NTNU
Norwegian University of
Science and Technology

A Hybrid Topological and Geometrical Robot Mapping Approach

Aasmund Nordstoga

Master of Science in Informatics
Submission date: June 2009
Supervisor: Keith Downing, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Robotic mapping systems are traditionally separated into the metric and the topological paradigm. The metric approach provides geometrical accuracy, but is fragile because it is bounded in an absolute coordinate system and depends on the use of odometry for navigation. The topological paradigm provides a compact representation and navigation free of accumulated error. In this thesis the topological and the metric paradigm is combined into a hybrid representation where a topological map joins a set of local maps. Each local map contains a pair of self-organizing maps, one that maps the metric space of the local map, and one that maps the perceptual space of the local map. Local navigation is performed over the SOM mapping the metric space and position correction is performed over the SOM mapping the perceptual space. A boundary-tracing behavior is used for navigation within the global map, while the local metric maps allow for more precise navigation and is navigated by performing path-integration.

Contents

List of Figures

Chapter 1

Introduction

Robotic map building is the task of autonomously generating a spatial model of an unknown physical environment and is performed by a mobile robot. Robotic map building, or robotic mapping, is often considered one of the core problems in the area of mobile and autonomous robotics. Without a sense of location in the world, globally or locally, a mobile robot will be unable to move safely in its environment, much less perform intelligible action and high level tasks in it.

1.1 The Problem

This work is motivated by the classical problem of robotic map building. The problem of robotic mapping has enjoyed a substantial amount of attention in the previous three decades within the AI community. It is widely regarded as the hardest perceptual problem in robotics, and progress in this area is expected to propagate progress in related research [53]. Although the area of robotic mapping has experienced considerable progress through the last three decades of research it is far from complete.

Robotic mapping systems are traditionally separated in two central paradigms, topological and geometrical systems. A recent trend in robotic mapping concerns the merging of these two approaches into a hybrid representation able to implement the favorable aspect of both paradigms, while omitting the less favorable aspects affiliated with each approach in isolation. The work presented in this thesis investigates the motivations that have introduced a range of hybrid approaches by discussing the attributes of purely topological and purely geometrical approaches. Existing hybrid systems are discussed and a novel hybrid mapping approach, combining topological and geometrical representations, is proposed.

Geometrical approaches represent the environment as a grid of evenly spaced cells, in which each cell represents a fraction of coordinate space in an absolute coordinate system. The cells of such grids are normally small, as most geometrical systems seek to map an environment in detail. Topological approaches represent the environment as nodes in a topological graph, in which each node represent a location in the environment and may be linked to topologically neighboring nodes , through a set of arcs with information on how to move between the connected nodes. Such information includes metrical data in some systems [5, 6], such as the angle and distance between nodes, while other systems use no metrical data to define the relations between nodes [26, 28]. The topological

and the geometrical paradigm both exhibit positive and negative attributes for use in environment representation. These attributes can be linked to a set of central issues in robotic mapping.

Odometric error Odometry is the process of estimating the pose of a robot based on the information given by wheel encoders. The wheel encoders are sensors reading the number of degrees a wheel has turned. The pose of the robot means the relative position and rotation of the robot. Odometric information can be used to continuously estimate the pose of a robot by the use of relatively simple geometric calculations. If the odometric readings given by the wheel encoders of a robot were always correct, robotic map building would be a straightforward business; this is unfortunately not the case. In the case of odometric information one problem leading to false sensory reading is wheel slippage which could happen if the surface the robot is exploring is slippery or the robot could bump into an obstacle, shifting its orientation slightly. The statistical dependence of odometric noise leads to an accumulation of error. Odometric noise is statistical dependent since noise at earlier measurements is propagated into later measurements. Consider for example a robot navigating only by integrating odometric information. If the robot should bump into a wall and this collision shifted its orientation by 2 degrees to the left its current rotation would now be 2 degrees off. Now when the unlucky robot moves i.e. 20 centimeters ahead its position will be off the estimated new position in addition to being of relative to its estimated rotation.

This accumulation of error over time, leads to a large overall error in the generated map, often rendering the map completely useless. As noise in odometric information is all but unavoidable it is not enough to use such readings in its raw form alone to build an accurate map.

The process in which odometric information is used to continuously estimate the pose of a moving robot is called path integration, also referred to as dead reckoning. Geometrical approaches depend on path integration for navigation. This makes geometrical approaches sensitive to accumulated pose error from wheel encoder readings, and makes additional corrective measures necessary. One often used correction mechanism in this context is the use of a Bayes filter for integrating perceptual data and odometric data in a probabilistic model. The Bayes filter and its use in robotic pose and map estimation is discussed in section 2.2.4 of this thesis.

The sensitivity to accumulated error in topological approaches depends on the degree of metrical information used in navigation. It is of course possible to implement topological systems that make use of path integration alone for navigation. Such systems will naturally be just as sensitive to odometric error as geometrical ones. Most topological systems however do not rely on path integration for navigation to the same degree as geometrical systems do. A typical characteristic of topological systems is the use of perceptual information and landmarks linked to the nodes of the topological graph. A landmark is a collection of perceptual cues that defines a location in an environment. Perceptual cues are not limited to the visual domain; it can entail any perceivable attribute of a location in an environment. By periodically estimating the pose of a robot through landmark recognition, topological approaches partly circumvent the problem of accumulated odometric error even if metric information is used to define the relations between the nodes of the representation. This is due to relative nature of topological

nodes. In a geometrical system each cell of the map is anchored in an absolute coordinate system, conversely, nodes in a topological map are not linked to absolute coordinates but defined by their edges to other nodes in the map alone. Because of this odometric error are not statistically dependent in topological systems where robot pose information is periodically corrected against external landmarks. Some topological approaches omit the use of metric information completely, thus avoiding the problem of odometric error altogether.

Perceptual aliasing In the problem of perceptual aliasing, physically distinct places in the environment have a similar perceptual signature, meaning they may for example look identical visually. The problem of perceptual aliasing becomes relevant if a robot need to estimate its pose in the map based on perceptual cues alone. In a scenario where several distinct locations in the environment look perceptually identical to the robot, and its only means of pose estimation is by analyzing perceptual cues, the robot could easily estimate a faulty pose, which could at best lead to a less accurate map and at worst render the robot immobile. If a robot knows approximately where it is in the map, either topologically or in absolute metric terms, the problem of perceptual aliasing becomes less relevant. This is the case in geometrical approaches. In such systems the robot continuously estimates a pose in absolute metric space. Thus it is able to separate similar perceptual location by considering its estimated pose in space. The issue of perceptual aliasing is mainly a problem in topological approaches. Topological maps are not set in an absolute coordinate space as is geometrical one, and are thus unable to rely on absolute coordinate information to separate perceptually similar locations. One method used in topological systems to counter perceptually similar locations is "simply" to increase the number of perceptual cues constituting landmarks, for example by including both visual and sonar information. Other methods may involve analyzing the perceptual properties of neighboring locations to separate similar locations.

The Correspondence problem Another well known issue in robotic mapping is the correspondence problem. The correspondence problem is that of matching a unique place in the environment at different times in the mapping process. This problem is linked with the problem of perceptual aliasing, where the problem is to differ between similar places at different locations in the environment. The correspondence problem is valid in both metric and topological approaches. In both approaches the problem usually present itself at the closing of a cycle or a loop, and the problem is therefore often referred to as the cycle or loop closing problem in literature. In metric systems the successful closing of a cycle enables the map to be corrected, but this correction must also be propagated backwards trough the map.In topological maps error is usually not allowed to accumulate freely as in metric maps, and the correspondence problem is as a consequence not as important for the building of an accurate map as in the metric approach. One consequence of cycle closing in topological maps however, are the resulting graph where nodes are closed in a loop, allowing for potentially shorter routes, conversely if cycles or loops are not closed the routes implicit in the map will be longer for nodes close to physically close but topologically unconnected places in the environment.

Environment dimensionality Another substantial problem encountered in robotic mapping is the inherent high dimensionality of a physical environment. Even small environments such as a laboratory room or an office present a large amount of features. The challenge for the robotic mapping system is then to reduce the dimensionality of the map representation sufficiently so that the computational cost stays acceptable and still provide enough information necessary for a functional representation.

Metrical approaches seek to map an environment highly accurately. Geometrical representations contain a grid of evenly spaced cells covering the entire area of the environment to be mapped. Though this allows for highly geometrically accurate maps, the cost is potentially very large in computational means. A detailed geometrical map will need to store large amounts of information for even moderately sized environment. In addition to requiring large amounts of memory, the detailed representation of geometrical systems affects the computational efficiency of robot path planning. Path planning is arguably an important aspect of robotic navigation. Path planning is the process of estimating the optimal path between two locations in an environment. The computational price of this process is decided mainly by the amount of information that is necessary to estimate a path. In geometrical systems the amount of information to be considered is potentially very large, increasing the complexity of a path planning process. However, the detailed representation of geometrical maps enables path finding algorithms to find the shortest physical path available between two locations.

Topological approaches do not map an environment in the highly accurate way of metrical maps, but rather seeks to describe the environments topological features. This approach tends to produce coarser representations of the environment, with consequently lesser memory requirements. This is because topological maps only store information of specific locations in the environment without the need to describe the space between the nodes, or locations, in the map. Though this attribute of topological systems produces compacted representations of an environment, it means such approaches must in some way ensure that a sufficient amount of topological nodes are produced to fit the complexity of an environment. This is necessary to enable the robot to move unhindered between the locations represented by nodes in the topological map.

As topological approaches stores less information than a geometrical one, path planning is normally a less resource demanding process in the topological approach as a lesser amount of information needs to be considered. Another favorable attribute of topological maps is the transferability of well known optimal path algorithms known from graph theory, such as the A* algorithm.

1.2 The Proposed Answer

The topological and the geometrical approaches to robotic mapping arguably each have problematic elements as discussed above. A number of methods have been proposed to alleviate the different weaknesses of both paradigms. Examples of relatively recent methods of this type involves the use of probabilistic methods in both pose and map estimation in geometrical maps. Although exclusive topological or geometrical approaches have been considerably improved over the last decades of research, such approaches still suffer from problems traditionally related to each paradigm.

One obvious method to meet the challenges presented by the topological and the metrical approach is to combine them.

I present in this thesis a novel robotic mapping system with a hybrid topological and geometrical map representation. The proposed system builds a topological graph representing the environment by performing environment exploration based on a boundary tracing technique. The representation is hybrid topological and geometrical as local metric maps are anchored to each node of the topological map. The merging of topological and geometrical approaches is motivated by the potential to implement the favorable elements and dampen the perceived negative elements presented by each paradigm. Hybrid approaches are not a new concept in robotic mapping; it has indeed received a great deal of attention as a means of avoiding the shortcomings of isolated metrical and topological systems. Most prominent in the field of hybrid systems are perhaps the work of Thrun [46],though many approaches exists [33, 37, 47, 54]. These systems are discussed in detail in section 2.3 of this document. What is novel in the system proposed here is dominantly the use of a self organizing map (SOM) as the representation of local metric maps and the pose estimation technique related to the local maps.

1.2.1 A Dynamical Topological Map

The map representation proposed here joins a topological map with several local metric maps. At the heart of the representation lies the topological map. The topological map is in structure a topological graph, consisting of nodes that represent locations in the environment. I refer to the topological map as the global map and the metric maps as local maps in this thesis. This is because the metric maps are anchored to each node of the topological map, and represents the immediate space around each topological node. The topological map however defines the overall mapped environment.

The global topological map is dynamically constructed. This means that the nodes of the map are created and added to the map only when significant changes in the topography of the environment are encountered during map generation. Detection of topographical changes is closely linked to the way environment exploration is performed. Exploration of the unknown environment is done by a process of boundary tracing, where the robot moves along the boundaries of the environment. Boundary tracking is implemented as a simple reactive wall following behavior, where the raw readings from the robots infrared proximity sensors are mapped directly to a left and right robot wheel speed.

The choice of a reactive wall following behavior for environment exploration is motivated by several factors. Firstly it allows for the integration of non-metric information in map navigation. This non-metric information is in the form of wall following behavior data, which is recorded in the environment exploration phase, when the map is generated. Wall following behaviors constitutes the main link between topologically adjacent nodes in the global map, and are contained in directional arcs connecting the nodes of the topological map. An arc linking two topological nodes will contain some metric information though, in the form of a distance value. The distance between two nodes is estimated from performing path integration, and is necessary to accurately define the relation between global nodes.

A second motivation for using a reactive boundary tracing exploration technique is

founded in the simplicity of operation in reactive behaviors. As boundary tracing is implemented as a direct mapping from perception to action, it is computationally fast.

1.2.2 Local Self-Organizing Maps

Anchored to each topological node of the global map is a local self-organizing map. Self-organizing maps are a type of neural network that are adjusted trough a process of unsupervised learning [55]. A Self-organizing map, or SOM, consists of two layers, the input layer and a layer containing typically a one or two-dimensional collection of topological connected nodes or neurons. A SOM is used to produce a discrete and low dimensional representation of the input space of a set of training samples. A SOM is able to keep the topological structure of the input space by the use of a neighborhood function trough the training process. This ability to retain the topological properties of the input space is the central property of a SOM, differentiating it from other types of neural networks where topological relations are not kept [55]. A more detailed description of SOMs are provided in chapter 3 of this thesis.

The SOM has attributes that makes it promising as the foundation for a robotic map. One such attribute is indeed the learning capabilities central to a SOM, the ability to topologically order and map the distribution of the input space of a set of samples in an unsupervised way. The unknown environment of an autonomous robot may be interpreted as a two-dimensional input space with a high probability for open space and a low probability for occupied space. In the proposed system the SOM is trained to statistically map the open space surrounding each global node of the global map, where the open regions are sampled trough robotic exploration in a systematic manner.

One of the main research questions this work tries to answer is how well the use of a SOM in this capacity handles noise in the sensory data. As already mentioned sensory noise is a complicating factor in robotic map building and must be handled. In the system proposed here path integration is used to calculate and continuously update the self position of the robot when mapping local environments. In a pure metric system using path integration but no position correction based on external cues to update the self position, an accumulated error would be expected due to unavoidable and statistically dependent odometric error. This accumulated error would distort the produced map increasingly relative to the distance traveled. By filtering the input data, sampled coordinates, trough a SOM, this accumulated error would still impact the resulting map, though the effect might be dampened to a certain degree in comparison to a direct metric interpretation of the sampled coordinates. In the system described here a method of self position correction over external cues is used to counter statistically dependent error. The hypothesis is therefore that a SOM will be tolerant to noise in the sampled data and thus produce a more accurate map of the environment.

Another attractive attribute of a SOM in this context is its topological structure. This topological structure, except for its use in the training process itself, can be used path planning. A standard A* path-finding algorithm, based on either Euclidian distance heuristics, or topological distance heuristics, can be applied to the map, calculating the shortest open path from the current node of the robot to a specified goal node [56].

1.2.3 Self-Organizing Maps For Landmark Storage and Self-Localization

In addition to mapping the coordinate space surrounding the global nodes of the topological map, local perceptual space is mapped as well. Mapping of perceptual space is performed through an unsupervised clustering of visual data in a self-organizing map. Visual data is sampled systematically from the local space of global nodes by the use of a camera attached to the robot. The excitation pattern of the self organizing map when presented with visual samples at a later stage can then be seen as indicating the position of the robot in perceptual space. This technique supporting self-localization by a static mapping of perceptual data in a SOM is implemented in a range of systems [26, 27, 28, 57]. In one type of systems implementing this technique [26, 27, 28] the mapping of perceptual space and the resulting excitation patterns function as an imitation of place cells found in the hippocampus of rats [49] . In the system proposed here however, the clustering of perceptual space is linked directly to coordinates in Euclidian space.

The clustering of local perceptual data in self organizing maps is motivated by the error that can be expected as a consequence of odometric drift. Although path integration is used only in part to generate robot motion between global nodes, as it is used to estimate the distance between nodes in the topological map, it cannot be discarded as a problem entirely.

1.2.4 Periodic Self-Position Correction Over Perceptual Cues

Even if limited use of metric information in the global representation is expected to dampen the effect of accumulated position error due to path integration, path integration is the sole method for navigation in the local maps of the representation and methods to counter drift error accumulated as a result of this is needed. Over a large enough time and distance such an accumulated error would grow large enough for the map to be potentially non-functional.

To account for odometric drift most robotic mapping systems using path integration for robotic navigation, either partly or fully, implement mechanisms to periodically correct this error by analyzing external perceptual data.

The approach presented in this work estimates the robots position within the map by correction over perceptual data when robot arrives at a node in the topological map, and thus the local metrical map connected to the node. As described above, the local metric map is affiliated with a number of landmarks stored in a self organizing map. By sampling perceptual space through a set of camera images and retrieving the landmark most similar to the sample from the self organizing map, the robot also receives the coordinate affiliated with the landmark and can use this to update its position within the map.

1.3 Central Research Issues

The central hypothesis underlying the work of this thesis is that a hybrid representation, merging the topological and the metrical paradigm into one representation, will provide a robust representation that draws on the strengths of both approaches while dampening the less favorable aspects affiliated with the paradigms in isolation.

The motivations for each part of the proposed system can be described clearly by reviewing them in light of a collection of paradigm related issues.

Environment dimensionality and map resolution The approach presented in this thesis provides a global topological representation joined with a collection of local metrical maps anchored to the nodes of the global topological map. The global map provides a compact representation of the overall environment, while the local maps provide higher resolution and geometrical accuracy in regions of the map of significant changes in topography.

Pose estimation and odometric drift To dampen odometric drift due to path integration, a non-metric wall following behavior is used to define the relation between nodes in the global map together with a metric distance value. By avoiding the use of a metric heading value describing the angle between nodes in the global map, the effect and indeed the presence of odometric drift should be largely avoided in global map navigation. Additionally, correction over perceptual cues is performed periodically in map navigation. The suitability and robustness of a self-organizing map for landmark storage are investigated.

