfaces behind the exit to the new ASR. But how can it decide what is “behind”? From the image it is clear that a simple cut along the infinite line defined by the exit is not a good idea: some surfaces which should belong to ASR1 are behind this line.

In order to solve this problem, the robot must employ logical reasoning. If a surface in ASR2 can be seen from inside ASR1, then it must be visible through the exit. Subsequently, the line of sight from  $R_1$  to any surface in ASR2 must intersect the exit passage. Conversely, if the line of sight to a surface does not directly cross an exit, it belongs to the same local space in which the robot resides.

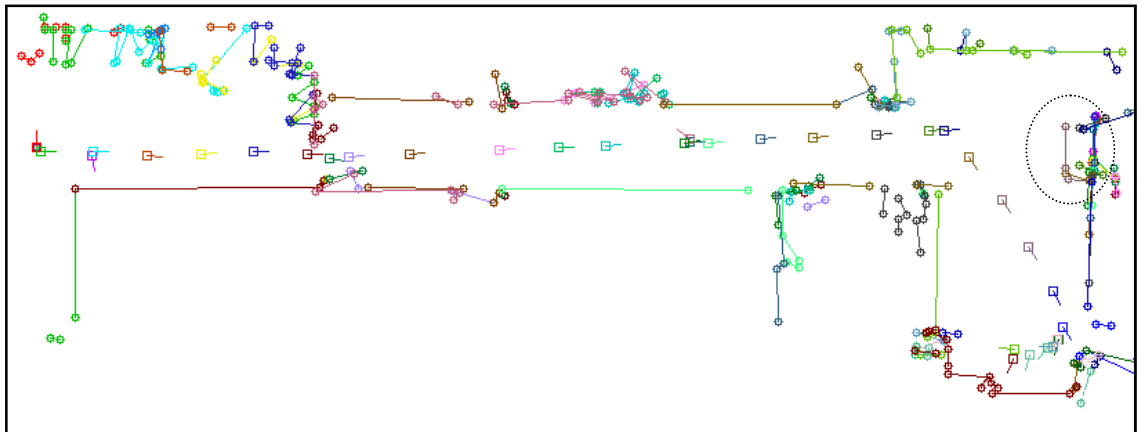
With the MFIS and ASR thus implemented, experiments can be carried out in order to see how well the algorithm performs.

### 3.2.3 Experimental Results

A laboratory robot equipped with a laser rangefinder has been run through different environments, mapping its surroundings by using the new implementation. An integral claim of the cognitive approach is that it can generate usable maps without needing any form of error correction. In order to verify this, the bad robot is used exclusively. The experiments are conducted in the same way as the ones for the traditional algorithm: robot movement is controlled by user input, but the mapping process is completely automatic. Also the robot has no prior knowledge about the environment or its own position.

#### Surface placement in the MFIS

Of great interest is the update mechanism which inserts new surfaces into the MFIS. Can the algorithm place them correctly without error correction? Figure 3.16 below shows the global map of the MFIS after travelling approx. 15m through an office environment, in 34 steps. The surfaces which have been seen are marked in a different colour for each view, together with the position of the robot at that particular time.

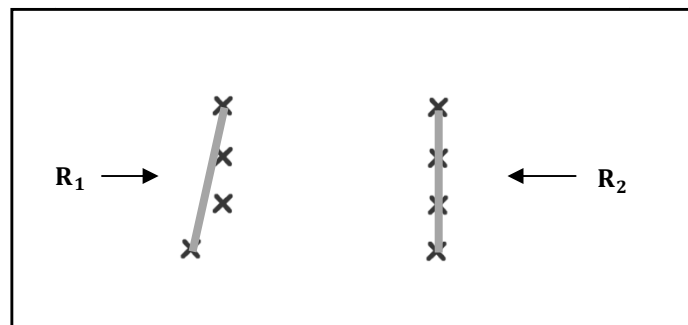


**Figure 3.16:** Surface placement in the MFIS

It can be seen that there are little duplicates found in the MFIS, especially for the long surfaces. There are still a lot of small surfaces which change shape when viewed from a different angle but, as explained before, these are not our primary concern. The long surfaces, in contrast, look very good: most of them exist only once in the picture, which means the algorithm has correctly matched them in subsequent views. It is also apparent that, while single surfaces are sometimes displaced, the overall shape is

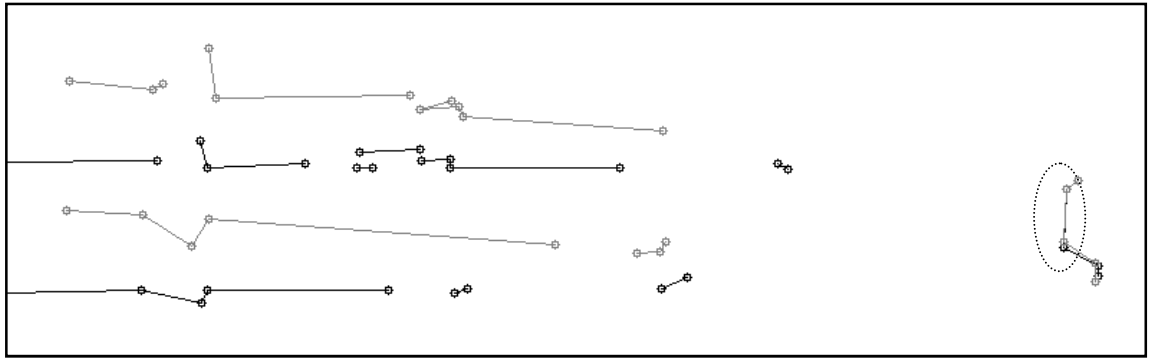
almost perfectly rectangular. This supports a basic claim of the algorithm, namely that errors - although contained in the map - are not propagated and thus do not need to be corrected.

One surface stands out from the image: the one encircled to the far right. It is apparent that this surface should have been matched to the wall next to it; but this is not the case and it has instead been placed at a wrong location. This is very troubling, since the spatial deviation is so high. How could this happen, especially as the other surfaces seem to have been placed quite well? The reason for this lies in the recognition of common objects between two views. Although a similar surface did exist in the MFIS before, it could not be matched to the new one, since the matching criteria (distance and orientation) are highly ambiguous. Subsequently, the surface was treated as if it had not been seen before, and added to the map via reference target. As a coincidence, it seems that a bad reference target had been chosen and thus the surface got displaced.



**Figure 3.17:** A bad reference target

Consider Figure 3.17 above: when scanned from a different viewpoint, surfaces sometimes change shape – this is because of inaccuracy in the laser input. Matching them can then become a problem: the algorithm assumes that an error always affects both surfaces together, and is negated by matching them. In this case however, only one of them carries the error, which subsequently cannot be negated upon insertion. As a result, the matching pair is unsuitable as a reference target: it will displace any surfaces relative to it. Figure 3.18 shows an exaggerated example:



**Figure 3.18:** Surface displacement when picking a bad reference

Here, two views are shown superimposed on each other. The algorithm has identified a common surface (encircled to the right), containing an error such as described above. Using this flawed match as a reference target then results in a considerable displacement of new surfaces, as can be seen from the image.

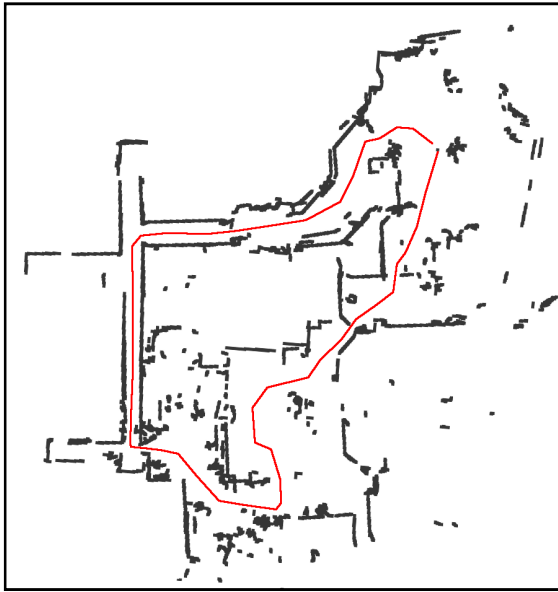
The obvious conclusion is that such inaccurately recognized surfaces must not be used as reference targets. Accuracy in surface detection depends on a number of factors, such as surface length, distance from the robot, or the orientation relative to the laser sensor - all of which have been incorporated into the implementation in order to avoid the problem. However, at some point there will always be errors in one form or another and the only solution is more powerful recognition. The problem observed above clearly shows that comparing two-dimensional surfaces is not enough to reliably identify common objects.

In spite of these difficulties, the environment initially shown in Figure 3.16 has still retained its overall shape; and even though some surfaces have been heavily displaced, the bulk of the environment has been mapped quite accurately. This strongly suggests that the errors encountered are not propagated across views, and that the MFIS remains usable even with them. Naturally, in order to test this assumption, the algorithm has to be tested in a larger environment.