In the local geometrical maps of the representation, navigation is performed by path integration. A central research issue considered in this thesis is the impact of odometric drift from navigation in the size limited space of local maps and the noise tolerance of a self organizing map as the local map representation.

1.4 Summary

Chapter 2 provides an overview into the field of robotic mapping and seeks to describe the prominent paradigms and how they handle the issues relevant to this thesis. Section 2.1 discusses the area in general and section 2.2 describes a select set of relevant systems.

Chapter 3 describes the implementation of the systems in depth. The different elements of the systems are described sequentially.

Section 3.2 describes the control system used in the robot in brief.

Section 3.3 describes the fundamental principles of the path-integration mechanism implemented in the system. Path-integration is the process of position estimation over self-motion. This process is a simple but important part of the mapping system and it is used in many part of the system.

Section 3.4 describes the parts of the system used in map generation. Firstly, the boundary tracing behavior is described in section 3.4.1. The boundary tracing behavior is used to explore unknown environments and in global topological navigation. Section

3.4.2 describes how the global topological map is created, and section 3.4.3 how local maps are created and anchored to nodes of the global map.

Section 3.5 describes how the robot performs navigation in the global topological map, and in the local maps anchored to the nodes of the global map. Section 3.53 describes how the position of the robot is periodically corrected over external cues.

Chapter 4 presents a range of experiments where the system is tested and provides analysis and discussion over the results. Section 4.1 describes the simulated environments in which the experiments are performed and how noise is simulated to provide a realistic framework for system testing.

Section 4.2 presents experiments over the map building part of the system. In section 4.2.1 the experiments testing the global mapping part in isolation is tested. Section 4.2.2 presents experiments over the complete mapping system, where local maps are created ans anchored to the nodes of the global map.

Section 4.3 presents experiments over the map navigation part of the system. In section 4.3.1 experiments where the global map is navigated in isolation is presented, while section 4.3.2 presents experiments where the complete map is navigated and section 4.3.3 presents experiments where position correction is performed periodically during navigation.

Chapter 5 sums up the project and discusses the implications of the experimental results as well as stating some thoughts on possible future revisions of the system.

Chapter 2

Background Information

Robotic map building has received enormous amounts of attention in the robotics community over the last decades. The problem of autonomously generating a reasonably accurate and usable map of a previously unknown, dynamic and otherwise complex part of the real world is widely regarded as the hardest problem in perceptual robotics. A set of core challenges has been identified through a variety of observations on the performance of robotic mapping systems, demanding new approaches to the problem.

A trend reaching back to the late eighties [1, 2, 3] has introduced the use of probabilistic methods in robotic mapping. This trend has strengthened over time, and practically all state of the art systems involve the use of probabilities in some way or another. The use of probabilistic methods in robotic mapping is motivated by the inherent uncertainties in autonome robotics, stemming from i.e. noise in sensor readings and accumulation odometric error.

Another often used classifier of robotic mapping systems are the structure of the representations, traditionally topologic or metric. Recently systems mixing both metric and topological representations have been suggested, this kind of system is called a hybrid map.

Robotic mapping research has also sought inspiration in the domain of biology. Systems built to simulate the so called cognitive maps [4] found in animals. Cognitive maps are map like spatial representations found in the hippocampus of i.e. rodents and are believed to contain information about relevant locations in an environment visited by the animal and the relation between places allowing for complex navigation. Approaches seeking to imitate cognitive maps for use in robotic mapping are closely related to topological approaches, and many topological systems inhibit elements of certain characteristics found in cognitive systems [27, 29, 28, 5, 30] without trying to simulate cognitive maps accurately.

Below I describe the history of robotic mapping in brief. In section 2.2 I discuss the different paradigms of systems in more detail and in section 2.3 I describe a number of key systems relevant to the work in this thesis.

2.1 The History of Robotic Mapping

Historically robotic maps have been split into two classes; metric and topological maps. Metric maps usually tries to represent the geometrical features of an environment in detail,

while topological maps represent the environment trough a topographically connected graph of nodes, were each node store information on how to reach the nodes it is connected to. Metric information though is usually necessary at some level also in topological maps to define the relations between nodes in the topological graph. Hybrid systems have also been proposed, mixing metric information and topological information to a greater extent than that usually found in purely topological approaches [34, 36, 37, 38, 39, 40].

An early example of a purely geometrical system is the Occupancy Grid system proposed by Elfes in 1989 [1, 2]. The Occupancy Grid approach represents the environment by mapping it as an evenly spaced grid of empty and occupied cells. A key element in Elfes approach was the use of probability profiles in the determination of occupied and empty grid cells. The classical approach used sonar readings, integrated from multiple points of view using a Bayes filter, to generate a two-dimensional map. The approach assumed that the pose of the robot was known and correct at all times, and did not consider the problem of robot localization. Occupancy Grids have been used and modified in a range of systems since it was first introduced [41, 42]. Another classical metric approach was that suggested by Chatila and Laumond in 1985 [4]. Their approach modeled objects in the environment in the form of polyhedral shapes. The model suggested by Chatila and Laumond incorporated robot position correction trough a process of "fading" where identified accumulated odometric error were propagated backwards to earlier robot positions in a non-uniform way. Other significant and early metric approaches include that of Crowley and Durrant-Whyte [43, 44]. In Crowleys approach the environment is represented as a collection of line segments and in Durrant-Whytes by specific geometric features.

An early example of a topological mapping system is that proposed by Mataric in 1990 [5]. Mataricss approach represents the environment as a two-dimensional acyclic graph of landmarks in the form of permanent structures such as walls and corridors. The nodes in the graph were added dynamically and represented a distributed map of the environment, as they functioned as concurrently acting behaviors, not unlike the nature of biological neurons ex. Path finding were implemented as a activation spreading process, where the goal node calls its neighbors and the call is propagated trough the graph, recording the direction of the call at each receiver node, to register the direction to the goal node from any node in the graph. Other early topological approaches include that of Kuipers [6] and McDermot [9].

Robotic mapping systems, topological as well as geometrical, in the last two decades have been largely united in the use of probabilistic methods. The use of probabilistic methods may be in having a probabilistic model of the environment and the robot and also for integrating sensory information into maps. One class of systems tightly connected with probabilistic methods is the systems relying on the use of a Kalman filter to simultaneous estimate the map and the position of the robot within the map [16, 17, 18, 19, 20, 21, 22]. These systems are often referred to as SLAM systems, an acronym for simultaneous localization and mapping. SLAM is the definition of a problem within robotic mapping, but has nonetheless become synonym with Kalman filter approaches in recent years. SLAM approaches are usually metric in representation representing the environment through a collection of landmark locations.

Other methods closely linked with probabilistic methods are those based on the Expectation maximization algorithm introduced by Dempster in 1977 [45].

2.2 Representation Types

2.2.1 Topological and Geometrical Maps

The difference between geometrical and topological maps is often hard to identify in a clear manner. Generally, geometrical maps represent the environment in an absolute coordinate system using evenly distributed cells which in turn forms a two dimensional grid. Topological systems however represent the environment as a graph in which the nodes represents a location in the environment. The nodes in a topological graph are linked with arcs where there exists a direct path between them. Often arcs contain metric information however, such as the direction and distance between the nodes which it connects. Systems of this kind [6, 5] can be said to be partially metric since they depend on metric information to some degree. Topological and metric approaches exhibit characteristics that can be related to vital problems in robotic mapping. One such problem is the perceptual aliasing issue, concerning the ability to separate perceptually similar but spatially distinct places. Geometrical approaches, such as the Occupancy Grid method [1, 2], continuously updates the position of the robot by a process of path-integration. The robot knows its location at all times within a global and absolute coordinate system and it can thus separate spatially distinct location on this information alone. This of course, assumes that the self position of the robot is updated accurately. Under this assumption it can also recognize spatially related places even with differing sensor readings. This is the Correspondence problem, usually linked with the closing of a loop in the map. Topological methods however are generally not anchored to absolute coordinates in the same way metric approaches are. The position of the robot is computed relative to the map based on perceptual cues in the environment. This fact means that topological approaches may have problems in separating places that looks alike but are in fact not. Topological systems usually implements additional procedures to reduce the perceptual aliasing problem. One approach [12] combines local topology information and local landmarks to disambiguate perceptually similar places, other approaches rely on perceptual information from neighboring locations as well as local metric information [7, 8].

The assumption that the position of the robot position in metric systems is always accurate is naturally difficult to satisfy. Metric models must therefore constantly try to correct error in odometric readings due to phenomena such as drift and slippage, as is the case in most SLAM implementations where Kalman filters are employed to statistically estimate the robots position [22, 21, 16, 17, 18]. Topologically approaches are usually not as sensitive to this kind of error, because they do not need to operate on the exact pose of the robot at all times. This attribute, exhibited by most topological approaches, allow them to avoid the accumulated error suffered by metrical systems.

Another attribute of the typical metric approach is that of resolution. As geometric maps cover the entire environment to be mapped, so to say, the grid cells of the map need to be small enough to provide a sufficiently accurate model of the world. The grid-based structure of metric maps means that they keep a more geometrically correct representation of the world than topological maps, or in other words a higher resolution of the environment. The high resolution of a geometrical map may be needed in certain cases, but it also makes the memory and efficiency of the map larger. For large maps this may be catastrophic. Topological maps in relation contain a far more compact representation. In

topological approaches the resolution of the map tends to correspond to the resolution of the environment. This is because nodes in the graph are usually dynamically added based on some requirement, which may i.e. be a maximum distance between nodes or some kind of novelty detection method, adding nodes only when a sufficiently different region of the environment is discovered. The compactness of topological approaches reduces the memory requirement for the representation itself, but it also makes path planning faster and shortens the divide between symbolic inference methods and the map representation.

In path planning however, metric approaches usually allows for the computation of shorter paths between places in the environment than topological ones, because of their high resolution. Topological path planning approaches must operate on the nodes in the graph only, and the arcs connecting them. This is usually remedied to some degree by closing loops in the graph, though such methods assume a well functioning solution to the correspondence problem.

As both approaches provide favorable attributes, hybrid approaches have been proposed to take advantage of attributes from both paradigms.

2.2.2 Hybrid Topological and Geometrical Approaches

A relative recent development in robotic mapping research has introduced approaches combing topological and metric mapping frameworks in order to retain the positive attributes of both paradigms and compensate for the negative ones.

Hybrid approaches combine the paradigms in varying ways. In a system proposed by Thrun [46] adopts the Occupancy Grid approach in building a metric maps of the environments and a topological map is posterior computed from the metric map. Approaches of this kind inherit the positive attributes of topological maps, but they also inherit negative elements from the metric approach. Because the topological map is merely extracted from the metric representation it will be affected by accumulated odometric error present in the metric map, thus compromising the consistency of the topological map as well as the consistency of the geometrical map.

In [47] the environment is represented as a collection of local metric maps, all of which are anchored to nodes in a global topological map. The node in the topological graph of this approach, records a set of landmarks visible from the boundary of each obstacle discovered by the robot, and uses this visual only information in later navigation between nodes in the graph. This approach avoids robot pose estimation entirely.

This is to some degree similar to the approach in [34] which also connects a set of local metric maps in a global topological map. In the latter approach the local maps are represented as a set of infinite lines and topological nodes are connected to the local maps by a necessary distinct transition point. This transition point is indicated by a stored metric feature and is used to relocate the robot correctly in the local map. The metric maps themselves are built and navigated by the use of an Extended Kalman Filter. In the topological navigation process a Partially Observable Markov Decision Process is used for state estimation.

Systems like the two mentioned above manages to limit the effect of accumulated error found in purely metric maps relying heavily on odometry to construct a global coordinate system representation of the environment. This is because the metric maps are merely local in nature and long distance navigating is performed over topological maps

not anchored to a global coordinate system. They do however enjoy higher precision in the area of each node, as local metric maps exist at each node.

2.2.3 Biologically Inspired Approaches and The Cognitive Map

The approach to the mapping problem in autonome robotics can be formulated as a question of how to represent a physical environment in an abstracted simplified way and what elements to store in such a representation. It is natural and perhaps unavoidable, too view this question in relation to how we humans, as well as animals and insects navigate and map environments successfully.

The considerable amount of research done on biological navigational systems has yielded results not only interesting to neuroscientists but also to computer scientist striving to build robust artificial navigation systems for mobile autonomous robots. Animals such as rats, and indeed humans, seem to inhabit both robust and scalable systems for the purpose of navigation in the usually complex real world. Although robotic mapping has seen steady progress through the last decades, state of the art artificial systems can hardly be compared to biological ones when it comes to performance in simple as well as complex and dynamic environments.

The problem interesting both researchers of cognitive science, neuroscience and researchers within the area of robotic mapping is the how and why of how spatial information is stored in the brain of animals. There have been several experiments trying to simulate biological mapping systems [29, 31, 32, 30, 33, 38, 39] and many artificial robotic mapping systems have been built partly inspired by the findings in biological mapping research [27, 29, 28]. Biological mapping systems have evolved into robust and impressive standards, arguably outsmarting state of the art artificial systems by far. The motivation behind studying such systems is quite clear from the computer scientist s perspective; if the processes behind biological mapping systems were well understood it would be possible to build artificial copies and use them in autonomous robots. Although the processes underlying the mapping capabilities in biological systems are not yet fully understood there have been a lot of research done and interesting findings have inspired a variety of artificial systems.

The notion of a cognitive map is that of an abstract spatial representation of the environment generated through experiences of the environment over time. In a series of behavioral experiments done on rodents in 1976 by OKeefe [49] a type of cell named place cells were discovered in the rat hippocampus. These place cells were found to fire when the rat moved into certain areas of the environment, seemingly independent of the rats orientation. OKeefe named the area in the environment which modulated the firing of a place cell the most the place field of the place cell. It was discovered that place cells were not restricted to represent one environment, but could indeed represent several, though with changes in the relationships between the firing fields in the neurons [51].

Soon after the discovery of place cells another element was found, thought to be part of the cognitive map processes in the rat brain, namely head-direction cells [40]. These cells were found to discharge when the rats head were pointing in a certain direction.

In 1978 OKeefe & Nadel suggested that place cells provide the core elements of a distributed and noncentric map. They also suggested that this map provided the rat with a dynamic and continually updated representation of the environment in allocentric space

and the position of the rat in this space. Recent findings within neuroscience research on the navigational system of rats suggest the process of path integration is central to the initialization and maintenance of so called grid cells which are found in the entorhinal cortex of the rat [52]. There is also evidence suggesting the existence of a general metric navigational system of which these grid cells may be a part of [52]. There have been and still is discussion on the role of metric and topological information in biological navigation systems. Central is the question of how much quantifiable metric information is stored in biological cognitive maps and how much is contained in topological form. There is no conclusive answer to this question thus far, but many different models are proposed and explored [29, 31, 32, 33, 38, 39].

Usually the dominant influence of biological research on artificial systems concerns simulations of place cells. In an approach suggested by Owen and Nehmzow [26, 28] a self-organizing feature map is proposed as the map representation. In this system a robot uses the SOFM to cluster sensory data obtained in an exploration phase. The excitation pattern of the network when provided with a perceptual sample from the environment locates the position of the robot in perceptual space. Route learning is realized by associating control input from a joystick with the "place cells", or nodes, in the network during a route learning phase. Another approach by Mataric, mentioned already in section 2.2.1, focuses on simulating the distributed nature of biological mapping systems by simulating biological neurons by implementing the nodes in her graph as concurrently simulation behaviors [5].

2.2.4 Probabilistic Approaches

Probabilistic approaches to the problem of robotic mapping and navigation have become the core unifying factor of most robust mapping algorithms in the last two decades. On the reason why probabilistic approaches are beneficial Thrun in [24] states; "A robot that carries a notion of its own uncertainty and that acts accordingly, will do better than one that does not".

The probabilistic approach span a lot of process in total, but one can separate the appliance of these into two main categories.

The first category is that of robotic perception. Robotic sensors are flawed by noise and the inherent uncertainty of complex environments. Alternative to producing a single uncompromising quantity from censor readings the use of probabilistic methods allow for the computation of a probability distribution over the observed data, resulting in a more consistent integration of sensor data and ambiguity handling.

The second category is that of robotic control. Because an autonomous robot cannot rely on perfect sensors it cannot justify an assumption that it knows exactly how an environment is structured and where it is in that structure. Probabilistic approaches operate by integrating this uncertainty when making control decisions.

Bayes rule and Bayes filter Practically every probabilistic approach to the robotic mapping problem builds on variations of a central principle in statistical inference, Bayes rule.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.1}$$

In equation 2.1 $P(A)$ defines a prior probability of A, $P(A|B)$ states the conditional, or posterior, probability of A given B, $P(B|A)$ is the conditional probability of B given A and $P(B)$ is the prior probability of B. Bayes rule allow for the specification of a probability $P(B)$, stating the probability of B prior to the introduction of other data which is multiplied with the probability of observing a variable d under a hypothesis A, $P(B|A)$, this again gives the probability of observing a variable A given a condition B.

Bayes rule is most notably adopted for simultaneously robot pose and map estimation in a Bayes Filter:

$$\begin{aligned} z^t &= z_1, z_2, ..., z_t \\ u^t &= u_1, u_2, ..., u_t \end{aligned} \tag{2.2}$$

where in example z^t is a sensor measurement z taken at time t, and u^t is a robot motion command sampled at time t. The data is used in a recursive Bayes Filter:

$$p(s_t, m|z^t, u^t) = \eta p(z_t|s_t, m) \int p(s_t|u_t, s_t - 1)p(s_t - 1, m|z^{t-1}, u^{t-1})ds_{t-1} \tag{2.3}$$

where m denote the map and s the robots pose.

Equation 2.3 describes a Bayes filter that estimates the map and the robot pose simultaneously and assumes a static environment to simplify the estimation. The recursive nature of a Bayes filter gives it the favorable ability to compute updates in constant time, allowing for incremental integration of sensor readings and motion data for map and pose estimation.

Many robotic mapping algorithms have been derived from the Bayes filter. Most notable are perhaps Kalman Filter approaches [22, 21, 16, 17, 18]. Kalman filter approaches are often referred to as SLAM approaches, an acronym for Simultaneous Location and Mappping. In SLAM the robot pose and the map is estimated simoultanesly. Kalman filters are Bayes filters which make the assumption that posteriors $p(s_t, m|z^t, u^t)$ can be represented by Gaussian distributions. The Gaussian model in Kalman Filter approaches is a state vector x, constituting a robot's pose s and the map m:

$$x_t = (s_t, m) \tag{2.4}$$

The robots pose in Kalman filter approaches is usually modeled by three variable, the Cartesian coordinates in the environment and the heading of the robot while the map is usually represented as the coordinates of a set of features, i.e. landmarks.

Other approaches derived from the Bayes filter includes the EM approach, The Lu/Milos approach and Incremental ML among others.

2.3 Related Approaches

2.3.1 A Hybrid Approach by Tomatis, Nourbakhsh and Siegwart

The hybrid approach proposed by Tomatis, Nourbakhsh and Siewgeart in [33] is an excellent example of a hybrid topological and geometrical approach for map building and localization, where the metric and topological modules are completely separated in two levels of abstraction. In the approach a global topological map connects a collection of local metric maps. The topological map consists of a set of nodes with edges to other nodes and metrical places in the environment. Metrical places are modeled by local metrical maps, which consist of infinite lines that belong to the same place.

To travel between a topological node and a metric location require the presence of a detectable metric feature to determine the transition point where the system changes from the topological to the metric paradigm. Transition from metric navigation to topological navigation is performed purely through metrical navigation. The metric approach in the system utilizes an Extended Kalman Filter for localization and adopts the Stochastic Map approach for metric map building while topological navigation uses a partially observable markov decision process for state estimation.

Tomatis, Nourbakhsh and Siewgeart empirically validate their approach through a range of experiments described in [33]. The notable contribution of their system is the compact and computationally efficient representation gained from merging the topological and the metric paradigm, while allowing for precise metrical local representations and a robust global representation by integration of the POMDP approach in topological navigation.

2.3.2 A Hybrid Approach by Thrun and Bucken

In [58], Thrun and Bucken propose to combine the topological and the metric paradigm to gain the accuracy and consistency of metric maps and the efficiency of topological maps in a hybrid representation. Central to the approach is the extraction of topological maps from grid-based maps. The metric approach used is in the form of occupancy grids, proposed in [58]. Thrun and Bucken use artificial neural networks trained by back propagation to estimate the occupancy values of the occupancy grids which input is a collection of sensor readings. Bayes rule is adopted to estimate occupancy probabilities over time while the robot pose is estimated through a process of gradient descent over pose probability densities acquired from wheel encoder data, map correlation data and wall orientation data. A topological map is extracted from the occupancy grid based map by partitioning the free-space of the grid map into a set of regions, separated by critical lines, which correspond to narrow passages in the environment. The extraction process is done in a series of steps, where the cells of the occupancy grid are first separated into occupied and unoccupied space by thresholding and then transformed into a voronoi diagram. From the voronoi diagram a set of critical point is identified. Critical points are points from the voronoi diagram that minimize clearance locally. Finally critical lines connecting the critical lines are constructed and the partitioning mapped into an isomorphic graph, regions of which correspond to topological nodes and critical lines to

arcs.

By extracting topological maps from metric maps, the different locations of the topological map are naturally disambiguated and nearby locations naturally identified. Thrun and Bucken show in a series of experiments described in [58] that planning is several orders of magnitude more efficient in their hybrid approach than in purely metric maps, while the decrease in performance due to loss in precision was negligible. The approach does however suffer from odometric drift over large distances as the base for the resulting topological map, an occupancy grid map, relies heavily on path integration for navigation.

2.3.3 Atlas

Atlas is a hybrid topological and metric approach to SLAM proposed by Bosse et al. in [35]. In Atlas the environment is represented in a graph of coordinate frames, where each vertex in the graph corresponds to a local frame and edges between vertexes represent the transformation between adjacent frames. Each frame contains a map of the local fraction of the environment. This map captures the uncertainties of the map itself, as well as the uncertainties of the robots pose with respect to the local coordinate frames. The uncertainty of the edges in the graph is derived from the output of a SLAM algorithm that is the method used in local mapping. The computational complexity of each local map is measured and this complexity measure is not allowed to exceed a defined threshold.

Atlas is intended as a generic framework in which a variety of small-scale SLAM algorithms may be integrated. The central motivation behind Atlas was the various problems presented by the complexity and potential robot pose errors due to the use of path-integration driven navigation in existing SLAM approaches employing a single, globally referenced coordinate frame for state estimation. By joining several local coordinate frames in a topological graph, Atlas is able to restrict representations of errors to local regions, without affecting the entire map.

2.3.4 Huang and Beevers

In [59], Huang and Beevers propose a topological approach to the map building problem with respect to sensing-limited robots. Their approach utilizes a topological map with edges defining a set of wall-following behaviors that describe the path between topological locations, or the nodes of the topological graph. A wall-following behavior is used to create an initial map of an open area. As corners and similar well defined features are detected by changes in the wall-following behavior, vertices are added to the topological graph until the robot returns to the initial point of departure and the loop may be closed. Because the mapping process requires a return to the initial robot position to terminate, the algorithm assumes an enclosed environment. Each vertex of the graph is given a probability distribution, which is monotonically growing as the map expands, that model the odometry error of the robot with respect to the estimated pose in relation to the initial robot position. Loop closing is performed through a process of hypothesis evaluation. A initial hypothesis of a closed loop event is created upon odometric information. The initial hypothesis is then evaluated through a evidence-based approach that compares characteristics of subsequent edge traversals.

Additionally Huang and Beevers introduce a concept of portals and forays. Portals

are special links connecting open space maps, built through wall following, to corridor like spaces. Forays are special links connecting open space nodes by crossing open spaces to create more direct links than that produced by wall following alone. Huang and Beevers approach is mainly motivated by the loop closing problem in robotic mapping. To better support correct loop closing and consistent maps, the system features a probabilistic modeling of error and hypothesis evaluation. Furthermore, the use of portals implies that loops can be identified and closed in local environments, lowering the scope of error related to each loop.

2.3.5 Owen and Nehmzow

The system proposed in [26] by Owen and Nehmzow is an interesting example of a robotic mapping system inspired by the cognitive map found in rats. The approach builds a topological map of the environment by mapping the perceptual space of the environment. This mapping is done by systematically sampling robot range sensors readings from a set of locations in the environment. The range samples are fed to a self organizing map, which in turn clusters the samples through an unsupervised clustering process. Each node in the trained SOM can then be linked to a specific location in perceptual space, enabling the robot to perform self-localization by sampling perceptual space and retrieve the closest matching node from the trained SOM. To enable route following, Owen and Nehmzow associate motor actions with nodes of the trained SOM, a process in which the robot is controlled manually and actions retrieved from a joystick used to control the robot.

Ownen and Nehmzow show in a range of experiments that their approach consequently navigated successfully over repeated experimental runs in fairly complicated environments. The approach stores no metric information in its representation and the relevance and irrelevance of perceptual features is decided autonomously through a process of unsupervised clustering of perceptual data. This use of sub-symbolic information alone to map an environment is favorable in that no predefinition of perceptual features is needed, it requires however human interaction for route learning purposes in this approach.

Chapter 3

Implementation

3.1 The Simulated Robot

The mapping system presented here is implemented and tested on a simulated e-puck robot in the Webots simulator. The Webots simulator is a 3 dimensional simulator that simulates all features of the physical robot.

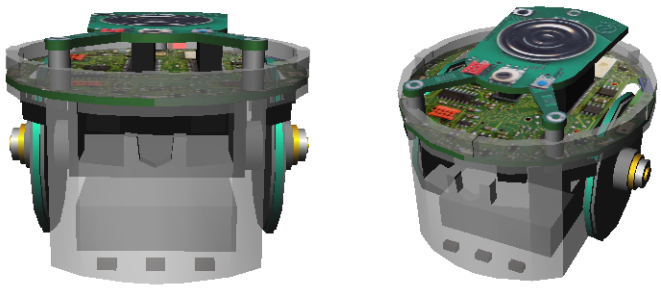


Figure 3.1: The E-Puck Robot

3.1.1 Sensors

The e-puck comes with a range of sensors. Below I list and outline the sensors the implementation in this thesis makes use of.

Differential wheels and wheel encoders The e-puck robot is set up with two differential wheels for locomotion. Because the wheels are connected to a differential mechanism they are able to rotate at different speeds, and this mechanism is naturally used to turn the e-puck in different directions. The wheels are also connected to wheel encoders, which are sensors that record the number of wheel rotations. This information can be used to estimate the relative direction and distance traveled by the robot over time, this process is referred to as odometry.

The wheel encoders are used to perform path integration in the system proposed here, a process that is further discussed in later sections of this thesis.

Infrared proximity sensors The e-puck features a set of 8 infrared proximity sensors, distributed along the circular frame of the robot. These sensors estimates the distance to a physical obstacle by measuring the infrared light radiating from objects in its field of view. To supplement the IR sensors, which are passive receiving sensors only, the e-puck employs a ring of LED light sources that can be used in concert with the IR sensors to intensify the radiation of nearby objects.

The proximity sensors of the e-puck is used in the processes of obstacle avoidance and wall following, both of which are discussed in detail later in this document.

Camera The e-puck also has a color frontal camera, with a maximum resolution of 640*480 pixels. The camera is used to retain visual landmark information in the approach presented here

Other sensors Besides the sensors described above the e-puck robot is set up with a 3d accelerometer, which can be used to estimate changes in three dimensions. It also features three microphones for sound recording.

3.2 The Control System

Robot control systems have been a hotly discussed subject in the field of autonomous robotics for the last three decades. Although it is not the focus this thesis it constitutes an important part of the complete system, and I will describe its design and motivation in brief.

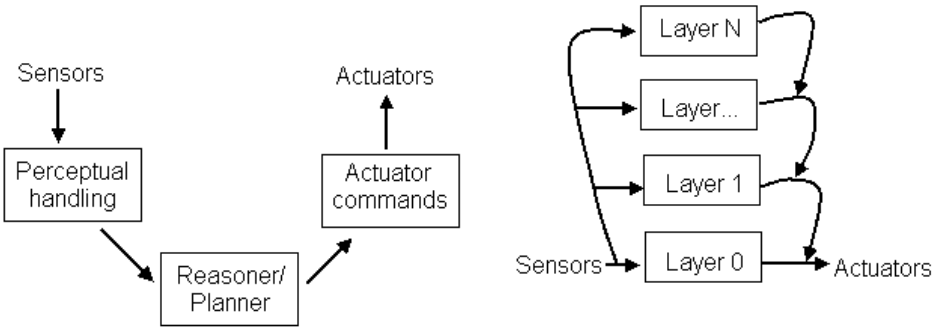
The problem of robot control concerns the issue of how to connect sensory percepts to the capabilities of the robot and on to actuator commands. Traditionally this problem is viewed as that of decomposing a set of tasks into either a vertically or horizontally layered set of modules. Each module contains some form of competence, which it uses to generate or propagate actions from perceptual input or input propagated from other modules in the system. This type of systems can be separated into three main categories; reactive, hybrid and hierarchical systems.

The control structure chosen for the system proposed in this thesis is a horizontally layered hybrid system. To clearly describe the motivations for this choice I discuss the prominent alternatives in brief.

3.2.1 Control System Paradigms

Hierarchical systems were the dominant paradigm in early robotic mapping. In hierarchical systems percepts travels through the entire set of modules before generating any action. Figure 3.2(a) illustrates this process. The hierarchical approach is characterized by its horizontally organized architecture, where the path between sensory input to the effectors are decided by the number of modules in total, increasing complexity and time consumption relative to the capabilities of the robot.

Arguably the main drawback of the hierarchical control architecture is the need for a centralized modeling and planning mechanism. After each perception received, modeling based on the received percepts is done, after this planning is performed based on the



(a) an hierarchical architecture (b) Brooks' Subsumption architecture

Figure 3.2: Control system architectures

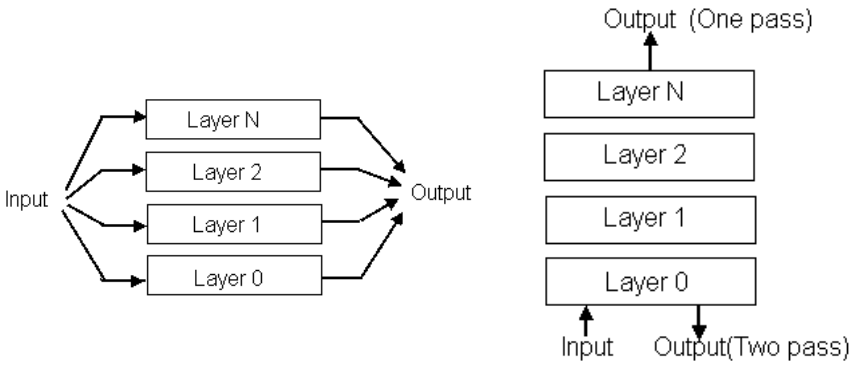
updated model. The hierarchical architecture was designed with a strong focus on robot planning in mind, and must be judged with that in mind. Hierarchical architectures was largely abandoned for use in autonomous robots with the introduction of reactive and hybrid architectures in the 1980s.

One typical and often used example of the reactive paradigm is the subsumption architecture of Brooks [52]. Brooks approach were founded on a set of theses stating that intelligent behavior is possible without the explicit representations of symbolic AI and without the explicit abstract reasoning of symbolic AI and finally that intelligence is an emergent property of complex systems.

The subsumption architecture, illustrated in figure 3.2(b), introduces a vertically organized architecture. The modules of the system are implemented as independent behaviors, each of which is directly connected to the sensory input and each of which is able to produce action output by mapping perceptual input directly into actions. Each behavior was assumed to contain no complex symbolic information and be responsible for some particular task. Additionally the layers in the subsumption architecture presented by Brooks were in the form of finite-state machines. Another characteristic of Brookss subsumption architecture is that all behaviors can fire simultaneously. Brooks proposed a method of controlling which action to choose from the actions proposed by each behavior that relied on the behaviors being set in a subsumption hierarchy. In this system lower level behaviors can inhibit behaviors on higher levels in the hierarchy, thus prioritizing the lower level behaviors over the higher ones. Higher level layers are assumed to represent more abstract behaviors while lower level layers are assumed to represent the more important behaviors for operational robustness, such as obstacle avoidance behaviors. The higher level layers goals subsume those of underlying layers. The subsumption and inhibition mechanisms of this system allows the lower level layer to operate in a reflex like manner, while the higher lying layers work to achieve some overall goal.

The largest advantage of the subsumption architecture in its original form perhaps, is the modularity of the approach, allowing for iterative development and testing of real-world systems. It is also computationally tractable, the action selection function in Brooks proposed system has a worst case of $O(n^2)$ where n is $max(behaviors, percepts)$. The subsumption architecture and other purely reactive control architectures however suffers

from some fundamental problems yet unsolved. Reactive architectures assume that the local environment alone provide enough information to generate a sensible action, as they contain no explicit model of the environment. This assumption is a large one and arguably not possible to satisfy for large and potentially complex tasks. Another limiting factor connected to the way decisions are made based on local information only, is the inability to include non-local information in the decision process. Another issue is that of emergent behavior in reactive systems. The overall behavior of a purely reactive system can be said to emerge from the interaction of the modular and simple behaviors of the system. Such emergence of behavior could certainly be favorable in many situations, but it is hard for a designer to implement a system meant to perform specific tasks, by relying on the emergence of certain behaviors.



(a) Horizontally layered system (b) Vertically layered system

Figure 3.3: Hybrid control systems

Hybrid control architectures seek to unite reactive approaches with proactive abilities. They do this by handling reactive behaviors and proactive behaviors in separate subsystems of the architecture. Hybrid systems usually separate these subsystems into layers which are able to interact in some way. There are two central hierarchies in which to place such layers, a vertically layered architecture and a horizontally layered architecture.

Both horizontally layered and vertically layered systems are advantageous over purely reactive ones in that they are able to handle reactive behaviors as well as proactive behaviors such as performing planning over a model of the environment. This ability means that reactive behaviors such as obstacle avoidance can be executed rapidly if the environment changes suddenly for example, while explicit abstracted behaviors such as following a path retrieved from a map representing the non-local environment can be executed when there is no immediate obstacles in front of the robot.

3.2.2 The Horizontally Layered Control System

I use a horizontally layered hybrid control system including a mediator function in my approach. As can be seen from figure 3.4, the system contains 5 vertical layers, each connected indirectly to perceptual input through a mediator function and each able to generate action output. Additionally the structure contains a sixth layer, which is not connected to the mediator function, the path-integrator.

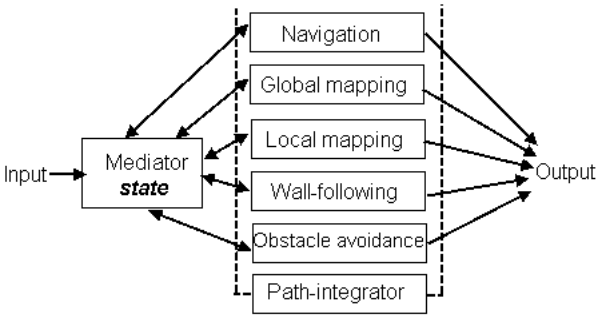


Figure 3.4: The horizontally layered control system

As discussed in the above sections, the different control system paradigms have certain advantages as well as disadvantages over each other. My choice of a horizontally layered hybrid paradigm was motivated by the following key points

1. both reactive and proactive behaviors
2. conceptually simple
3. Iterative development and testing of layers
4. Limited interaction between layers

While the hierarchical paradigm only supports proactive behaviors and the purely reactive approach only supports reactive behaviors, the hybrid paradigm supports both these behaviors. This is the main reason I chose the hybrid system instead of a purely reactive or a hierarchical approach. Reactive behaviors are arguably favorable in that they map perceptual input directly to action, without the need for an intermediate reasoning process. This exclusion of an intermediate reasoning stage allows for fast execution of motor commands, which is necessary in certain situations such as obstacle avoidance. I implement two reactive behaviors in my system, obstacle avoidance and wall following, both of which are described in detail in the following parts of this section.