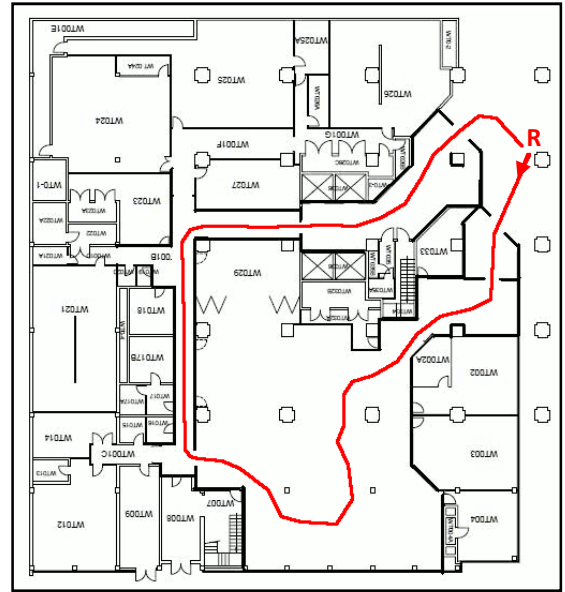


### MFIS accuracy in a large environment

The algorithm has been tested on a large environment to see if it produces useable maps even after longer journeys. It has been run through an office building which contains both large rooms and long corridors. The route travelled in this experiment is approx. 120m long, and the MFIS has been updated in 98 steps. Also the start and end points of the route connect, forming a loop.



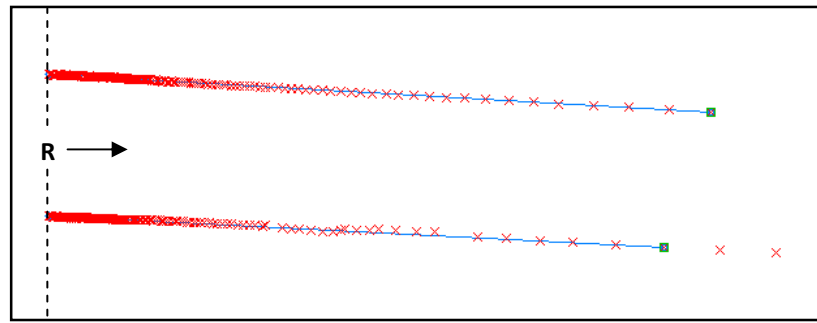
**Figure 3.19:** MFIS of large environment A



**Figure 3.20:** Floor plan of large environment A

The MFIS shown in Figure 3.19 captures the spatial layout of the environment quite well; there seems to be no error propagation whatsoever and the ends of the route match almost completely. However, similarly to the smaller environment described before, many small surfaces litter the MFIS. These are random obstacles – the rooms traversed were full of tables, chairs and cardboard boxes. Some duplicate surfaces can also be seen, showing that errors do exist, but they seem to have no impact on the overall shape of the map. The path of the robot is clear and not blocked at any point, thus resulting in a usable map.

There was however one special situation during the experiment: as the robot was travelling along the straight corridor on the left, it could at one point not identify any common surfaces between successive views. This can be very dangerous, since the algorithm cannot add new surfaces – or locate itself – without a reference target. The cause of this can easily be identified when looking at Figure 3.21:



**Figure 3.21:** Surfaces detected by laser sensor inside a corridor

The robot has encountered the so-called corridor problem: inside a perfect corridor, one cannot identify any unique features by which to orient oneself. As can be seen in the image, the only things available are two walls – and if the corridor length exceeds the range of the sensor, then no information about their length or position is available whatsoever. From only this input it is thus impossible to determine how far along the corridor one has travelled already.

When no reference target has been found, the robot would normally have to stop since it is lost. To be able to continue, it was instructed to transform the problematic view with odometer data (like in the traditional implementation in section 3.1). Luckily, after two more steps, the robot detected new surfaces near the exit of the corridor, providing it with a new reference target. Thus it was able to update the MFIS again. Of course this workaround is very limited, since the algorithm has to rely on its position estimate when travelling without a reference target. If this happens too often, then the errors observed in the traditional approach will be introduced and the map will get corrupted. In this experiment however the robot travelled only a very short distance like this, and thus the overall map retains its shape. But had it encountered heavy errors at this specific point then the map would have been rendered useless.

### Comparing the MFIS to the traditional approach

The algorithm has also been tested in the same environment where as the traditional approach. The robot has been steered on the same path, and thus both algorithms have been tested with similar input. The length of the route traversed is approx. 90m, and the MFIS has been updated in 106 steps. Figure 3.22 shows the map obtained by the perceptual mapping algorithm:

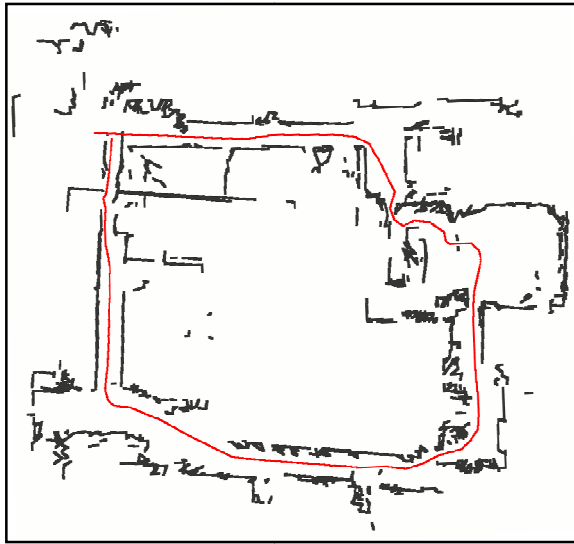


Figure 3.22: MFIS of large environment B

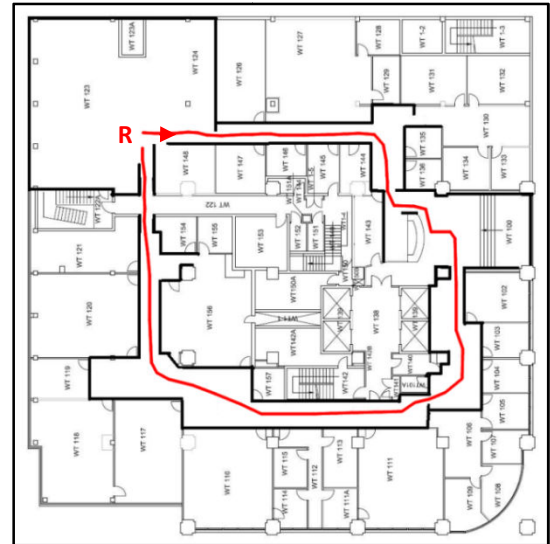


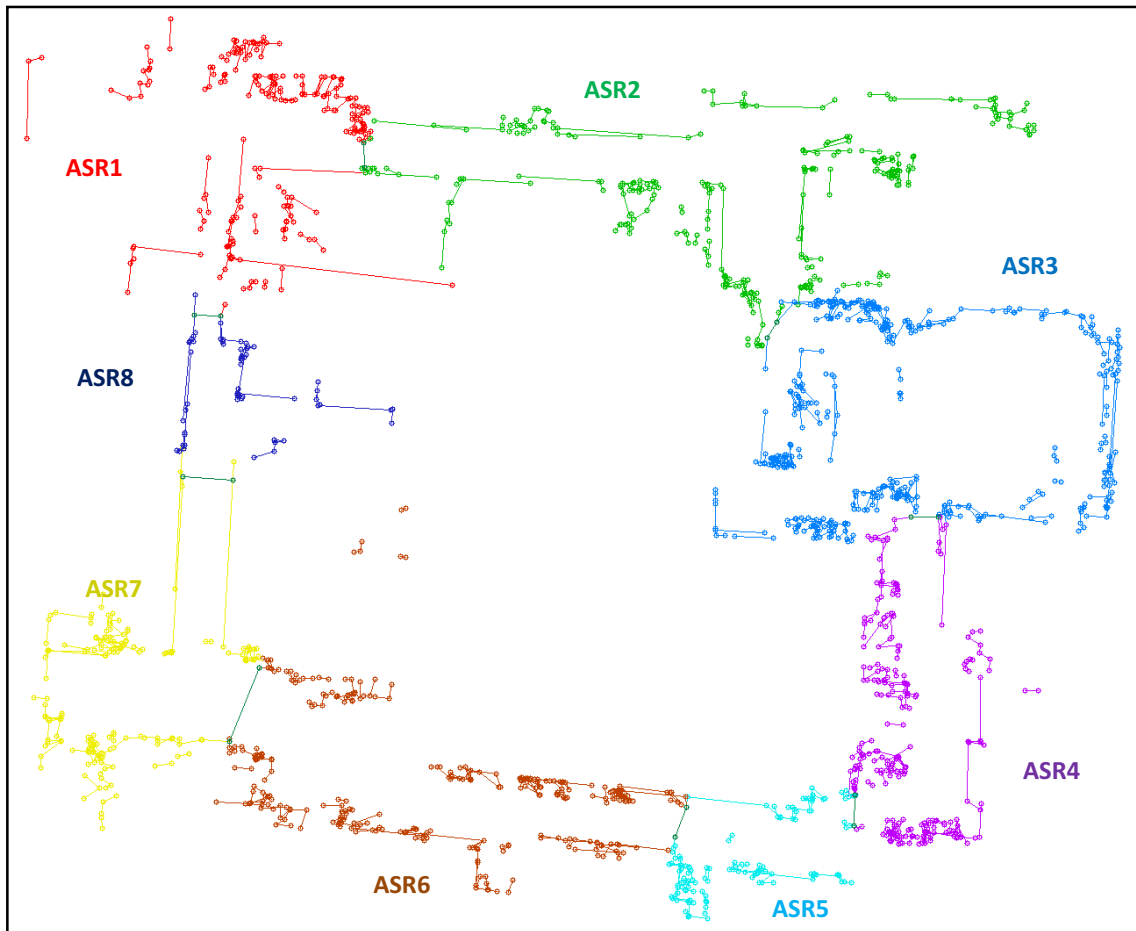
Figure 3.23: Floor plan of large environment B

It can easily be seen that the MFIS has a good shape and that no duplicate surfaces are blocking the way of the robot. This stands in sharp contrast to the result obtained by the traditional approach (see section 3.1): while the traditional map is rendered useless by the errors of the bad robot, the perceptual mapping approach can still generate a useful map. Thus it can be considered superior in terms of mapping capability. At this point, the good robot is not considered because its accuracy makes it unattractive for testing the merits of the algorithm.

The only visible flaw is that the ends of the map do not completely connect; this can be seen when comparing the route with the floor plan. It shows that, although the MFIS is fairly accurate, it alone is not enough for reliable navigation. At some point there will always be errors – a spatial map can go only so far. As suspected in the algorithm description earlier, an abstraction into ASRs is necessary.

### Generating ASRs

While travelling, the algorithm does not only update the MFIS, but it also tries to detect exits and form ASRs from the environment. When the robot traversed the large environment “B”, it created a network of ASRs at the same time. Figure 3.24 shows the separation of local spaces:



**Figure 3.24:** Large environment B separated into ASRs

The image shows how the surfaces from the MFIS have been assigned to individual ASRs, where each colour denotes a different local space. Thus the figure shows 8 ASRs. The separation seems quite reasonable in most cases, for example ASR2 and ASR3 might have been separated in the same way by human who is looking at the map. It seems a little strange to separate ASR5 and ASR6 in this manner though, as the exits shown are not quite the narrowest passages. This can be attributed to the exit finding technique used, which is quite simple, however it is not necessarily a bad thing. After all it is better to separate a very large space into several smaller ones – little can be gained if the whole map consists only of a single ASR.

Figure 3.24 also shows the primary benefit of using an ASR network: the loop has been detected. The ends do not perfectly match in the MFIS; however they are still reasonably close. After exiting ASR8 to the north, the algorithm tries to create a new ASR. However, it then realizes that the new space mostly overlaps with ASR1. Thus it can simply reason that it has arrived in ASR1 again, and connect it to ASR8. In the picture produced by the implementation, this can be seen from the red-coloured

surfaces: both the start and the end of the path are contained in the same ASR. The robot has thus detected the loop and successfully connected the network.

### 3.2.4 Discussion

The algorithm produces maps that show a good spatial layout of the environment traversed, even though no error correction is being used. When compared to the traditional approach, the MFIS is much more accurate, even though extreme precision had not been the main aim.

The problem of overlapping surfaces still remains: this happens because the surface matching technique employed cannot always identify the correct matches. This is also the cause of the corridor problem observed – even though the robot could recover from the situation. It should thus be stressed that more powerful methods for recognition are needed, for example using cameras, in order to unlock the full potential of the approach.

It has been shown that a combination of MFIS and ASRs enables a mobile robot to recognize places it has already visited. In this manner the looping problem can be solved, and any global mapping errors - however small - become irrelevant. The prime advantage here is that, instead of having to change the MFIS to accommodate the loop, the algorithm can simply reason that the ASRs are the same. This seems to confirm the claim that that a topological abstraction of places is needed, especially to map larger environments. The ASR recognition algorithm is only expected to fail if the errors are so great that not even a rough similarity between local spaces can be observed anymore. This could be improved in the future: in this implementation, similar ASRs are determined by proximity alone. Ultimately though, an ASR could have many defining features, such as specific objects inside of it, or the shape of its boundary. If techniques are developed for recognizing such features, the algorithm can be expected to become even more resistant to mapping errors.

### 3.3 Making the robot go home

Now that the robot is able to build a representation of its surroundings, the question is: how can this information be used to travel and navigate autonomously? The previous sections have shown experiments where the robot simply follows human instructions. This section, in contrast, will look at navigating automatically to a defined target, again employing human-inspired strategies as much as possible.

#### 3.3.1 Motivation

One basic problem typically described by researchers (Pfeifer, 1995; Wong, Schmidt, & Yeap, 2007) is having the platform find its way back home after exploring an unknown environment. At a first glance, the solution seems quite straightforward: shouldn't it be easy to just retrace the movement path and thus travel right back to the start?

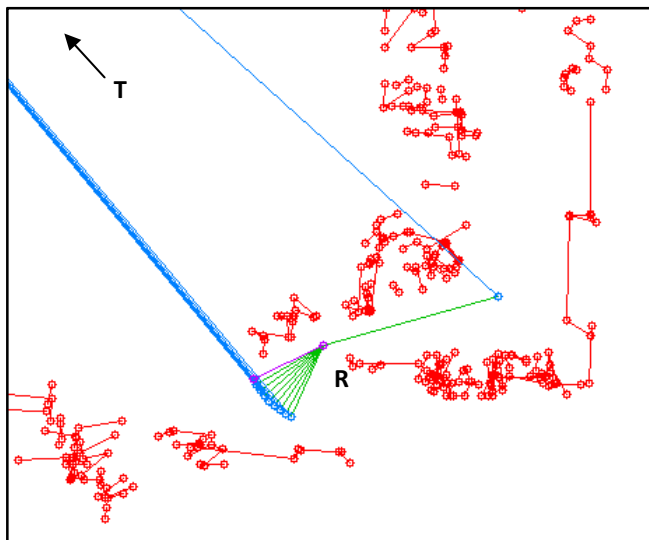
The answer is yes and no. Yes, because both the path and the environment are already known. No, however, because the world keeps changing, and thus one should not blindly trust one's memory. If the robot simply follows a path from memory, it will quickly bump into things that weren't there before; or crash into doors that have been closed. That means the solution has to be more sophisticated: even while going home, the robot has to watch its surroundings and make spontaneous decisions based on what it sees.

Such spontaneous decisions can be used for more than just obstacle avoidance. A skill at which humans excel is the identification and taking of shortcuts. Even while already following a path, the possibility of a shortcut should be explored if it presents itself. For this humans employ an intuitive sense of direction, quickly abandoning the old path when a promising opportunity arises - even if it means entering previously unexplored areas (Hochmair & Frank, 2000). So how would one go in equipping a robot with a sense of direction?

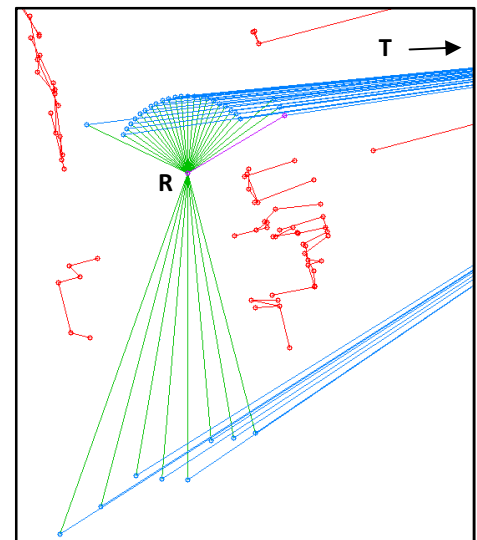
### 3.3.2 Deciding which way to go

We start with a robot that has travelled into the unknown and wants to go back. It roughly knows the direction of its target (from memory) but has no idea how to get there. Although a possible route has already been explored on the way - and a map (MFIS) exists - it is not guaranteed that this route is the shortest one, or even still valid at all. As seen in section 3.2, there might also be inaccuracies in the map which would make it difficult to retrace the exact path. The algorithm described here will thus focus on exploring new, shorter ways to the target, and only fall back to the known path if absolutely necessary.

The fundamental question for the robot is: “in which direction should I move?” To answer it, the current view is scanned and divided into a limited number of movement options. These are empty spaces where the robot can pass through, and it has to choose one of them. The number of such options can greatly vary - in a narrow indoor scenario there might be only 1-3 options, whereas in the outdoors (or a big room) a much greater range of movement is possible. Compare for example the figures below: a green line denotes a visible movement option, and a blue line is an assumed (but not necessarily correct) connection to the target.