Proactive behaviors are also favorable. Such behaviors can operate on non-local data as well as local data. A system using reactive behaviors exclusively relies on emergent overall behavior as discussed above. A system with abstract behaviors with access to non-local information does not rely on the emergence of overall behavior, as the overall behavior can be explicitly programmed into the representation itself.

Horizontally layered hybrid systems are conceptually simpler than vertically layered ones. This is because horizontal approaches contain independent layers, all of which generate their own action output based on some input. In contrast vertical layers are interconnected as either input has to be propagated through the layers of the architecture in one pass systems, and additionally action output is propagated back through the layers in two pass systems. This propagation of input and output through layers suggest that the designer of the layers must consider how this propagation is to be implemented and how to translate input and output signals between the different layers. In a horizontal system however, the layers are not connected in this way, and the designer needs not consider translation between layers, but can focus on the actual functionality of each separate layer.

The independence of the horizontal layers is also helpful in regard to the development process itself. Independent layers can be iteratively developed, and iteratively tested. In the case of a horizontal system the interaction between layers must be considered at the development of each layer in a larger degree than in vertical approaches, because each layer must contain a function to handle propagation of input and additionally output in two pass systems.

The horizontal approach needs a mediator function to decide which layer is to be in control at any time. The vertical approach does not require this, as inherently only one layer can be in control at any time in vertical approaches. As previously mentioned the need for a mediator function leads to a larger worst case number of layer interactions the designer of the systems needs to identify. This, of course, is a negative aspect of horizontally layered systems. I prioritized the conceptual simplicity and iterative nature of a horizontal system over the potential complexity introduced by a mediator function.

3.3 Path Integration

Path integration in robotics is the process in which a number of movement cues are continuously integrated to estimate the position of a robot. This process is also called dead reckoning. Path integration is continuously performed during both the mapping phase and the mapping phase in the system described in this thesis. Because the process is both fundamental to the operation of the system and at the same time exist as a layer of its own in the control system, I describe its implementation in detail in this section.

To perform path integration it is necessary to consider the geometry of the robot and the actuator system it uses. The e-puck robot is a differential drive robot, meaning it produces motion from turning two independent wheels. These wheels can rotate either backwards or forwards and are connected to a pair of wheel encoders that measure the angle of each wheel periodically. The wheel encoders use this measurement of wheel angle to count the number of steps a wheel has turned. Each step of rotation is set to a specific fraction of a turn of a wheel. If a wheel rotates backward this counter counts down, if it rotates forward it counts up.

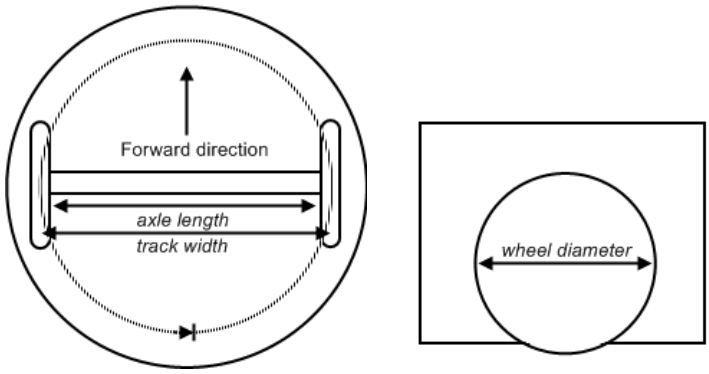


Figure 3.5: Geometrical properties of the e-puck robot

The distance traveled by a wheel in one step of rotation depends on the number of steps in one full revolution of the wheel and the diameter of the wheel.

$$distancePerStep = \frac{\pi * diameterWheel}{stepsPerRevolution} \tag{3.1}$$

The calculation of the distance traveled when the robot moves in a straight line is then straightforward.

$$\Delta distance = \frac{leftSteps + rightSteps}{2.0} * distancePerStep \tag{3.2}$$

To estimate the heading of the robot after a rotation both the track width and the diameter of the wheels must be considered. First the number of steps per in place full rotation of the robot is calculated. This rotation equals $2 * \pi$ radians, which translates to 360 deg.

$$stepsPerRotation = \frac{trackWidth}{wheelDiameter} * StepsPerRevolution \tag{3.3}$$

We can rearrange equation above to estimate a new heading based on the wheel encoder readings.

$$radiansPerStep = \frac{\pi * \frac{wheelDiameter}{trackWidth}}{stepsPerRevolution} \tag{3.4}$$

$$heading = (rightSteps - leftSteps) * radiansPerStep \tag{3.5}$$

Equation 3.5 gives us the new heading in radians. However, both the distance and heading equations above assumes that the robot only moves in straight lines and rotates in place. To continuously estimate the position and heading of the robot when its following an arbitrary path it is necessary to handle the robots total motion in many small and discrete movements, which are continuously accumulated to estimate the robots actual position. This technique assumes that each discrete movement does not contain large variations in heading. This means that the wheel encoders must be sampled in sufficiently small intervals. The required minimum sampling rate is decided by the speed of the robot, if it moves and turns quickly a faster sampling rate is needed than at lower speeds. To estimate the position over small discrete movements, the results from equation 3.2 and 3.5 is used in equation 3.7.

$$\begin{aligned} \Delta X &= \Delta distance * \cos heading \\ \Delta Y &= \Delta distance * \sin heading \end{aligned} \tag{3.6}$$

ΔX and ΔY are then added to the previous position estimate.

$$\begin{aligned} \Delta X_t &+= \Delta X_{t+1} \\ \Delta Y_t &+= \Delta Y_{t+1} \end{aligned} \tag{3.7}$$

The complete path integration algorithm is described in pseudo code in appendix 2.

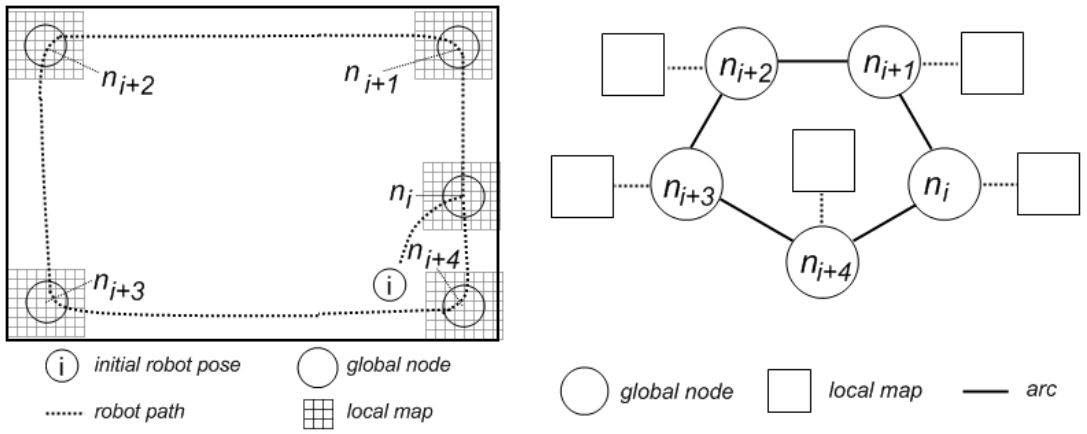
3.4 The Mapping Process

The map building process in the system presented in this thesis consists of two separate mapping operations, global and local mapping. The process of global mapping is responsible for creating a quantitative global map of the environment, where areas of sufficient change in topology in the environment are represented as nodes in a topological map.

Each node in the global topological map is linked to a qualitative local map, representing the immediate local space surrounding a node in the global map.

The overall mapping process can be decomposed into a series of steps:

1. Explore the environment through boundary-tracing until a sufficient change in environment topography is detected.
2. Create a new global node n_i and connect it to the previous node n_{i-1} if such a node exists.
3. Map the metric and perceptual local space surrounding global node n_i and anchor the resulting local map to global node n_i.
4. Repeat the process from step 1 until a loop is closed or a maximum number of global nodes is created.



(a) An example mapping process (b) The resulting topological graph

Figure 3.6: An exemplified mapping process

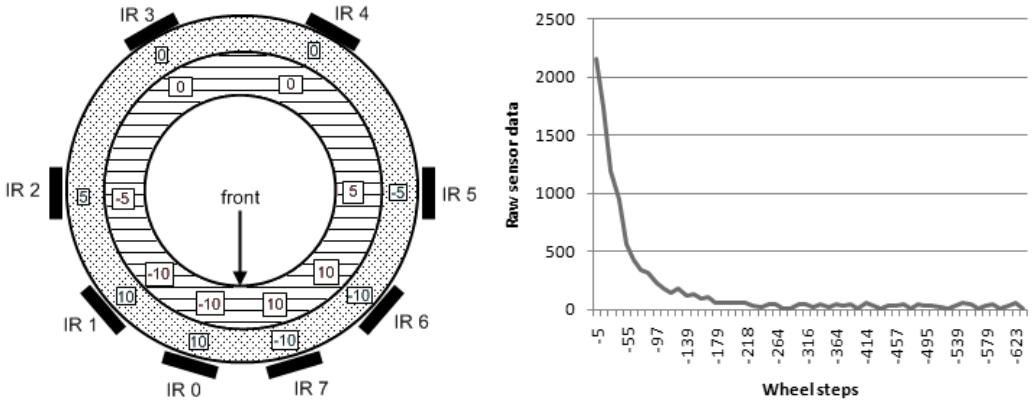
Figure 3.6(a) shows an example of the mapping algorithm in an enclosed rectangular room. The robot detects the change in topography at the corners of the room and creates global nodes at these locations. Local metric maps are also created and anchored to the respective global nodes. Figure 3.6(b) shows the resulting closed graph produced by the mapping algorithm. The nodes of the graph are topologically connected via arcs, these arcs contains information on how to navigate between the nodes of the graph.

The remainder of this section describes the central elements of the mapping process in detail.

3.4.1 Environment Exploration by Boundary Tracing

The global mapping process employs a boundary tracing routine to build a topological representation of the environment. Boundary tracing is realized through the wall-following behavior.

The wall-following behavior is implemented as a purely reactive behavior. It is purely reactive because the raw distance information acquired from the IR sensors of the robot is mapped directly to a left and right wheel speed. The mapping is performed through a simple multiplication of raw distance information with an array of distance biased weights.



(a) IR sensors on e-puck

(b) Raw sensor readings

Figure 3.7: E-Puck proximity sensors

Figure 3.7(a) displays the ring of IR proximity sensors equipped on the robot. The patterned rings in the figure shows a series of example weights related to each sensor. These weights are used to modulate the proximity signal from each sensor in a way that produces a wall following behavior. The outermost ring of weights governs the speed of the right wheel, while the innermost ring governs the speed of the left wheel.

Figure 3.7(b) shows the sensory data from infrared sensor 0 as the robot gradually backs away from a wall. The x-axis show the number of steps the wheels has turned as the robot goes backward, 628 steps equals 12.5 centimeters. It is seen in the figure that at approximately 160 steps, translating to 3.2 centimeters, from the wall the distance is too far for the sensors to discern.

The wall-following behavior computes a left and right wheel speed based on the proximity data from the IR sensors and matrix of weights, each of which is connected to a specific IR sensor. The matrix I contains the eight values of the IR sensors:

$$I = [IR_0, IR_1, IR_2, IR_3, IR_4, IR_5, IR_6, IR_7] \tag{3.8}$$

There are two matrixes of weights, one for computing the left wheel speed of the robot L, and one for computing the right wheel speed of the robot R:

$$\begin{aligned} L &= [-10, -10, -5, 0, 0, 5, 10, 10] \\ R &= [10, 10, 5, 0, 0, -5, -10, -10] \end{aligned} \tag{3.9}$$

To allow for bidirectional navigation between nodes, a process described in section x, two types of wall following behavior is implemented; a left-side behavior and a right side behavior. The left-side behavior aligns the left side of the robot with a wall, while the right-side behavior aligns the right side of the robot with a wall.

The difference in alignment is produced by adjusting the weights connected to IR sensor 0 and IR sensor 7. In the case of left-side behavior this can be written:

$$\begin{aligned} L[0] &= 10 \\ L[7] &= 10 \\ R[0] &= -10 \\ R[7] &= -10 \end{aligned} \tag{3.10}$$

the inverse of the values used above is used in the right-side behavior:

$$\begin{aligned} L[0] &= -10 \\ L[7] &= -10 \\ R[0] &= 10 \\ R[7] &= 10 \end{aligned} \tag{3.11}$$

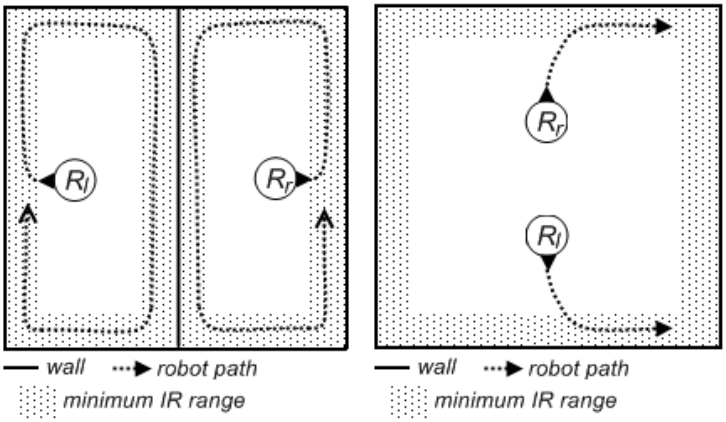
The speed of the right and the left robot wheel respectively denoted by s_r and s_l, is calculated by:

$$\begin{aligned} s_l &= b + \sum_{0 \leq i \leq 7} L[i] * \frac{I[i]}{16} \\ s_r &= b + \sum_{0 \leq i \leq 7} R[i] * \frac{I[i]}{16} \end{aligned} \tag{3.12}$$

where b is a bias speed value.

When any of the robot's sensors are within the minimum IR proximity range, meaning it is close enough to a wall for its IR sensors to detect it, the calculations of robot wheel speed in equation 3.12 is valid. If, however, none of the robot's sensors are within the minimum IR proximity range the robot will either circle towards the left or the right, depending on the type of wall-following behavior engaged, until any of its sensors detect a wall. If right-side behavior is used the speeds are calculated by:

$$\begin{aligned} s_l &= b \\ s_r &= \frac{b}{2} \end{aligned} \tag{3.13}$$



(a) Within minimum proximity range (b) Outside of minimum proximity range

Figure 3.8: One robot performing left-side behavior, R_l, and one the right-side behavior, R_r

and if left-side behavior is used:

$$\begin{aligned} s_l &= \frac{b}{2} \\ s_r &= b \end{aligned} \tag{3.14}$$

where b is a bias speed in both equations.

Figure 3.4.1 exemplifies both types of wall-following behavior within and outside of the minimum IR proximity range. Please see appendix 5 for a complete wall-following algorithm.

3.4.2 A Dynamic Topological Map

A dynamical topological map is chosen as the core representation of mapped space in the system proposed in this thesis. Dynamical in this context means that the nodes of the topological map are created during the mapping process, when a certain set of requirements are met. The topological map contains both global information, in the form of topological nodes, and local information, in the form of geometrical maps assigned to each global node.

Figure 3.9 illustrates the conceptual architecture of the global map representation. The representation consists of a linear list containing what I refer to as global nodes. Each global node represents a location in the environment. A global node may contain a set of arcs connecting it to other global nodes. These arcs keep wall behavior information and a distance value, defining the relation between the connected nodes.

Nodes

Global nodes are the core units of global topological map. A global node defines the topography of a mapped environment through its links to other global nodes. A global

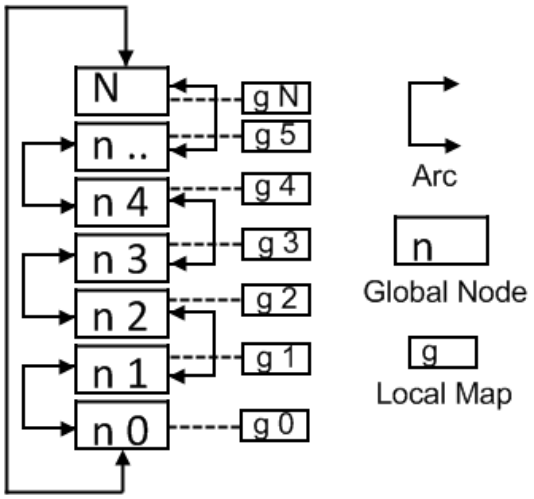


Figure 3.9: A global topological map. Global nodes are connected by arcs and contain a local map each

node contains three attributes, a node id, separating it from other nodes, a collection of directional arcs, connecting it to other nodes and a local metric map.

GLOBAL NODE	
Node ID	[0-∞]
Arcs	<a1,a2,...aN>
Local Map	local map

Figure 3.10: Global node

Generation of nodes Generation of new nodes in the global map is executed whenever the global mapping algorithm detects a significant change in local topography. The global mapping algorithm can detect topographic changes by auditing path-integration data generated by the wall following behavior, which is responsible for environment exploration, and recording the change in distance and rotation from the previously created node to the current position of the robot.