**Figure 3.25:** Movement options in a narrow environment

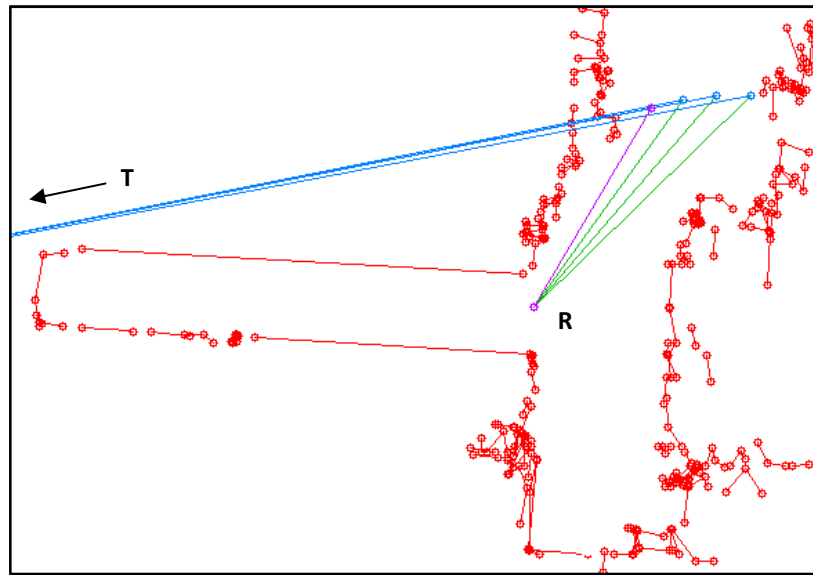


**Figure 3.26:** Movement options in a wide environment

**Legend:**

<span style="color: red;">—</span>	Geometry from MFIS memory	<span style="color: purple;">—</span>	Movement choice
<span style="color: green;">—</span>	Movement options	<b>R</b>	Robot position
<span style="color: blue;">—</span>	Estimated path to target	<b>T</b>	Assumed target position

From these options the algorithm must choose which one seems to be the most promising, hopefully yielding the shortest way to the target. Since much of the environment is unexplored, the choice may not be optimal. The robot tries to choose the most promising option, but upon exploring it might find out that it is a dead end. Consider Figure 3.27: the robot was about to move in direction of the target, only to realize the way is blocked. Now it has to turn back, and take a big detour to find a different path. The spaces to the left and to the bottom have no openings wide enough for the robot, thus the only viable movement options are to the top.



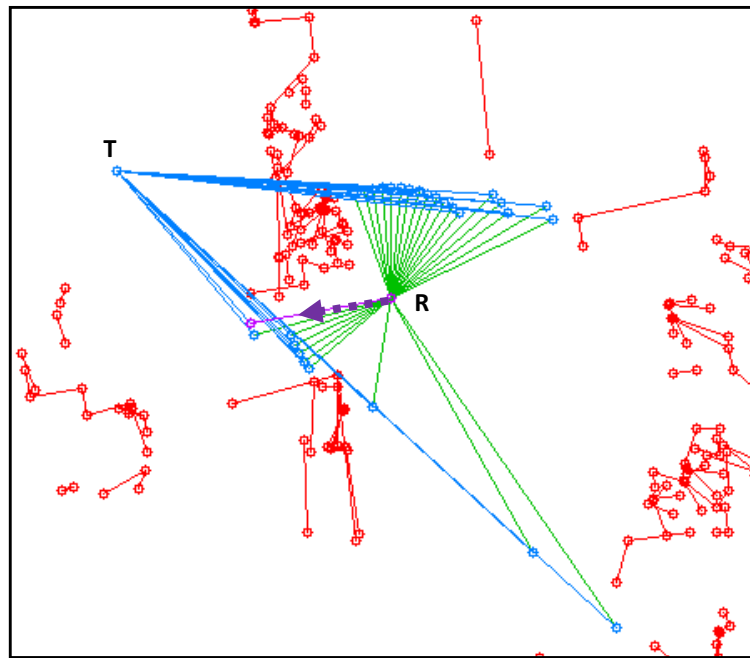
**Figure 3.27:** Moving away from a dead end

The chosen path consists of the part that we can see (movement option), and the part that we can't see but assume (connection to the target). That means it involves guesswork: if we follow the path up to where we can see, can we then reach the target from that position? This reasoning tries to mimic a human, planning around the next corner: “This direction leads away from my target, but in the distance I can see a turn, leading in the right direction again.” Such logic may seem quite simple, but there is good reason to believe that humans, intuitively following their sense of direction, make equally simple choices. An optimistic human will often just think up to the next corner, and if it turns in the right direction explore the option (Hochmair & Frank, 2000). In the same way, the “sense of direction” algorithm uses no path planning at all, but rather navigates around corners on an otherwise direct way to the target.



Coming back to Figure 3.27, it is evident that the blue lines (assumed path to target) cross impassable geometry of the MFIS. The data in question is quite old and has not been seen for some time, thus the robot simply ignores it for path planning – optimistically hoping it can find its way to the target. Of course - as will be observed in an experiment - once it does decide to move in that direction, it will see the obstacle again and then must plan a way around it.

These movement options are the heart of the algorithm. Of all identified options, it is decided which one seems the most direct. And since the robot tries to follow a simple sense of direction, it is the one forming the straightest line to the target promising (highlighted with an arrow in Figure 3.28). Here it should also be noted that the platform does not necessarily have to travel the full length of this option. As it moves, there might be new, better options emerging and the robot will thus keep scanning and re-evaluating its decision while travelling.



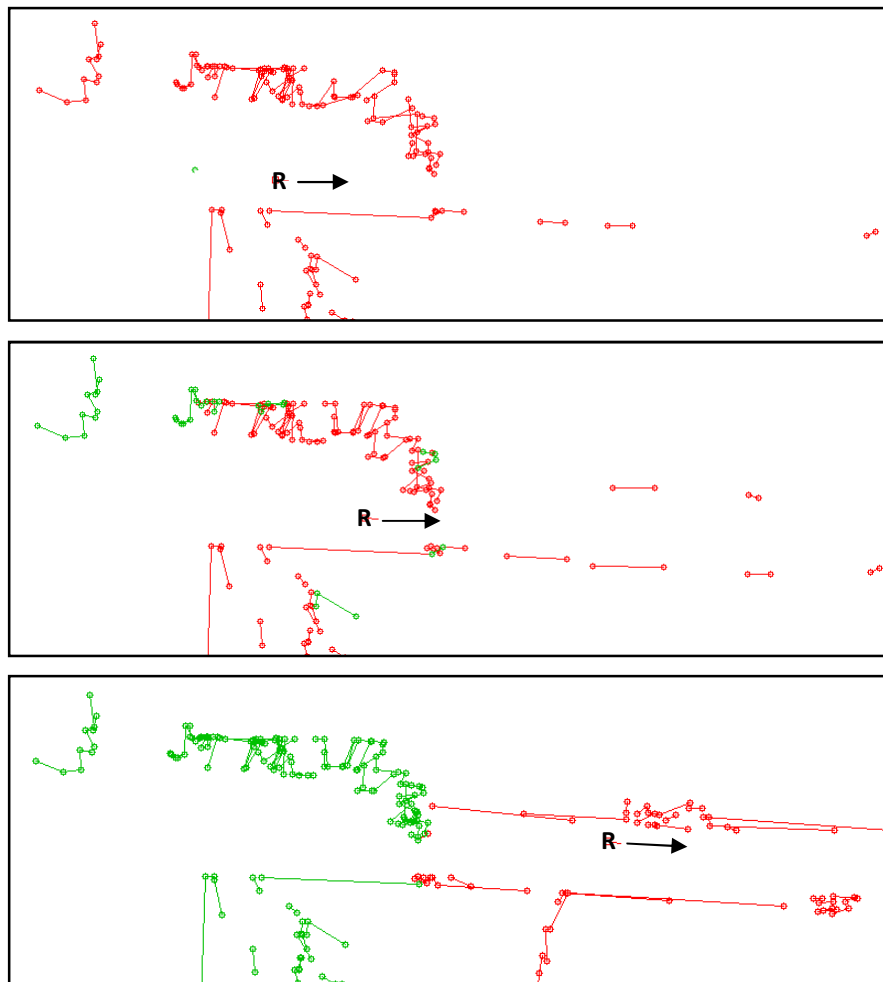
**Figure 3.28:** The most promising option

### 3.3.3 Using the MFIS while travelling

While trying to find its way home, the robot has to keep updating its MFIS at every step. Otherwise it would lack the “vision” to properly recognize new movement options as it travels. But how could one use this geometry information? Since the MFIS

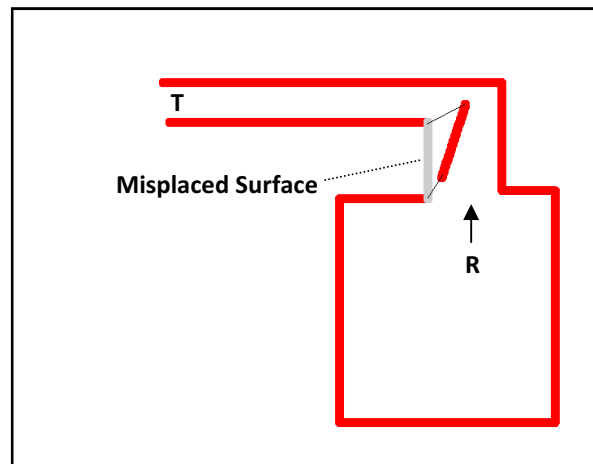
is built while travelling, it is highly incomplete. Traditional, mathematical, path planning can be very difficult to use with such a map: after all, it is highly improbable that the robot will explore the environment thoroughly enough for such a calculation. It has also been shown in section 3.2 that, although the MFIS can be reasonably accurate, individual surfaces might be heavily displaced. That means that even if geometry information about a distant place exists, it is probably not accurate enough.

For this reason, the robot can only trust what it recently saw and should thus only remember a small number of old views. In this manner it knows little more than its immediate surroundings – not much, but enough to start moving. In the same way, a human would forget the smaller details of a past journey and not take them into account when intuitively deciding whether to go left or right (Wang & Spelke, 2002).



**Figure 3.29:** Forgetting old geometry while moving

Figure 3.29 shows the robot R forgetting old geometry (green colour) while it is moving to the right. The old data is removed if sufficiently far away from the platform, and if it has not been seen in several previous scans. There are, of course, some special situations to be accounted for. What happens if the robot gets stuck and can't figure out any movement option at all? This can occur sometimes, for example when the pathway is very narrow, or when a wrongly recognized MFIS surface appears to block the way (Figure 3.30). So if there are truly no more promising movement options, what failsafe mechanism will allow the platform to escape this "trap"?



**Figure 3.30:** Robot gets stuck through inaccurate MFIS

In addition to this – as old geometry is quickly forgotten - the robot runs the danger of making the same choices twice. While trying to find to the target it will make many decisions that lead to bad situations, like dead ends. There has to be a mechanism that makes sure it doesn't pick the same - bad - option again. At all times, the robot should know where it came from, or else it could get caught in an infinite loop – exploring and subsequently forgetting the same places time and time again. Consider the following example: Figure 3.31 shows the platform moving away from a dead end. Several steps later, after old geometry has been forgotten, the algorithm wants to move back since it is the most direct connection to the target (Figure 3.32). Since the dead end is continuously re-explored and then forgotten, the situation results in an infinite loop.

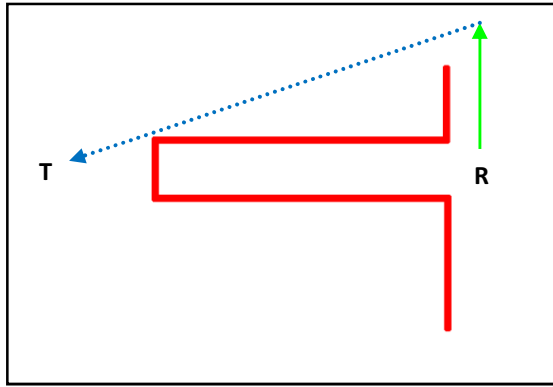


Figure 3.31: Moving away from a dead end

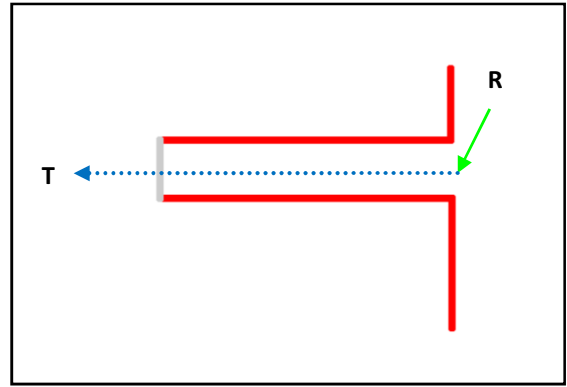


Figure 3.32: Exploring the dead end again

To counter this, there has to be at least a rough representation of the way already travelled, or else the robot might truly get lost. If all newly explored paths fail, it will still be possible to follow the route in memory back home. Of course, humans can also get lost if the environment is sufficiently complex. Getting lost however is hardly the aim of this research and thus - as a simple facility to recover from problematic situations - the robot retains an account of paths travelled in the past.

### 3.3.4 Following a known path from memory

If indeed the robot cannot find a way to travel in direction of the target, it has to trace back the remembered path. Since old MFIS data has been mostly forgotten, the only thing remembered is the route travelled which, in turn, can be generalized into major waypoints. When following, a movement option is picked which points in direction of the next waypoint, and the path is fitted on top of the (newly created) MFIS.

Naturally these waypoints have to be placed in such a way that the movement from one point to the next is trivial. When following the route, the robot will aim for the next waypoint on the path - again using its automatic direction-following system to get there. As a side effect of this, it will also be able to evade small obstacles when encountered. Such objects may randomly appear along the way, for example in the form of humans walking around, or even surfaces wrongly placed in the MFIS. Again this approach tries to imitate humans, as they will easily move around small obstacles in their way if the rest of the path is clear.

An example of such a path can be seen in Figure 3.33 below: the robot R wants to return to the target T, and the known route has been simplified and divided into waypoints. Though visible in the picture, most of the older MFIS surfaces have been forgotten and the robot merely knows the rough position of the next waypoint. As can be seen from the picture, there are indeed some minor obstacles between one point and the next. However, this is of no further consequence since to the algorithm obstacles appear as surfaces and are thus treated the same way corners: the robot simply navigates around them.

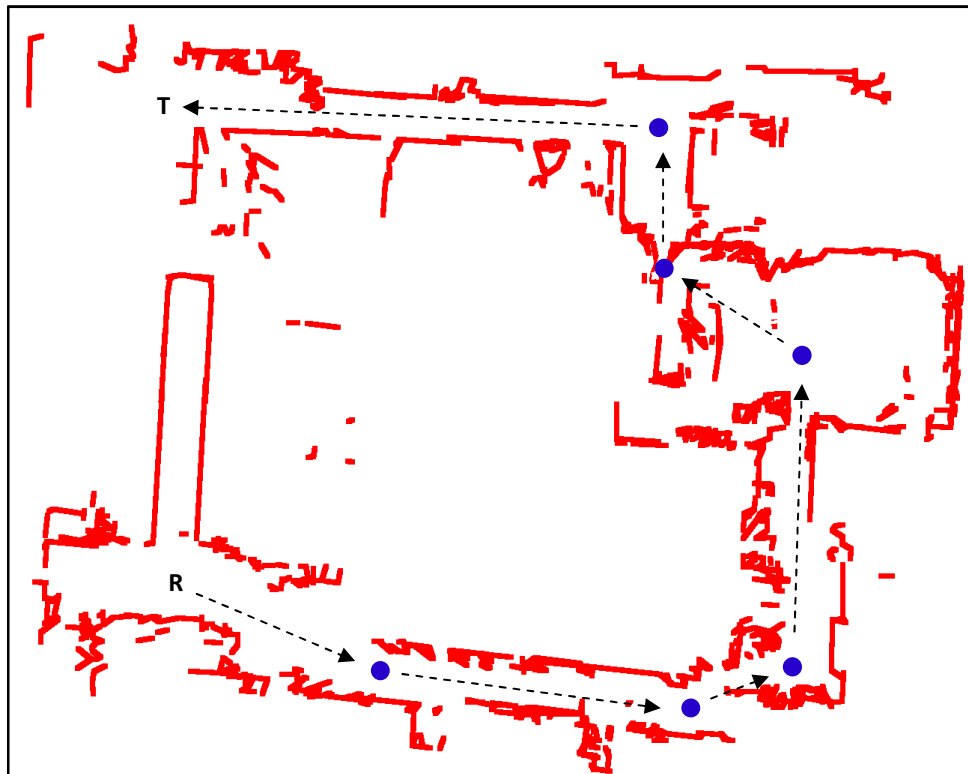
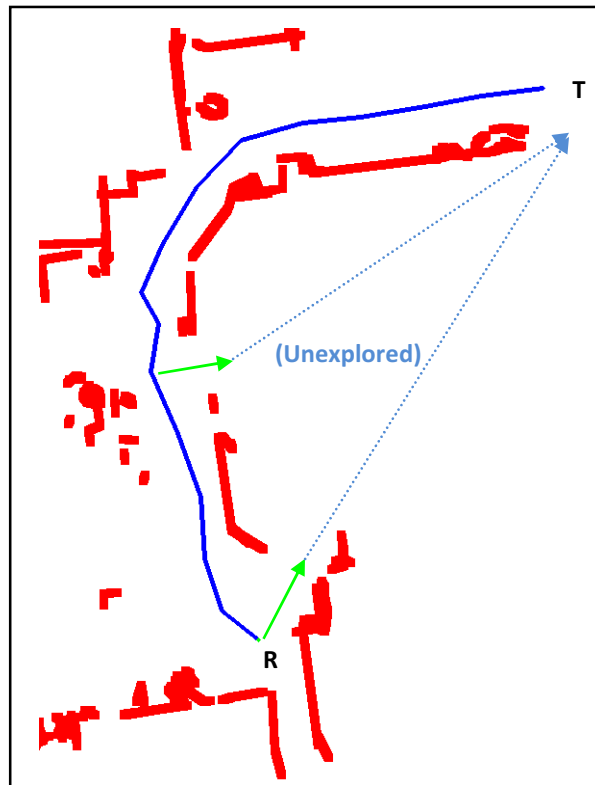


Figure 3.33: Waypoints for the path home

It should be noted though that, even when following a known path, if suddenly a new and promising movement option appears, the robot should stop retracing the old path and explore this direction, hoping to make a shortcut. Consider the path shown in Figure 3.34. In this example, the robot R wants to find back to its home and target T. While travelling along the already known route, it will find two points where a shortcut seems possible. Since the direction looks promising, the platform will explore these new options.