A change in topography, significant enough for a new node to be created, is decided by the Euclidian distance from the previous created node to the current position of the robot, as well as the variation in robot direction. Specifically, for a new node to be created, the distance must be greater than a specified distance threshold, and the variance in rotation must be greater than a rotational threshold. The choice of threshold values decides how detailed an environment is topologically mapped. The smaller the choice of threshold values, the greater the amount of global nodes.

Figure 3.11 illustrates the creation of a new topological node. At the initiation of the mapping process an initial node is created at the starting position of the robot. No odometry information is recorded prior to this event, and the direction of the robot is therefore set to 0 radians and the initial coordinate position to 0.0.

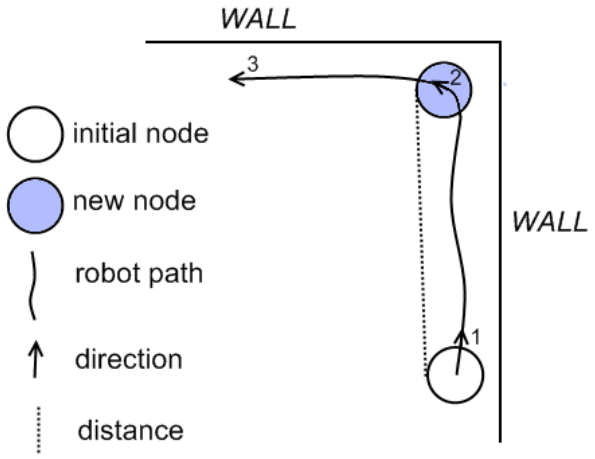


Figure 3.11: A new node, marked in blue, is added to the global map. The node is added because the Euclidian distance between the initial node and the new node position is larger than a defined distance threshold, and the angular difference between the heading related to the initial node(1) and the heading related to the new node(2) is larger than a defined angular difference threshold.

The position of the robot is continuously estimated by the path integration process, described in section 3.3, as the robot begins exploration of the environment through the wall following process described in section 3.4.1. The global mapping algorithm preserves the direction and position values from the point upon which a previous node was created. At the beginning of the global mapping process, the previous node corresponds to the initial node. By continuously comparing the pose linked to the previous node with updated odometry data, a change in direction and the distance traveled are estimated.

The relative angular difference between the previous node pose and the current pose is calculated by equation 3.16 where the previous node heading and the current heading are kept in the two headings:

$$P^{\circ}, C^{\circ} \tag{3.15}$$

$$angularDifference = |wrapvalue(P^{\circ} + 180 - C^{\circ}) - 180| \tag{3.16}$$

The function *wrapvalue* used in equation 3.16 wraps any value greater than 360 around.

The distance between the previous node pose and the current pose is calculated as standard Euclidian distance, where the previous node pose coordinate and the current pose coordinate is kept in the two points:

$$P(p_x, p_y), C(c_x, c_y) \tag{3.17}$$

ARC	
From node	n...N
To node	n...N
Wall-following behavior	<left-side,right-side>
Metric distance	[0-∞]

Figure 3.12: An arc with possible values

$$distance = \sqrt{(p_x - c_x)^2 + (p_y - c_y)^2} \tag{3.18}$$

To decide when a new topological node is to be added to the global map, the estimated traveled distance and angular difference is continuously compared to a pair of threshold values. This process is conceptually very simple, as illustrated in algorithm 1.

Algorithm 1 Generate new node

1: **while** mapping globally **do**
2: $\theta = angularDifference(P^\circ, C^\circ)$
3: $distance = distance(P(p_x, p_y), C(c_x, c_y))$
4: **if** $\theta > thresholdAngle \&\& distance > thresholdDistance$ **then**
5: Add new node to map
6: $P^\circ = C^\circ$
7: $P(p_x, p_y) = C(c_x, c_y)$
8: **end if**
9: **end while**

Arcs

The arcs linking the global nodes of the topological map describe the relation between the nodes, that is, how to navigate between each pair of linked nodes. In the system proposed here, this information is in the form of a wall following behavior and a distance value. By recording such information during map generation, the robot can navigate the map at a later stage.

Figure 3.12 show the structure of an arc. An arc is a directional link between two nodes, and therefore contains a reference to the node from which the connection goes and a reference to the node in which the connection ends. An arc also contains information on how to navigate between the connected nodes. Information needed for navigation consist of a combination of a wall-following behavior and a metric distance value.

Navigation based on wall-following As described in section 3.4.1, wall following behaviors come in two types, a left-side and a right-side behavior. In right-side behavior, the robot aligns its right side to a wall before following it in the forward direction, conversely, in left-side behavior; the robot aligns its left side to a wall before following

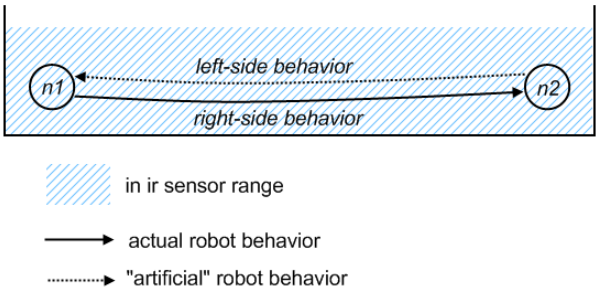


Figure 3.13: The wall-following behaviors defining the relation between two global nodes.

it in the forward direction. The attributes of the two types of wall-following behavior enable the system to link global nodes bidirectional in a single pass.

Figure 3.13 illustrates how two global nodes, $n1$ and $n2$, are linked with wall-following behaviors. In figure 3.13 the robot moves by following its right side wall from node $n1$ to node $n2$, which it creates according to the process described above. For the nodes to have any topological meaning, their relations must be described, and for the robot to be able to navigate between them at a later stage, this description must contain information on how to move between them. The entire robot motion path between node $n1$ and $n2$ in figure 3.13 lies within the minimum proximity range of the ir sensors, meaning that direct wall-following was performed to move between the nodes.

The relation from node $n1$ to node $n2$ is thus described by the right-side wall-following behavior and a metric distance between the nodes, estimated through path integration. The relation information is kept in an arc object, linked with node $n1$ When the robot later needs to move from node $n1$ to node $n2$ it executes right-side wall-following behavior until it has moved equal to the distance stored in the distance value of the arc.

To enable bidirectional navigation of the global map, it is necessary to describe all relations between topologically connected nodes, regardless of how and in what order they are connected. In figure 3.13 the robot only moves from node $n1$ to node $n2$, but in later navigation ir must be able to also move from node $n2$ to node $n1$. To describe the relation between node $n2$ and node $n1$ another arc must be created and linked to node $n2$. This arc will contain the same distance value as the arc found in node $n1$, but the wall-following behavior will be the inverse of that found in node $n1$, a left-side wall-following behavior.

3.4.3 Local Maps Anchored to Global Nodes

Anchored to each global node of the topological map are localized metric and perceptual representations in the form of self-organizing maps. Localized maps are meant to provide accurate representations of interesting areas in the environment, the immediate space surrounding each global node in this representation. The global nodes represent locations in the environment of change in topography, while the space between the global nodes is uniform in the sense that it ideally contains empty space and approximately straight boundaries. It is therefore contended in this approach that the immediate space surrounding global nodes are inherently more complex than the intermediate space between global nodes, and would gain from a more accurate representation, both in terms

of overall map accuracy and in terms of local navigational capability.

A local map consists of two self-organizing maps, one for metric mapping and one for perceptual mapping of the space surrounding a global node. Before describing the local map in detail, it is necessary to introduce the concept of a self-organizing map.

Self-Organizing Maps

Self-organizing maps(SOM) are types of neural networks which performs unsupervised topological clustering of sample data through a learning mechanism. The learning process of a SOM is done in such a way that the nodes of a SOM become ordered in relation to each other, which leads to a mapping of a high-dimensional continuous signal to a lower-dimensional topological representation of the input signal, thus the topological properties of the high-dimensional input signal is preserved in the lower-dimensional representation. A SOM consist of one layer of nodes and is typically one or two-dimensional in structure. Figure 3.17 shows the basic structure of a hexagonal two-dimensional SOM, with each node connected topologically to six immediate adjacent neighbors.

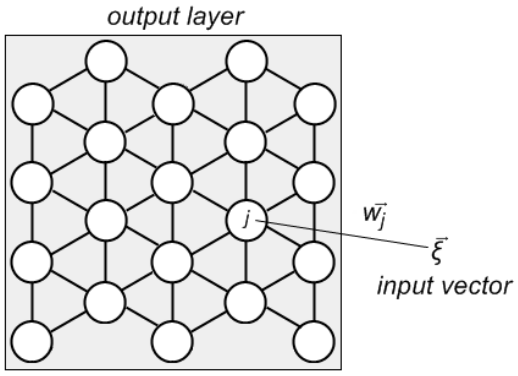


Figure 3.14: A hexagonal SOM

In figure 3.17 an input pattern is represented in the vector:

$$\vec{\xi} = [\xi_1, \xi_2, \xi_n] \in R^n$$

Where i represent an input sample and n the nth element of the input pattern.

The input pattern is presented to each node in the output layer of the map. Additionally each output node is connected to a weight vector:

$$\vec{w_j} = [w_{j1}, w_{j2}, w_{jn}] \in R^n$$

Where j is the identificator of a node j in the output layer and n is the nth element of the input vector.

The core training process of a SOM can be separated into three steps:

1. The weight vectors connected to each output node is initialized with either randomized values, or an estimate of the expected input distribution. Each value of the weight vectors are then normalized in the range of $[0, 1]$

2. A random input pattern $\overrightarrow{\xi}$ is drawn from a collection of input patterns and normalized in the range of $[0, 1]$. The Euclidian distance between the input vector and the output of each weight vector connected to the output layer nodes are calculated to identify the node closest to the presented input pattern by means of Euclidian distance.

   The output o_j of node j is the weighted sum of the inputs to node j:

   $$\sum_{k=1}^{n} w_{jk} i_k = \vec{w_j} \cdot \vec{i} \tag{3.19}$$

   where n is the number of elements in the input vector.

   The winning node p is given by:

   $$p = min(||\vec{\xi} - \vec{w_j}||), j = 1, 2, 3.., K \tag{3.20}$$

   where K is the total number of cells in the output layer.

3. The values of the weight vector connected to node p and the weight vectors of a set of topologically neighboring nodes to p are adjusted according to equation 3.21.

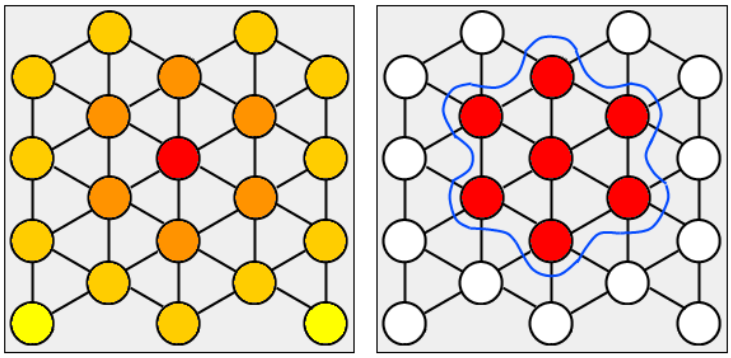
   $$\begin{aligned} \Delta p_{jk} &= \alpha(i_k - p_{jk}) \\ p_{jk}(t+1) &= p_{jk}(t) + \Delta p_{jk} \end{aligned} \tag{3.21}$$

   where α is a learning rate parameter in the range $[0, 1]$, that starts out large and decreases with time t. The number of nodes to adjust neighboring a winning node are usually also decreasing from an initial neighborhood size that may include all the nodes of the output layer before degenerating to the winning cell p only, after a certain number of time-steps.

   If no defined time-step or convergence limit is reached, the process is repeated from step 2.

As the algorithm described above iterates the network will organize into a structure in which similar input vectors clusters onto specific regions of the network, while dissimilar input vectors map onto different regions of the map.

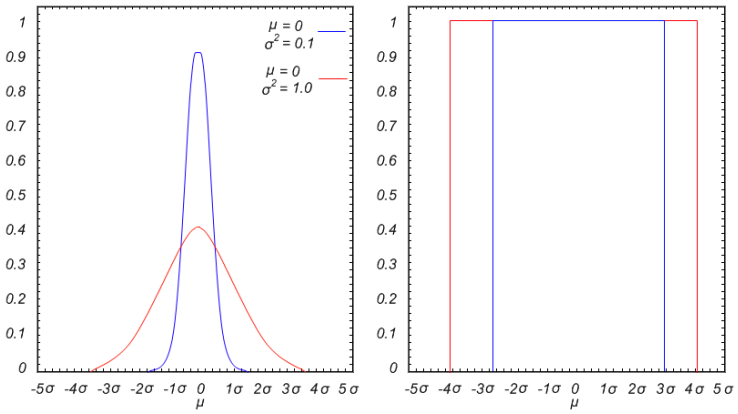
There are a some parameters to be considered when using a SOM, most important are perhaps the neighborhood function and the learning rate.



(a) A Gaussian neighborhood in a hexagonal SOM. The center node in red illustrates the winning node. The degradation into yellow color symbolizes the decreased impact of the winning node.

(b) A bubble neighborhood in a hexagonal SOM. The bubble function updates in this case the six closest nodes in the map. The constant red color symbolizes that the bubble function is constant over the defined neighborhood. The blue line outline the neighborhood size of the exemplified bubble function, the size being two in this example

Figure 3.15: A Gaussian and a bubble neighborhood function.



(a) Two Gaussian probability density functions (PDF). This distribution is also referred to as the normal PDF

(b) Two bubble function curves.

Figure 3.16: Function curves

Neighborhood function A neighborhood function decides to what degree the nodes neighboring a winning node should be adjusted. In this system I implement two differing neighborhood functions, a Gaussian and a bubble neighborhood function.

The Gaussian neighborhood function can be written:

$$\exp(-(\frac{||r_c - r_p||^2}{2\sigma^2(t)})) \tag{3.22}$$

where r_c is the topological location of node c in the output layer and r_p is the location of the winning node p. $\sigma(t)$ is the neighborhood radius at time t. Figure 3.15(a) shows how the impact of the Gaussian neighborhood function decreases outwards from the winning node.

The bubble neighborhood function is an approximation of the Gaussian function. Instead of a Gaussian gradual decrease of neighborhood impact outwards from the winning node, the bubble function is a constant function in the neighborhood of the winner node, thus every node within a neighborhood at time t is updated by the full difference between the winner node's weight vector and the input vector, while all nodes outside of the neighborhood is not updated at all. An example of a bubble function's impact is illustrated in figure 3.15(b).

Learning rate The learning rate is another parameter affecting all node updates, as seen in equation 3.21. The learning rate is typically in the range $0.1 - 0.9$, the learning rate being initially large before decreasing through the SOM training process. The manner in which the learning rate decreases can vary. In this system I implement two types of learning rate functions, a linear function and a power series.

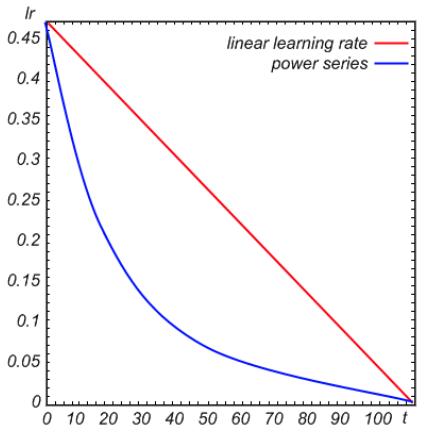


Figure 3.17: A linear learning rate function and a power series learning rate function

The linear learning rate function is simply a gradual and small linear decrease of the learning rate; typically a minimum learning rate is defined.

The power series can be written:

$$\alpha_0 \left(\frac{\alpha_T}{\alpha_0} \right)^{t/T} \tag{3.23}$$

where α_0 is the initial learning rate, a_T the minimum learning rate, t is the time step and T the total number of time-steps of decrease.

Both the learning rate and the neighborhood-function impacts how rigid or fluid a SOM behaves in the topological ordering process, and must be tuned through experimentation to fit the variables of individual cases in which the SOM is applied. In brief, such variables typically include the number of input patterns available, the level of generalization needed and more.

Local Metric Maps

In the system presented here, local metric maps are anchored to each of the nodes of the global map and represent the immediate two-dimensional metric space surrounding the global nodes. The robot samples the unoccupied metric space and the perceptual space surrounding each global node by exploring the local environment in a systematic manner, an exploration process in which navigation is performed trough path-integration alone.

The mapping of the local space surrounding a global node is performed directly after a global node is created. The size of the area to be mapped is set before the mapping process begins. I refer to the surrounding space of a global node to be mapped as "local space".

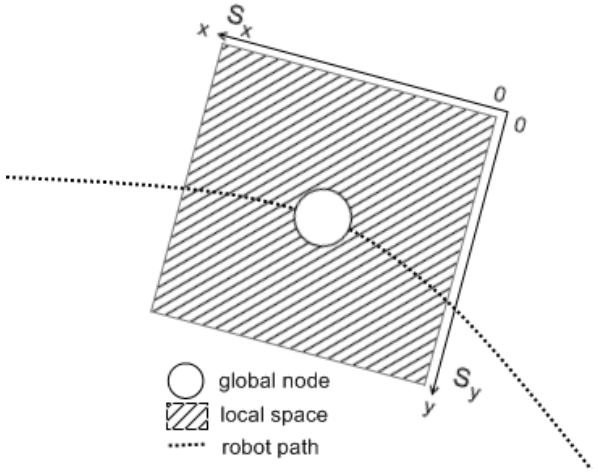


Figure 3.18: The local space of a global map.

Figure 3.18 shows the local space of a global map. The size of the local space to be represented in a local map is defined by a length value, S_x and a height value, S_y.