**Figure 3.34:** Shortcuts in a path

Naturally the decision of whether to explore or discard a new path has to be based on something. One could argue that it is a risk-vs.-reward situation: the possible reward is a shorter route; the risk is encountering a dead end and having to turn back. For the sake of simplicity, the algorithm will always try and take a shortcut if it points in direction of the target. In this manner, it tries to mimic an optimistic and curious human.

And finally, it should be mentioned that remembering an old path through path points may not be a human-like approach. There is evidence to suggest that humans rarely remember locations in a geometric sense, but rather by identifying objects and landmarks in these places (Golledge, 1999). Since this algorithm strives to make decisions similar to a human, this is a setback – but a necessary sacrifice because object recognition is very difficult with a simple laser rangefinder. Thus the robot has to “cheat” by remembering geometric data of old paths. Of course it would be preferable to employ object-recognition techniques in order to identify places, probably using cameras. This is a topic that can be expanded on and could be the focus of further research in the area.

### 3.3.5 Experimental results

In this section, a laboratory robot is run through a number of different environments in order to see how well it can find home. In each case the platform starts off travelling into the unknown, manually controlled by human input, until the “Go Home” command is given. From this point on, the robot acts autonomously. Forgetting most of its old MFIS memory it tries to explore paths that might possibly lead back to the starting point, using the known route only if seems to be the shortest option. The platform then travels until it is roughly near the starting position. As mentioned above, the algorithm does not include feature detection to recognize the exact starting point, thus a rough proximity to the target will be counted as success.

#### Finding home around a corner

The first experiment is rather simple: the laboratory robot is deployed in an office environment and steered around a corner – manually controlled by a human. After travelling a distance of roughly 10 metres, the “Go Home” command is given, instructing it to find its way back to the start autonomously.

Figure 3.35 is a plot of the route travelled by the robot, with the original path superimposed on the image. As can be seen, the platform successfully finds home, performing a U-turn around the corner. It should be noted that for this experiment the old, known, path was not used by the algorithm. The route is only similar because there are no shorter options.

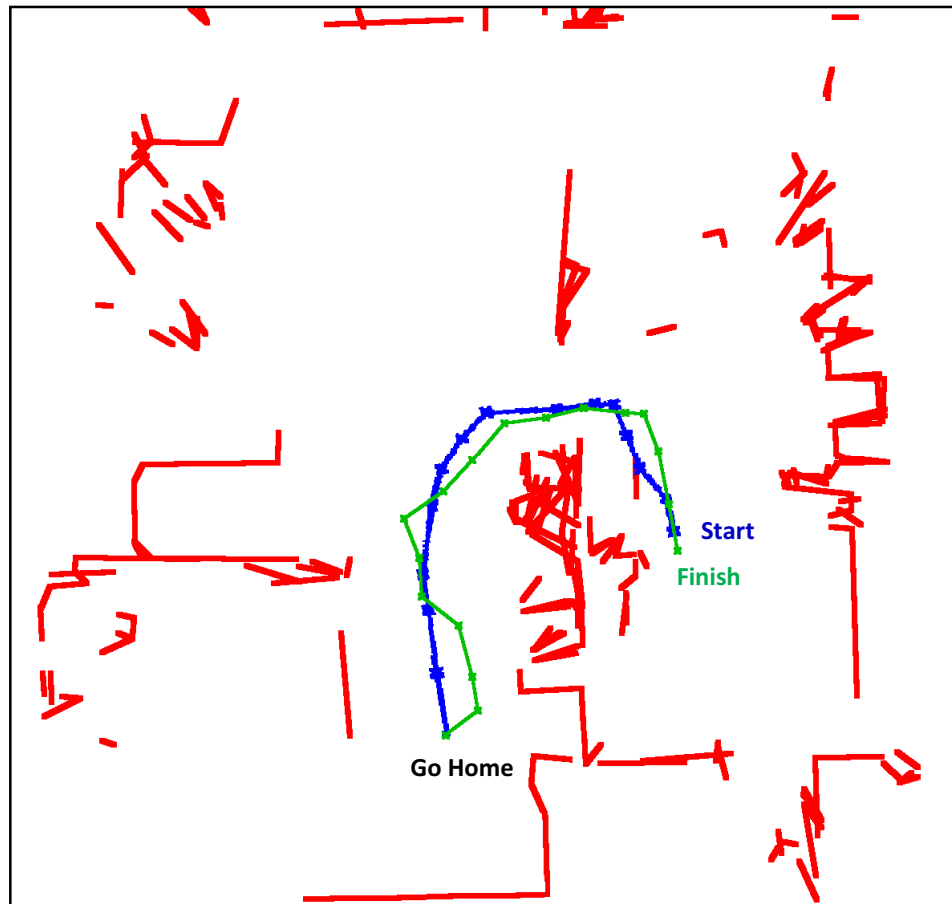


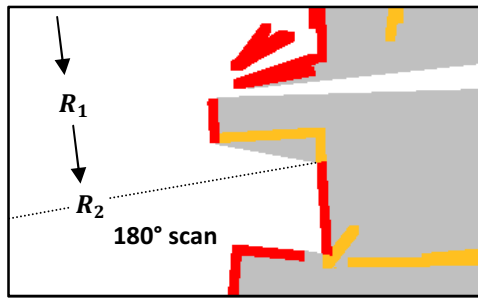
Figure 3.35: Robot travelling home around a corner

**Legend:**

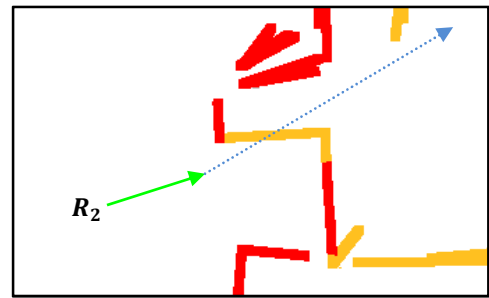
<span style="color: red;">—</span>	MFIS geometry
<span style="color: blue;">—</span>	Original path, manually controlled by human
<span style="color: green;">—</span>	Path taken autonomously by robot to find back to the start
<b>Go Home</b>	Point where the “Go Home” command is given

There is one rather curious phenomenon though: why is the green line not straight, but rather follows a zigzag pattern at the start? And why does the robot sometimes travel towards the wall, where there path is definitely blocked? The reason for this lies in the MFIS update mechanism. Since the obstacle in the middle is fairly complex, only parts of it have been seen by the robot. Thus the map may contain holes, and the algorithm will try to move towards those. Consider the figures below, showing the first decision after receiving the “Go Home” command. While travelling along the blue path, the machine can scan only what is in front of it, and thus the surfaces marked in orange have not been seen. In this particular case, the algorithm concludes there may be an opening in the wall and a valid movement option passes through it. Since this would be the shortest path to the target by far, the robot naturally decides to take it.





**Figure 3.36:** Visibility of MFIS geometry



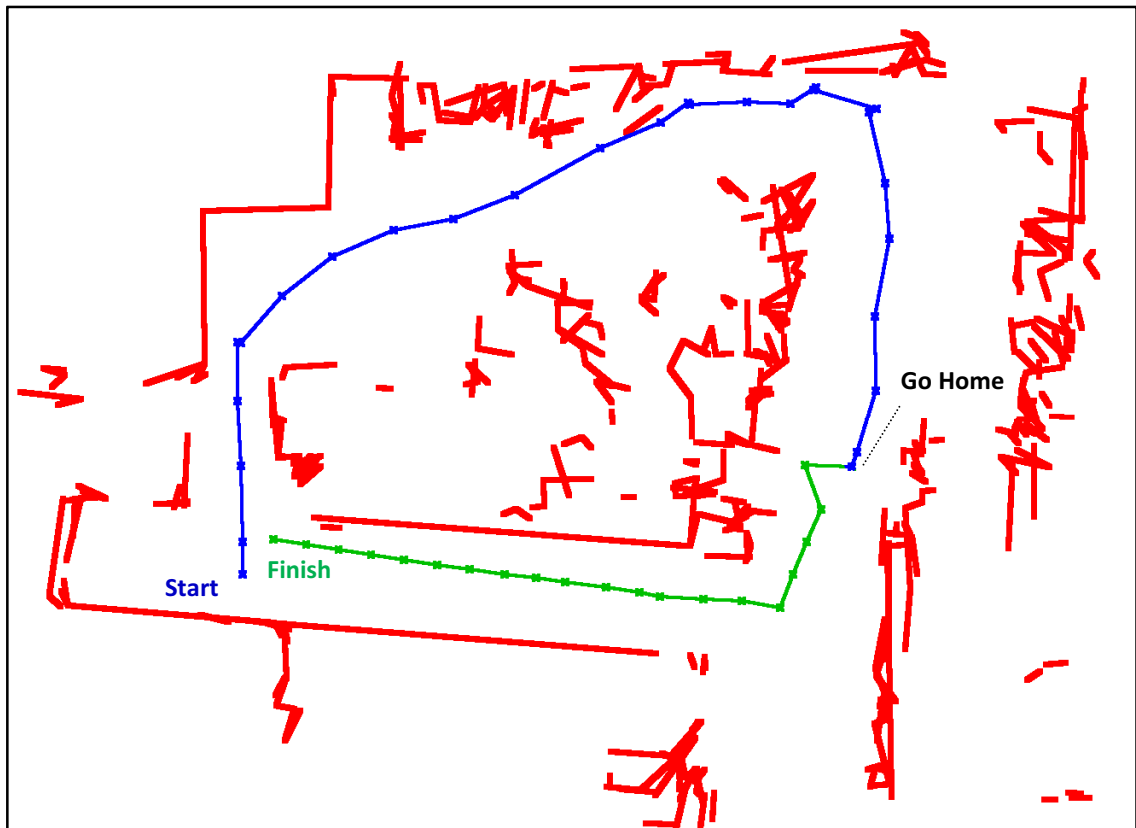
**Figure 3.37:** A valid movement option?

Of course, once the first step is taken, the platform turns towards the target and a new scan reveals the hidden surface. Now a movement in this direction is no longer feasible, and so the robot has no choice but to navigate around the corner, and find the correct way back to the start.

The experiment shows that the algorithm is able to handle simple obstacles, while it can also be seen that the robot will often drastically change its direction of travel as MFIS updates bring up new movement options. It is however able to travel back to the start, so the intended goal has been reached.

### Taking a shortcut

Again, the robot is deployed in an office environment. It starts off manually controlled by a human and is then positioned in such a way that it can take a shortcut home. The path for the shortcut is quite easy, and one might say that it appears obvious – for a human at least. Thus the aim of the experiment is to show that the algorithm can take those easy shortcuts. The environment is also larger than the first one: the length of the route travelled in this run is about 30 metres.



**Figure 3.38:** Robot travelling home and taking a simple shortcut

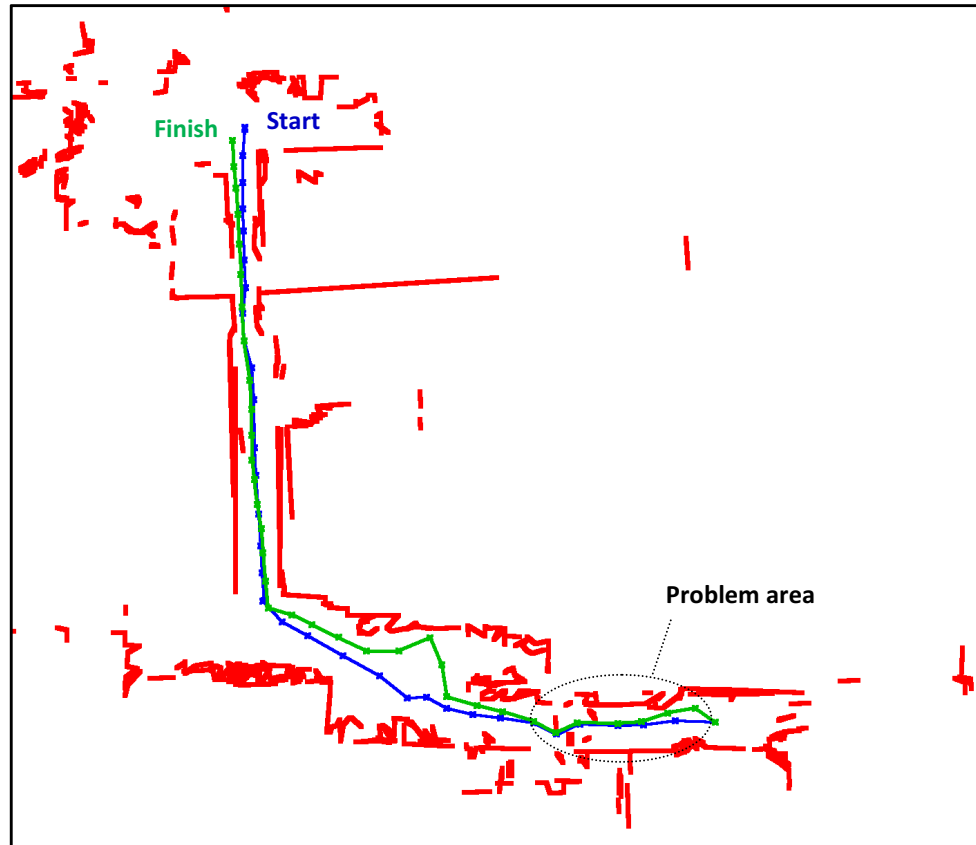
As can be seen in the picture, the robot manages to find back to the starting point, taking the simple shortcut by following its sense of direction. Note that the green path is significantly shorter than the blue one. Also - similar to the first experiment - in the first step the platform tries to move through an assumed gap in the wall, only to find it closed. The rest of the path is quite straightforward since the target is already in sight.

One thing that particularly strikes the eye is that the start and finish point do not match – by a distance of more than one metre! This is caused by inaccuracies in the MFIS, bringing back the issue of loop closing (see section 3.2). The shape of the map is not completely straight in this area. That in itself is not a problem, since the MFIS is not designed to be precise, but through this it becomes impossible to exactly identify the starting point.

The platform finishes in rough proximity to the starting point, so the basic goal of the experiment has been reached. It shows that the robot can take shortcuts – at least simple ones – but not being able to properly identify the target can cause problems, especially in larger environments.

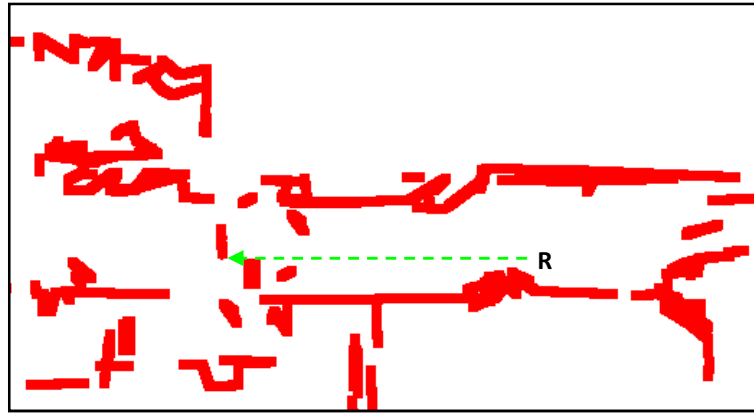
### A narrow obstacle course

As in the experiments before, the robot is once more used in an office environment. This time however the travelled route is significantly longer (approx. 50m), and the only path to the target snakes its way through a collection of obstacles.



**Figure 3.39:** Robot travelling home around obstacles

Figure 3.39 shows the route travelled by the robot. While it seems correct at first sight, the direction-finding algorithm was actually unable to find the route on its own. In the problematic area, which is encircled in the picture, the robot was unable to plan a path through the dense collection of obstacles (which consists of cardboard boxes). Figure 3.40 shows this in greater detail:



**Figure 3.40:** Dense collection of obstacles

It can easily be seen that there is no way to escape this situation by planning a path in a straight line. Since the area is quite small, all surfaces have been recently seen and have thus not yet been forgotten in the MFIS. Thus the algorithm believes it is blocked in all directions and cannot determine a movement option. This shows that the linear “around-a-corner” path planning is not very successful when confronted with a narrow obstacle course. This being said, how did the robot still manage to find back home?

If it cannot use its direction-following algorithm anymore, the robot switches to the fail-safe mechanism: it starts following the recorded route which it has taken to get here in the first place. By employing this strategy, it managed to escape the situation. This is also the reason why, in Figure 3.39, the paths almost overlap in the problem area. Once the robot had left the obstacles behind, it started to deviate from the known path again. The reason is simply that it could at this point detect new movement options and subsequently did not need to rely on the known route anymore.

It can thus be said that the algorithm is overwhelmed by a narrow and complex environment. While it can still recover from the situation, this is only made possible by retracing its steps. Once however the robot is out in the open again, it can reliably find to the target even over a longer distance.

### 3.3.6 Discussion

The experiments highlight several strengths and weaknesses in the approach. The robot has been shown to find simple paths through unknown environments, but once its surroundings are sufficiently complex it gets confused. Especially narrow turns are a problem, as could be expected, because straight lines are used to plan a path. Dead ends are also frequently encountered since the robot is very optimistic about exploring potential – but still unknown – routes. Often it can be seen to move in a certain direction, only to find out one step later that the way is actually blocked.

However it has been shown that the robot can recover from such dead ends and keep moving by following an old, known, path and escape from the problematic location. In such occasions, where it fails to find a new path by itself, the robot will simply take the same way back which it took on the way there. This is very important because, by doing so, it can always continue moving. The experiments show that there is rarely a situation where the robot has to stop because it truly cannot figure out what to do, requiring user intervention.

Finally, it has to be noted that the robot will often travel in zigzag patterns and take detours by trying to explore paths that are later found to be invalid. Trying to explore every option uses a lot of time – one might argue that just following the old path can sometimes be much quicker. At this point a more complex decision system should be designed which dynamically chooses whether or not to explore a movement option. There should be additional factors in determining just how promising a certain option is, and if the robot should really explore certain paths. There is good reason to believe that humans do not judge by their sense of direction alone, but make much more complex decisions in order to find to their target (Golledge, 1999). Such behaviour should be researched further, so that more powerful navigation algorithms can be developed, again employing human-inspired way finding techniques.