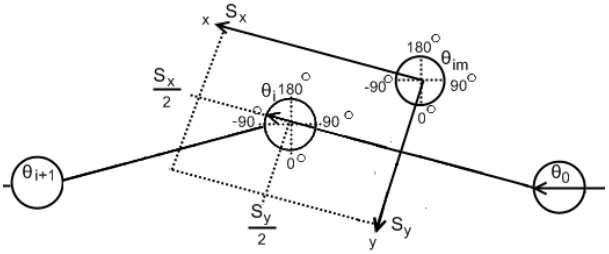


Figure 3.19: The angular orientation of the local coordinate system, $\theta_i m$, equals that of the angular heading of the robot as it created the global node, θ_i.

A local space is bounded in an absolute coordinate system. The orientation of this coordinate system is equal to that of the robot's angular heading upon creation of the global node the local space surrounds. Figure 3.19 shows three global nodes with a θ value related to each. This value represents the angular heading of the robot as it created the global nodes, I refer to this as the entry direction. The local SOM anchored to global node i in figure 3.19 is outlined in the figure. The local space is bounded in a coordinate

system with an orientation equal to the entry direction of the node i. To generalize the relation between the entry direction of a global node and the orientation of the coordinate system of the related local space we can state for node n_i and the related local space m_i:

$$\theta n_i = \theta m_i \tag{3.24}$$

The coordinate system related to a local space is also positioned so that the center of the global node equals the center point of the local space to be mapped. This is illustrated in figure 3.19 where the center of the global node is equal to the center of the local space. The center point S_c is defined by:

$$S_c = [\frac{S_x}{2}, \frac{S_y}{2}] \tag{3.25}$$

where S_x is the length of local space and S_y the height of local space.

Exploration of local space The systematic exploration of local space employs a temporary rigid topological graph, with nodes representing a collection of coordinate locations in local space and arcs connecting adjacent locations. The rigid topological graph used for exploration covers the local space of a global node in a uniform manner, as the nodes of the graph are evenly distributed over the local space. To provide a uniform distribution of the nodes with respect to the hexagonal neighborhood setup, every other column of nodes is offset in the y-direction according to:

$$o_i = \frac{s}{2}$$

where o_i is the offset of node i and s is the size of the spacing between nodes.

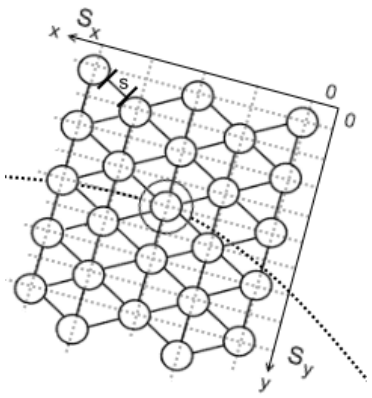
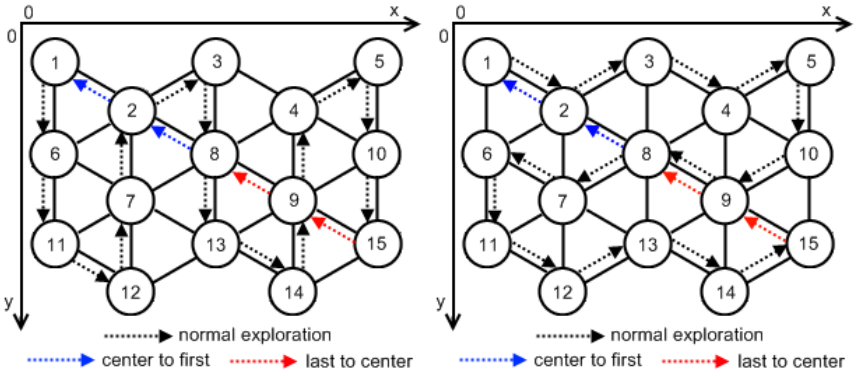


Figure 3.20: A temporary rigid map used for systematic exploration of local space

A rigid topological coordinate graph is only a temporary construction used in exploring a newly created local space. Figure 3.20 illustrates a rigid graph overlaid a local space. Implied by the grid in the figure, each node contains a coordinate in local space.

The reason why such a topological graph is used is that it provides a framework for a systematical exploration of local space. The robot is able to explore the local space in its entirety by moving from node to node in the rigid graph until it has visited all nodes. The movement from node to node is controlled by path-integration only, a process described in section 3.3.



(a) Column by column exploration (b) Row by row exploration

Figure 3.21: Exploration strategies

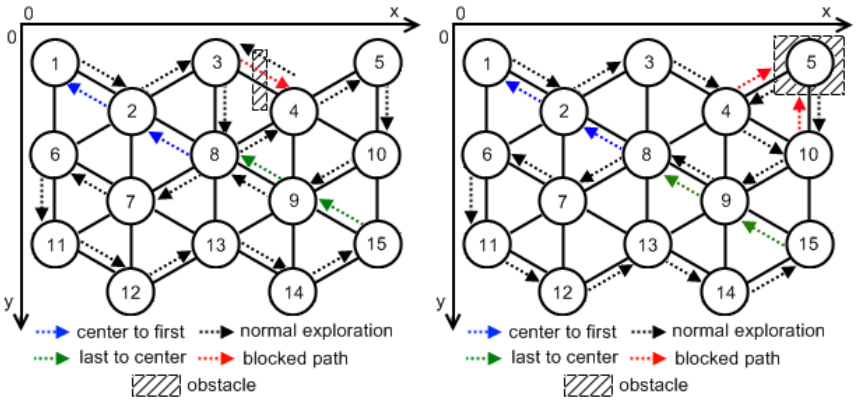
The middle node of the rigid graph is always centered over the global node which local space it maps. Because of this the robot is initially positioned at the middle node of the rigid graph. In the exploration strategies in figure 3.21 the robot will begin the exploration process by moving to the first node of the graph. The first node is defined as that with the initial weight vector $\vec{w_j} = [0, 0]$, and the path to the first node is illustrated by the blue arrows in figure 3.21. The red arrows in figure 3.21 illustrate the path leading back to the center node from the last reached node, which is node 15 in this example.

Both exploration strategies illustrated in figure 3.21 show the exploration path of the robot in a local environment with no obstacles. To handle obstacles blocking the path a method of topological path finding is applied. The method use a modified A* algorithm to find the shortest path between two nodes in a graph. The A* algorithm is a best-first, graph search algorithm that finds the optimal path between graph nodes by using a heuristic function to determine the order in which to search the graph, written as:

$$f(x) = g(x) + h(x) \tag{3.26}$$

where $g(x)$ is the actual shortest distance traveled from the initial node to the current node and $g(x)$ is the estimated distance from the current node to the goal node. For a more detailed description of the A* path-finding algorithm please see appendix 3

Figure 3.22 illustrates how the A* algorithm is employed to generate a new path when an arc between two nodes is blocked by an obstacle. In figure 3.22(a) an obstacle blocks the arc between node three and four. By use of the A* path finding algorithm a new path is generated to node four via node eight. In figure 3.22(b) the arc between node four and five is blocked by an obstacle, and a new path via node ten is generated. The arc between node ten and five is also blocked by an obstacle and since there are no more potentially



(a) A blocked path between node 3 (b) Node 5 isolated by a an obstacle and node 4

Figure 3.22: Paths after performing path-finding

open paths to node five, it is marked as unreachable and the robot will move to the next potentially reachable node of the exploration strategy, node ten in this example.

The purpose of exploring the space covered by a rigid graph in a systematic manner is to sample and map the metric and the perceptual space of that area.

Sampling and mapping of metric local space The metric representation of local space is in the form of a SOM. The metric SOM, as I shall call it, bears superficial similarities to the rigid graph used in local space exploration. However, the metric SOM is not a temporary construction; it is a representation of the unoccupied space surrounding a global node.

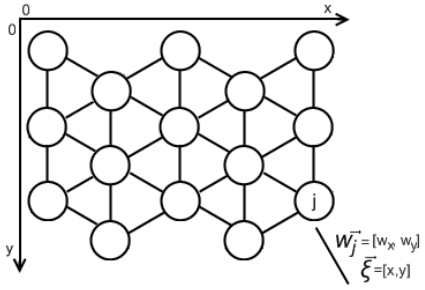
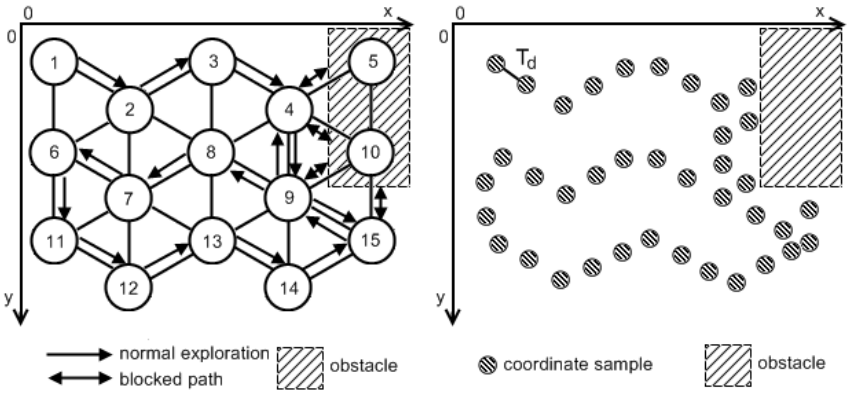


Figure 3.23: A metric SOM in its initial configuration

Alike the rigid graph, a metric SOM initially naively maps local space uniformly. Each node in the output layer of a metric SOM is connected to a weight vector $\vec{w}$ containing two weights, one for each value of the input vector $\vec{\xi}$:

$$\begin{aligned} \vec{w} &= [w_x, w_y] \\ \vec{\xi} &= [\xi_x, \xi_y] \end{aligned} \tag{3.27}$$

where ξ_x represents the $x-position$ and ξ_y the $y-position$ of an Euclidian coordinate sample.



(a) The path of a robot executing the column by column exploration strategy

(b) The coordinates sampled by the robot during exploration

Figure 3.24: The resulting robot path and sampled coordinates after exploration of local space

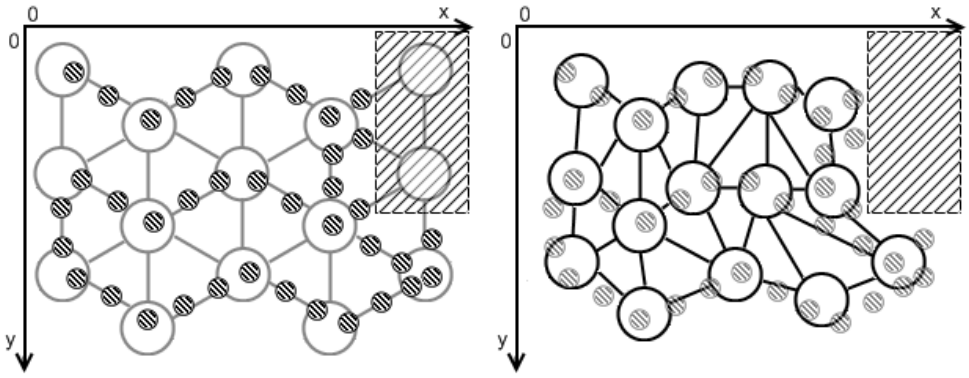
The robot samples coordinates from the local space by simply recording its position, generated by path-integration, at certain intervals during the exploration of a local space. The length of such an interval is defined a distance threshold denoted:

$$T_d \tag{3.28}$$

The robot will record coordinate samples each time it has moved for a Euclidian distance equal to T_d from the last sampled coordinate.

Figure 3.24 shows the path of the robot after exploring a metric SOM and the trail of coordinate samples that was sampled during the exploration. As illustrated in the figure, coordinates are sampled at distance intervals equal to T_d.

After a metric SOM has been explored and sampled, the metric SOM is trained over the recorded samples by the SOM training algorithm described in section 3.4.3.



(a) The SOM before training

(b) The SOM after training

Figure 3.25: The metric SOM before and atfer training

Figure 3.25 shows a metric SOM that is trained over coordinate samples recorded

during exploration. The nodes of the trained SOM are now distributed over the coordinate samples, and essentially provide a metric mapping of local unoccupied space.

Sampling and mapping of local perceptual space To support perception-based localization, the perceptual space surrounding global nodes are mapped in addition to the metric space. The mapping of perceptual data is not unlike the sampling of metric data, both processes employ a SOM for unsupervised sample clustering. In the metric mapping approach these samples consist of two-dimensional coordinate information, while in perceptual mapping the samples consist of camera images.

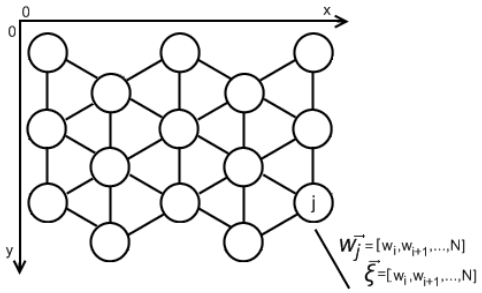


Figure 3.26: A perceptual SOM

The nodes of the perceptual SOM are connected to a weight vector $\vec{w}$ containing N weights, one for each value of the input vector $\vec{\xi}$.

$$\begin{aligned} \vec{w} &= [w_i, w_{i+1}, ..., w_N] \\ \vec{\xi} &= [\xi_i, \xi_{i+1}, ..., \xi_N] \end{aligned} \tag{3.29}$$

where ξ_i represents i'th pixel of an image sample and N the last pixel of an image sample.

Sampling of perceptual data is performed in parallel to the sampling of coordinates. The sampling rate of camera images is related to the sampling rate of coordinates by:

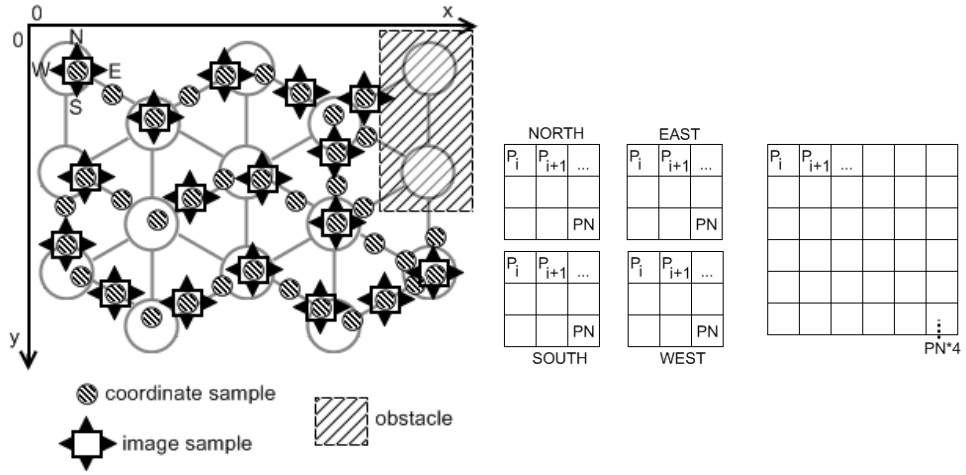
$$T_d * 2 \tag{3.30}$$

where T_d is the distance threshold defining the sampling rate of coordinates. Thus, camera images are sampled at half the rate as that of coordinates.

Image samples consist of four separate camera images, sampled in the four cardinal directions; north, east, south and west. Figure 3.27(a) shows how camera images are sampled during the exploration of local space. The arrows in the figure represent the four cardinal directions at which camera images are sampled. Figure 3.27(b) exemplifies the merging of four images sampled in these directions into a single image sample.

After a perceptual SOM has been explored and sampled, the metric SOM is trained over the recorded samples by the SOM training algorithm described in section 3.4.3.

After training is complete, the nodes of a perceptual SOM will exhibit different excitation patterns when presented with visual input patterns, indicating the position of the

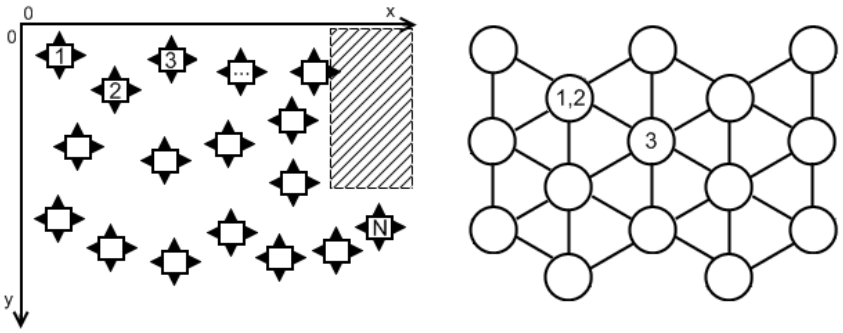


(a) Sampling of camera images and coordinates

(b) Four images in the cardinal directions merged into one image sample

Figure 3.27: Sampling of perceptual space

input patterns in perceptual space. To enable association between visual input patterns and metric coordinates, the coordinates at which the perceptual samples used to train a perceptual SOM were captured, are related to the nodes of the perceptual SOM.



(a) Perceptual samples

(b) The trained perceptual SOM

Figure 3.28: Association between the coordinates of perceptual samples and the nodes of the trained perceptual map

The association of coordinates with the nodes of the trained perceptual map can be described in the following algorithm assuming perceptual SOM of N nodes, and a collection of K visual samples:

1. Calculate the winning node n_w of input pattern $\vec{\xi_{i+t}}$ by:

$$n_w = min(||\vec{\xi_{i+t}} - \vec{w_j}||), j = 1, 2, 3.., N \tag{3.31}$$

2. Associate the coordinate of input pattern $\vec{\xi_{i+t}}$ with node n_w. If a coordinate is already associated with node n_w, calculate the middle coordinate of the existing and the new coordinate by:

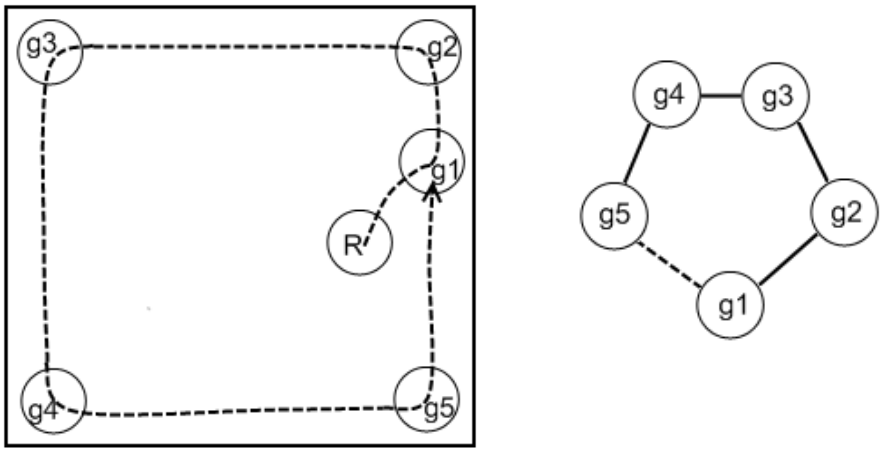
$$n_c = [\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}] \tag{3.32}$$

3. Advance input pattern count, $t + 1$. Repeat from step 1 until $\vec{\xi_{i+t}}$ equals $\vec{\xi_K}$

Figure 3.28 exemplifies how the coordinates of visual samples are associated with nodes in a perceptual SOM trained over those visual samples. The figure illustrates how visual sample 1 and 2 are associated with node i of the perceptual SOM and visual sample 3 are associated with node j of the perceptual SOM.

Loop-Closing

Loop-closing is the process of closing a loop in the topological map by connecting nodes that are close to one another. The mapping algorithm in this system assumes a closed environment, and uses a wall-following behavior in exploration. Due to these attributes of the mapping procedure, the robot is expected to revisit any node of the global map given exploration over a sufficient time. Loop-closing in topological maps are favorable mainly because the result is a more complete graph, allowing for shorter paths in later navigation.



(a) Global node revisitation (b) The topological graph.

Figure 3.29: Robot exploration path leading back to the initial node

Figure 3.29 illustrates how the exploration path of the robot eventually leads it "back" to an existing global node. To properly close the loop of the map shown, node $g5$ must be connected to node $g1$, and to this, the robot must in some way recognize that it has indeed returned to an existing node. In the system presented here this is handled by comparing the visual attributes of a new global node's surroundings with that of the surroundings of the existing nodes. The visual surroundings of existing nodes are mapped in the perceptual SOMs of the local maps anchored to existing global nodes.

To perform a comparing of visual surroundings the robot captures a four-directional image of the environment before creating each new global node. This sample is run over the perceptual SOM found in the first global node created and compared to the weight vector of the winning node, according to the algorithm:

1. Before global node creation, sample the perceptual space at the current position of the robot.

2. Retrieve the perceptual node n_w with the minimum Euclidian distance d_m from the sample by:

$$n_w = d_m = min(||\vec{\xi} - \vec{w_j}||), j = 1, 2, 3.., N \tag{3.33}$$

   where $\vec{\xi}$ is the input sample, w_j is the weight vector of node j and N is the total number of perceptual nodes in the first global node of the map.

3. If the minimum distance $d_m < T_d$, which is a distance threshold, the robot is assumed to be positioned at the transition position of the global node $g_w$to to which the winning perceptual node n_w is belonging. A loop is then found in the map. The loop is closed by connecting the previous node g_p with node g_w
   If $d_m < T_d$, the dissimilarity is deemed to large for a confident position estimate to be made, and a new global node is created.

3.5 Navigation in a Mapped Environment

Navigation over an existing map can be separated into two parts, global navigation and local navigation.

3.5.1 Global Navigation

Global navigation concerns navigation over the nodes in the global topological map. An existing global map consists of nodes representing locations in the environment. The global nodes are topologically connected through arcs, containing information on how to navigate between topologically adjacent nodes.

Because of the topological nature of the global map, path-finding is implemented as a best-first graph search over the global map, specifically in the form of a A* shortest path algorithm. The A* algorithm is fairly straight-forward, and is described in detail in appendix 3.

Assuming the robot initiates navigation from a known starting point that is positioned at the location of a global node; navigation is performed as a result of sequentially following the navigational data contained in the arcs connecting global nodes. Navigational data is in the form of a wall-following and distance pair.

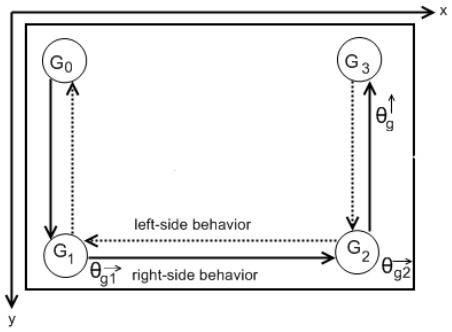


Figure 3.30: A global topological map

Global navigation algorithm To describe the global navigation process, we consider figure 3.30. The figure shows four global nodes G_1, G_2, G_3 and G_4 initially mapped by performing right-side wall-following behavior. The nodes are therefore connected by right-side wall-following behaviors in the original direction:

$$G_0 \rightarrow G_1 \rightarrow G_2 \rightarrow G_3$$

and by left-side wall-following behaviors in the opposite direction:

$$G_3 \leftarrow G_2 \leftarrow G_1 \leftarrow G_0$$

Navigation over the global map where the robot is initially positioned at global node G_i is defined by the following steps:

1. Generate the shortest path from initial node G_i to goal node G_n by:

$$P[N] = \text{findPath}(G_i, G_n)$$

where $P[N]$ is an ordered list of N nodes, constituting a shortest path from node G_1 to node G_i given by findPath(G_1, G_i) . findPath(G_1, G_i) is an implementation of the A* algorithm, described in appendix x.
Set step number $t = 0$.

2. Retrieve navigational information from the current node $P[t]$ to the next node of the path $P[t + 1]$ by:

$$\text{Arc<wall-following behavior, distance>} \; a_c = P[t] \rightarrow P[t + 1]$$
$$\text{WallFollowingBehavior} \; WF_n = a_c.[0]$$
$$\text{Distance} \; D_c = a_c.[1]$$

3. Initiate wall-following behavior WF_n for distance D_c.

4. Advance step count, $t + 1$. Repeat from step 2 until the goal node is reached.

Due to the self-aligning nature of the two types of wall-following behavior, the robot needs only to be at the location of the initial node, without considering the initial heading of the robot. This can be illustrated by a pair of cases:

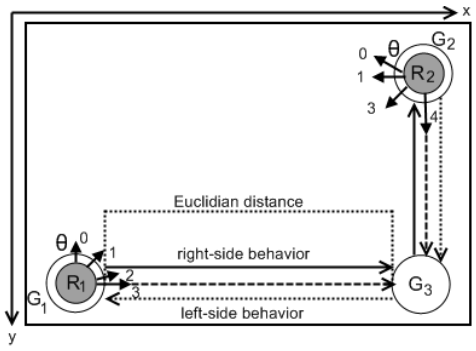


Figure 3.31: Navigation from directionally opposite initial nodes

Figure 3.31 shows the movement of two robots R_1 and R_2 from two different initial global nodes G_1 and G_2 to a third global node G_3. The nodes are connected with the wall-following behaviors:

$$G_1 \rightarrow G_3 \rightarrow G_2$$
$$G_1 \leftarrow G_3 \leftarrow G_2$$

where $\rightarrow$ denotes a left-side wall-following behavior and $\leftarrow$ a left-side wall-following behavior.

R_1 executes right-side wall-following behavior until it has moved the entire distance defined in the arc connecting node G_1 and G_3. The numbered arrows in figure 3.31

illustrates how the robot aligns its right-side to the wall, before moving in parallel to the wall for the specified distance. The same sequence of alignment is shown for robot R_2 that aligns its left side to the wall before moving parallel to the wall for the specified distance. The distance traveled is measured relative to the global coordinate system in which the global map is set, thus the exact shape of the wall-following behavior executed has no impact on the distance traveled.

3.5.2 Position Correction Over Landmarks

Global navigation is driven by wall-following navigation routines and does not require full path-integration for movement control. The only metric information used in navigation between global nodes is the Euclidian distance between the nodes, angular information is not used. Because of this accumulating position error related to path-integration is avoided. This is however no guarantee that the position of the robot will be devoid of error. During navigation over any significant distance the estimation of the robots position within the map may become erroneous. Reasons for erroneous position estimations may be i.e. wheel slippage when the robot transits between global nodes, resulting in erroneous distance estimations that would terminate the wall-following behavior at the wrong time.

Navigation over local maps is performed by path-integration. Local maps represent fractions of the environment, and are intentionally small of scale to limit the accumulated position error resulting from navigation within these maps. The accumulated error due to navigation over local maps may not be negligible, leading to incorrect navigation.

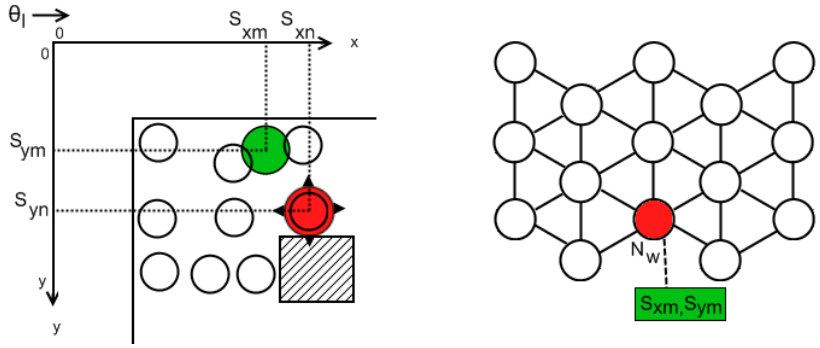
To negate robot position error a method of position correction over external cues is implemented. This position correction method allows the robot to estimate its position over the landmarks stored in the perceptual SOMs of local maps. The perceptual SOMs of local maps contain nodes that are trained to exhibit different excitation patterns when presented with visual stimuli in the form of four-directional images. Through the perceptual training algorithm these nodes were also associated with coordinates from the local space to which they are anchored. Coordinates of the robots current position within a local space may thus be retrieved from a perceptual SOM by sampling the perceptual space and present the sample to the local perceptual SOM. The node exhibiting the strongest excitation when presented with the node represents the position of the robot in perceptual space. If a coordinate is linked with the winning node, a position can be estimated based on purely external sensory stimuli.

A new position is retrieved from a perceptual SOM by:

$$p_i(t+1) = p_j \tag{3.34}$$

where p_j is a coordinate retrieved from a perceptual SOM.

Position correction algorithm To perform position correction over the landmark or perceptual SOM the robot must initially be positioned in a local map. If the robot is not positioned in a local map when a position correction is performed, the node responding the most to the supplied visual sample will still return the coordinate it potentially is



(a) A four-directional image sampled at position S_{xn}, S_{yn} colored in red, and a position S_{xm}, S_{ym} retrieved from the perceptual SOM

(b) The winning node N_w of a input pattern sampled at position S_{xn}, S_{yn}, and the coordinate it is linked with S_{xm}, S_{ym}

Figure 3.32: Coordinate retrieval from a perceptual SOM

associated with. A threshold value could potentially safeguard against a robot estimating its position to be within a local space when it is not, such a threshold value is not implemented in the current system.

1. Capture a visual input pattern $\vec{\xi}$ of the robots environment in the form of a four-directional image
2. Present the visual sample $\vec{\xi}$ to the perceptual SOM of the current local map.
3. Calculate the winning node of the perceptual SOM N_w by:

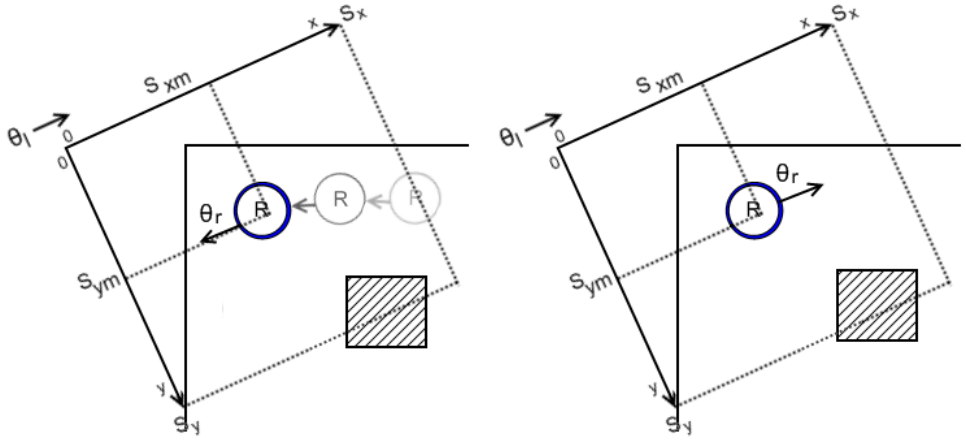
$$p = min(||\vec{\xi} - \vec{w_j}||), j = 1, 2, 3.., K \tag{3.35}$$

   where K is the total number of cells in the output layer.
4. Set the position of the robot to the coordinate retrieved from the winning node N_w
5. Navigate to the middle node of the map

3.5.3 Local Navigation

Local navigation concerns navigation over local metric maps anchored to the global nodes of the global topological map.

Transition pose When the robot approaches a global node, it must first correct its position according to the algorithm described in section 3.5.2. The heading of a robot as it reaches a global node is assumed to be approximately identical to the heading of the robot as the node were created, which defined the orientation of the local space anchored to that global node, given that it reaches the global node by engaging the wall-following behavior used in map creation. If the inverse wall-following behavior of that used in map



(a) A robot as it approaches a global node, and the local metric area anchored to that node

(b) A robot in a transition pose

Figure 3.33: A robot approaching a global node and entering the transition position to navigate the local map anchored to the global node.

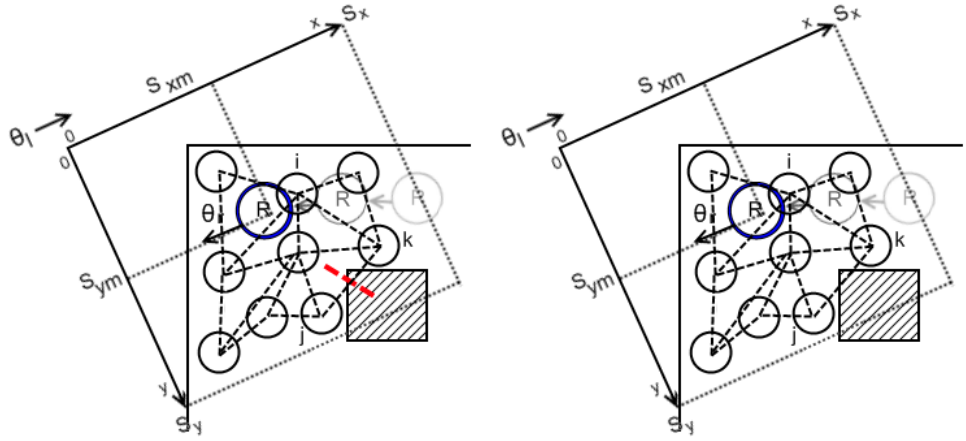
creation is engaged in navigation, the heading of the robot as it reaches a global node is assumed to be approximately inverse to the orientation of the local space anchored to that node. The robot must align its heading with the orientation of the local space in cases of inverse navigation. As the robot heading is assumed to be inverse of the local space orientation, this alignment is done by rotating the robot deg 180. In figure 3.33(b) the robot has aligned itself properly; it is now in a transition pose. The robot must be in a transition pose to properly navigate the local map by performing path-integration over the coordinate system related to individual local maps.

To navigate over the local map of a global node, the robot must place itself in a transition pose. A transition pose means that the robot is located at the center point of a local space and that the robot's heading equals the orientation of that local space. Figure 3.33 illustrates this operation. Figure 3.33(a) shows a robot as it approaches a global node, marked in blue. The figure also show the local space anchored to the global node. The local space has length S_x, height S_y and orientation θ_l.

After the robot has aligned itself to the local space of a global node, it is ready to navigate in local space by moving from node to node in the trained metric SOM of the local map. Initially the robot must move to the node of the metric SOM that is closest by means of Euclidian distance to the middle point of local space. Figure 3.34(a) illustrates this, where node i is the closest to the middle point.

When the robot finds itself at a node in the metric SOM, navigation is a straightforward process of navigating by path-integration between the nodes of the metric SOM. Paths between nodes in the SOM is generated by use of the A* algorithm, described in appendix 3

When the robot encounters an obstacle during navigation between two nodes it will mark the arc connecting the relevant nodes as blocked. Thus when navigation is performed over the same map at a later stage, the arcs will be excluded from the path-finding algorithm. Figure 3.34(b) illustrates how the robot will marked the arc connecting node j and k as blocked.



(a) The trained metric SOM of a local map

(b) A blocked path

Figure 3.34: The SOM of a local map

For the robot to return to the topological paradigm, it must return to the transition point, which equals the position of the actual global node.

Chapter 4

Experiments

The system presented in this thesis is tested on a simulated e-puck robot in the Webots simulator. The Webots simulator is a 3d robot simulator that supports simulation of Gaussian sensory noise. The simulation of sensory noise is vital to properly test the robustness of a robotic mapping system, as it is in practice unavoidable in the real world, where such systems are put to use.

The noise model in Webots was used to add cumulative uniform noise to the incremental wheel-encoders of the robot. At each simulation step, every 32 ms, an increase value is computed for each encoder. A random uniform noise is applied to the increase value before it is added to the encoder value. A noise component of $+/-10\%$ was used.