## 4 Conclusion

This thesis shows the implementation of Yeap's theory of perceptual mapping using a mobile robot with laser sensors. The algorithm is inspired by human behaviour and the experimental results that have been obtained support important claims of the theory, such as a robot's ability to map and navigate its surroundings without having to use a metrically accurate map. To test the implementation, a robot has been used which experiences a high amount of errors in sensory input. This is important since human perception is also known to be fragmented and inaccurate. Even though the algorithm is designed to work with an imprecise map, it has been shown that the global map of the MFIS preserves a much better layout of the environment than the one produced with a traditional error-corrected approach - especially when confronted with heavy errors. The implemented solution is also able to generate a network of ASRs and close loops in the environment.

Furthermore it has been shown that autonomous navigation with the map generated from the algorithm is possible - and that a mobile robot can find its way home in a range of scenarios. The navigation approach which has been implemented is also successful in finding basic shortcuts through unknown areas of the environment, and does not rely on the known path home unless it gets stuck. It has thus been shown that navigation with the algorithm for cognitive maps is indeed possible, and this topic could be explored more in further research.

Ultimately, it can be said that the results obtained in this thesis support Yeap's theory of perceptual mapping, and that it could serve as a platform for further research into human cognition, as well as robot navigation in the future.

### Future work

Since the heart of the algorithm is its ability to track objects across subsequent views, it is important to have powerful recognition capabilities. Two-dimensional surfaces generated by a laser sensor have been found to be unreliable for this, and virtually all mapping problems encountered have been caused by insufficient recognition. Future

research should thus look at different sensors, such as cameras, in order to detect additional features in the environment.

Furthermore, all experiments have been carried out in indoor office environments. It should be interesting to see how surfaces are integrated in an outdoor environment, which poses several additional challenges. For example, open spaces are much larger than rooms inside an office. How could an outdoor environment be divided into local spaces? Depending on the sensors employed, there may be many objects and surfaces that are too far away. This means the robot has much less information for creating the map. It also might be confronted with uneven terrain and slopes, which could result in orientation problems and even greater errors. Thus, being able to map such an environment would further demonstrate the power of the theory.

Finally, the navigation experiments carried out were quite rudimentary, and served only as a demonstration that navigation with an imprecise map is possible – but the particular implementation in this thesis showed several flaws, such as an inability to cope with narrow groups of obstacles, and an overoptimistic exploration strategy. Further work should be conducted on navigation strategies which can be implemented to use the cognitive map. Since Yeap's theory of cognitive mapping is inspired by human behaviour, future research in the area should also look at how humans use their cognitive map, not just at how they build it.

## 5 References

- Bailey, T., & Durrant-Whyte, H. (2006). Simultaneous localization and mapping (SLAM): part II. *Robotics & Automation Magazine, IEEE*, 13(3), 108-117.
- Beeson, P., Modayil, J., & Kuipers, B. (2008). Factoring the Mapping Problem: Mobile Robot Map-building in the Hybrid Spatial Semantic Hierarchy. *Int. J. Rob. Res.*, 29(4), 428-459.
- Burgess, N. (2006). Spatial memory: how egocentric and allocentric combine. *Trends in Cognitive Sciences*, 10(12), 551-557.
- Dissanayake, M. W. M. G., Newman, P., Clark, S., Durrant-Whyte, H. F., & Csorba, M. (2001). A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3), 229-241.
- Golledge, R. G. (1999). *Wayfinding Behavior: Cognitive Mapping and Other Spatial Processes*: The Johns Hopkins University Press.
- Hochmair, H., & Frank, A. U. (2000). Influence of estimation errors on wayfinding-decisions in unknown street networks – analyzing the least-angle strategy. *Spatial Cognition and Computation*, 2(4), 283-313-313.
- Intraub, H. (1997). The representation of visual scenes. *Trends in Cognitive Sciences*, 1(6), 217-222.
- Ishikawa, T., & Montello, D. R. (2006). Spatial knowledge acquisition from direct experience in the environment: Individual differences in the development of metric knowledge and the integration of separately learned places. *Cognitive Psychology*, 52(2), 93-129.
- Jefferies, M., Baker, J., & Weng, W. (2008). Robot Cognitive Mapping – A Role for a Global Metric Map in a Cognitive Mapping Process. *Robotics and cognitive approaches to spatial mapping*, 265-279.
- Jefferies, M., & Yeap, W. K. (2001). The Utility of Global Representations in a Cognitive Map. In D. Montello (Ed.), *Spatial Information Theory* (Vol. 2205, pp. 233-246): Springer Berlin / Heidelberg.
- Kuipers, B. (2000). The Spatial Semantic Hierarchy. *Artificial Intelligence*, 119(1-2), 191-233.
- Lynch, K. (1960). *The image of the city*. Cambridge: M.I.T. and Harvard University Press.
- Mou, W., McNamara, T. P., Valiquette, C. M., & Rump, B. (2004). Allocentric and Egocentric Updating of Spatial Memories. [doi:]. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 30(1), 142-157.



- O'Keefe, J., & Nadel, L. (1978). *The hippocampus as a cognitive map*. Oxford: Clarendon Press.
- Passini, R. (1984). Spatial representations, a wayfinding perspective. *Journal of Environmental Psychology*, 4(2), 153-164.
- Pfeifer, R. (1995). Cognition - perspectives from autonomous agents. *Robotics and Autonomous Systems*, 15(1-2), 47-70.
- Reeves, C. R. (Ed.). (1993). *Modern heuristic techniques for combinatorial problems*: John Wiley & Sons, Inc.
- Scholkopf, B., & Mallot, H. A. (1995). View-Based Cognitive Mapping and Path Planning. *Adaptive Behavior*, 3(3), 311-348.
- Scholl, B. J., Pylyshyn, Z. W., & Feldman, J. (2001). What is a visual object? Evidence from target merging in multiple object tracking. *Cognition*, 80(1-2), 159-177.
- Sünderhauf, N., & Protzel, P. (2010). Learning from Nature: Biologically Inspired Robot Navigation and SLAM—A Review. *KI - Künstliche Intelligenz*, 24(3), 215-221.
- Thrun, S. (2003). Robotic mapping: a survey. *Exploring artificial intelligence in the new millennium* (pp. 1-35): Morgan Kaufmann Publishers Inc.
- Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., et al. (2007). Stanley: The Robot That Won the DARPA Grand Challenge. In M. Buehler, K. Iagnemma & S. Singh (Eds.), *The 2005 DARPA Grand Challenge* (Vol. 36, pp. 1-43-43): Springer Berlin / Heidelberg.
- Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological Review*, 55, 189-208.
- Tunstel, E. (1995). *Fuzzy spatial map representation for mobile robot navigation*. Paper presented at the Proceedings of the 1995 ACM symposium on Applied Computing, Nashville, Tennessee, United States.
- Tversky, B. (1993). Cognitive maps, cognitive collages, and spatial mental models (pp. 14-24).
- Wang, R. F., & Spelke, E. S. (2000). Updating egocentric representations in human navigation. *Cognition*, 77(3), 215-250.
- Wang, R. F., & Spelke, E. S. (2002). Human spatial representation: Insights from animals. *Trends in Cognitive Sciences*, 6(9), 376-382.
- Williams, S. B., Dissanayake, G., & Durrant-Whyte, H. (2002, 2002). *An efficient approach to the simultaneous localisation and mapping problem*. Paper presented at the Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA), Washington DC, USA.

- Wong, C. K., Schmidt, J., & Yeap, W. K. (2007). *Using a mobile robot for cognitive mapping*. Paper presented at the Proceedings of the 20th international joint conference on Artificial intelligence, Hyderabad, India.
- Yeap, W. K. (1988). Towards a computational theory of cognitive maps. *Artif. Intell.*, 34(3), 297-360.
- Yeap, W. K. (2011). *A computational theory of human perceptual mapping*. Paper accepted for publication in the proceedings of the 33rd annual meeting of the Cognitive Science Society (CogSci-2011), Boston, United States.
- Yeap, W. K., Hossain, M. Z., & Brunner, T. (2011). *On the implementation of a theory of perceptual mapping*. Paper accepted for publication in the proceedings of the 25th national conference on artificial intelligence (AAAI-11), San Francisco, United States.
- Yeap, W. K., & Jefferies, M. E. (1999). Computing a representation of the local environment. *Artificial Intelligence*, 107(2), 265-301.

## 6 Glossary

<b>ASR</b>	Abstract Space Representation. A representation of a local space, which one has visited or is currently in.
<b>Cognitive map</b>	A complex representation of the environment experienced. It includes both perceptual and conceptual information.
<b>Exit</b>	The spatial connection between two ASRs. Usually a pathway leading from one local space to the next.
<b>Local space</b>	The limited part of the environment which the individual currently resides in.
<b>MFIS</b>	Memory For Immediate Surroundings. A global, spatial map representing the environment experienced recently.
<b>Perceptual map</b>	An initial spatial representation of the environment, which is directly obtained from one's perception.
<b>Reference target</b>	A common object (surface) recognized between two different views of the same environment. New surfaces are inserted into the map relative to its position.
<b>Surface</b>	A two-dimensional line, representing geometry of the environment in the map.
<b>View</b>	The robot's view of the environment. It consists of surfaces, with their position relative to the robot.