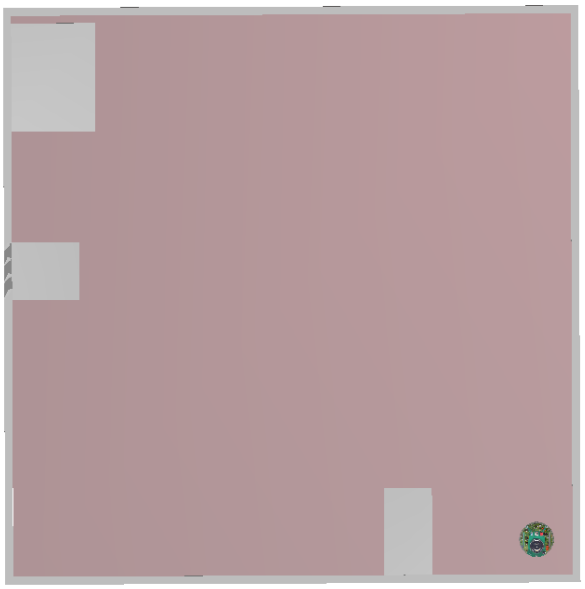
The experiments are separated into four main parts, where the separate modules of the system are empirically tested over varying scenarios. The results of the experiments are discussed at the end of each part:

1. Global and local mapping
   In the first part the global mapping algorithm is empirically tested both in isolation and in concert with the local mapping algorithm over environments of varying complexity.

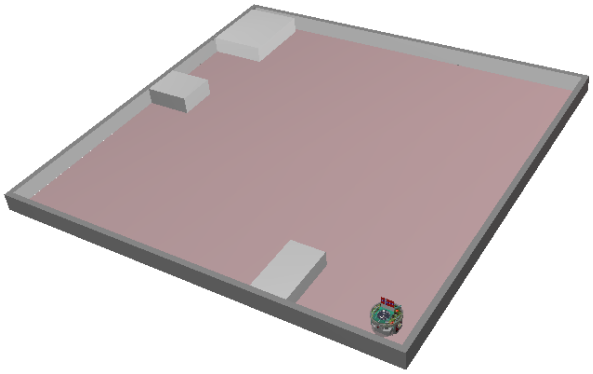
2. Global and local navigation
   In the second part the global and local navigation algorithms are empirically tested both in isolation and in concert over the maps created in part 1. This includes experiments on the localization methods over external cues.

3. Loop closing
   In the third part the loop closing capabilities of the system is empirically tested over a set of environments.

4.1 The Simulated Environment



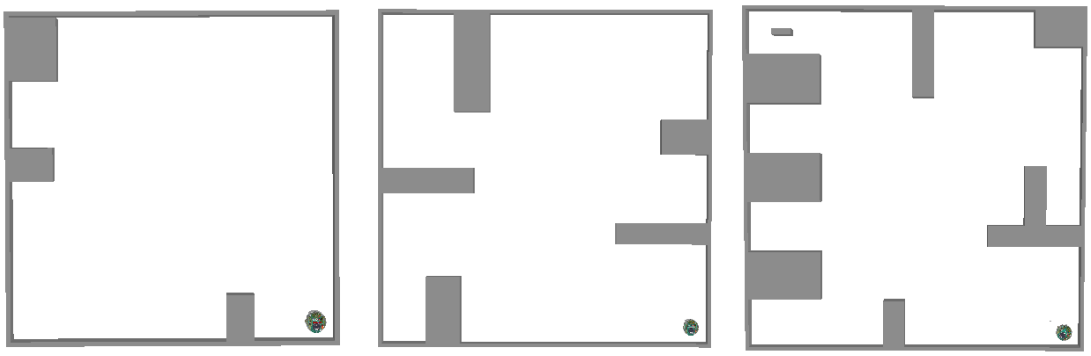
(a) An example environment in orthographic view



(b) An example environment in perspective

Figure 4.1: A simulated environment

The simulated environments used in the experiments described in this thesis all have elements in common, they are all fully enclosed, allowing for a cyclic environment, upon which the mapping system presented here depends. An example cyclic environment is illustrated in figure 4.1. The walls and obstacles of the environments are all colored white so that they will properly reflect infrared light. Additionally, the obstacles found in the environments are rectangular in shape.



(a) Environment 1, 1x1 meters (b) Environment 2, 1.5x1.5 meters (c) Environment 3, 1.5x1.5 meters

Figure 4.2: Three environments of varying size and complexity

4.2 Global and Local Mapping

4.2.1 Global Mapping

The mapping system presented in this thesis is tested in the three different environments shown in figure 4.2. The environments vary in size from 1 square meter to 1.5 square meters, besides varying in topographic complexity.

In a first set of experiments the global mapping algorithm was tested in isolation over the environments. Each environment was mapped ten times over, five using right-side behavior and five using left-side behavior. The initial position of the robot was identical in all runs. The parameters used in these experiments are listed in table 4.1.

Global map parameters	
Angular threshold	45°
Distance threshold	5cm

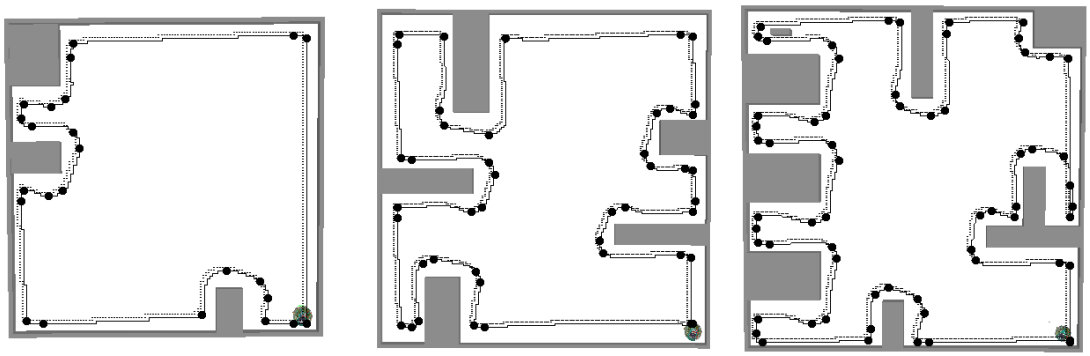
Wall-following parameters	
Minimum wall-distance	2.0cm
Robot speed	3cm/sec

Table 4.1: Mapping parameters

The global map threshold parameters define the change in heading and Euclidian distance from the previous global node required for a new global node to be created (Angular threshold and Distance threshold).

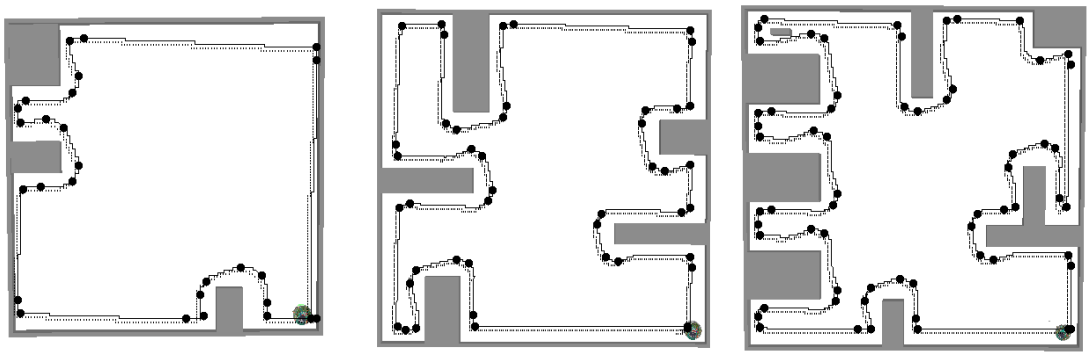
Figure 4.3 shows example paths of the robot during right-side exploration of the three environments, and in figure 4.4 paths resulting from exploration by left-side behavior. The nodes are also shown at the locations in which they were created.

The runs over each of the three maps were compared to find the degree of consistency exhibited by the wall-following exploration technique. A set of key comparisons of results from the global mapping experiments is presented in table 4.2. These results consider the distance covered by the maps in one cycle of the environment. Distance statistics from the runs are compared with the approximate actual distance covered by the robot in one such cycle. The actual distance presented in the table was approximated by mapping the three environments with no noise affecting sensors, including the wheel-encoders, essential in distance estimation. The average positional error at the end of the mapping



(a) (b) (c)

Figure 4.3: Approximate robot paths from right-side exploration



(a) (b) (c)

Figure 4.4: Approximate robot paths from left-side exploration

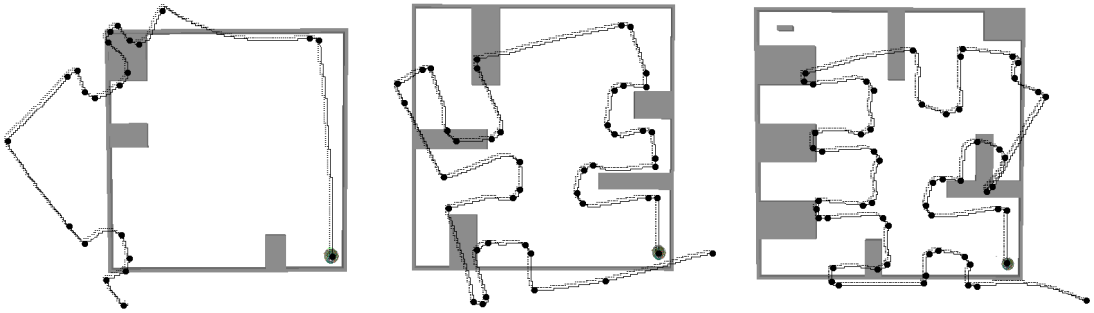
runs is also presented.

The distance value is important because it defines the relation between global nodes in addition to a certain type of wall-following behavior, and thus it defines the global map itself.

	Env. 1	Env 2	Env 3
Approximate actual distance	461 cm	909 cm	987 cm
Maximum distance traveled	472 cm	920 cm	1005 cm
Minimum distance traveled	448 cm	894 cm	979 cm
Average distance traveled	455 cm	915 cm	985 cm
Average distance variance	10 cm	15 cm	14 cm
Best-case positional error	5 cm	6 cm	6 cm
Worst-case positional error	12 cm	11 cm	18 cm
Average positional error	8 cm	9 cm	10 cm

Table 4.2:

Before discussing the implications of these results it is useful to review the accumulated position error resulting from using path-integration driven navigation over the same runs:



(a) (b) (c)

Figure 4.5: Accumulated positional error from path-integration

	Env. 1	Env 2	Env 3
Maximum positional error	95 cm	124 cm	122 cm
Minimum positional error	24 cm	25 cm	45 cm
Average positional error	61 cm	84 cm	91 cm

Table 4.3: Accumulated positional errors

Discussion, global map accuracy The resulting data from the experiments described above comes in the form of a global map, with global nodes connected through arcs containing distance and wall-following behavior information. The results will be discussed in relation to how accurate the resulting global maps represent the mapped environments.

The results presented in table 4.2 and 4.3 are interesting, and especially in comparison to each other. The one-cycle distance comparisons in table 4.2 show that the variance

in the distance estimates from the mapping runs are on average low over significant distances in a significant amount of runs, implying consistency in the two types of wall-following behaviors in that they consistently generate maps of approximately the same size. Additionally the average distance error is also averagely consistently low over the runs.

Figure 4.5 shows examples of node positions estimated by path-integration and table 4.3 lists the resulting positional errors from one cycle of mapping. These errors are shown to be very large, even in the smallest of the environments. This result was expected and can be explained by the way in which new positions are continuously estimated based on the previous position estimates in path-integration. Small errors in orientation, due to sensor noise, wheel-slippage etc., results in an accumulation of error, producing the un-operational maps exemplified in figure 4.5. In the global mapping algorithm presented here, global nodes are not defined as absolute coordinate positions as is the case in path-integration. A distance metric is used however, but the distance values defining the relations between nodes is not dependant on any previous position estimates, and error is thus not allowed to accumulate as in path-integration driven mapping.

The results presented above demonstrates the effect of using limited metric information in topological mapping in comparison to using path-integration driven navigation over significant distances. Even so, the global mapping algorithm is not noise-free. The reason for the noise can be explained by wheel-encoder error. Even though noise clearly does not accumulate in the same way shown in path-integration, it is still a factor. As the noise is not accumulative, it can be assumed to flatten out statistically over time. The results do support this, as the average error is shown to be very small. Of course, the worst-case maps cannot be discarded; they demonstrate that the system cannot generate perfectly accurate maps by discarding the heading value alone. As distance metrics are used in defining the relations between nodes in the topological map.

4.2.2 Local Mapping

Local mapping includes both a mapping of the metric space and the perceptual space of a defined fraction of an environment.

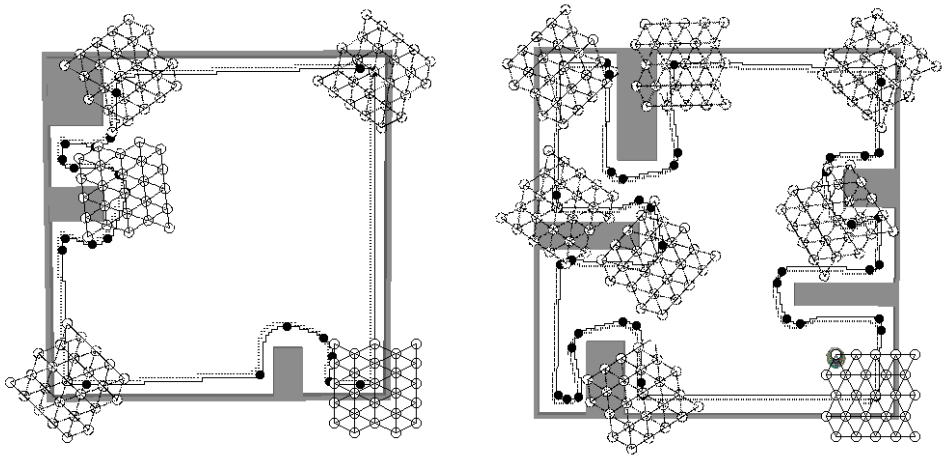
The local mapping algorithm was tested over the three environments shown in figure 4.2. Local metric and perceptual maps were created at the location of global nodes in a series of runs, where each environment was mapped six times over, three for each wall-following behavior. The parameters used in the global mapping process are identical to those found in table 4.1, the parameters specific to the sampling process of the local mapping algorithm are shown in table 4.4.

Rigid map setup	
Rows	5
Columns	5
Node distance	5 cm

Sampling parameters	
Metric sample threshold	$2cm$
Perceptual sample threshold	$4cm$
Perceptual sample resolution	$16 * 16px$

Table 4.4: Local map parameters

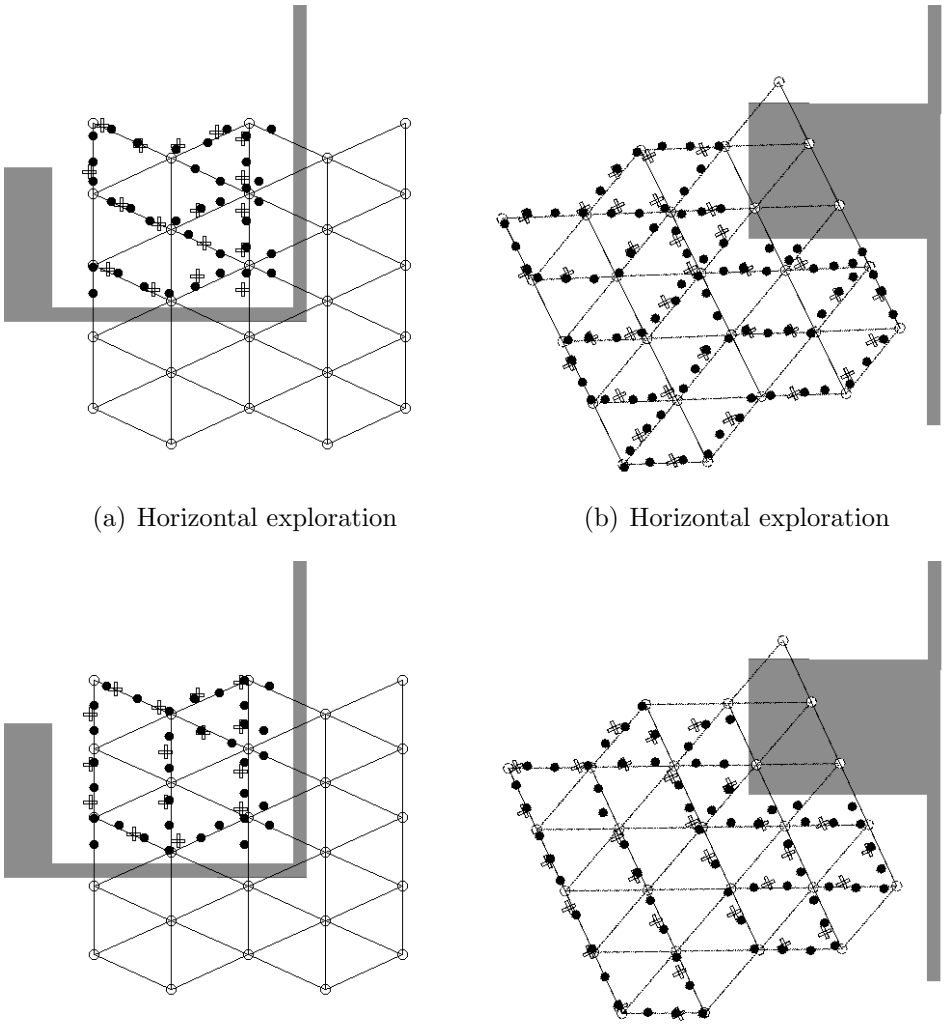
Exploration and sampling of local spaces First the exploration and sampling process is reviewed. Figure 4.6 shows the size of the local maps and their orientation relative to the robots heading at the time of local map generation.



(a) (b)

Figure 4.6: A selection of rigid maps anchored to nodes

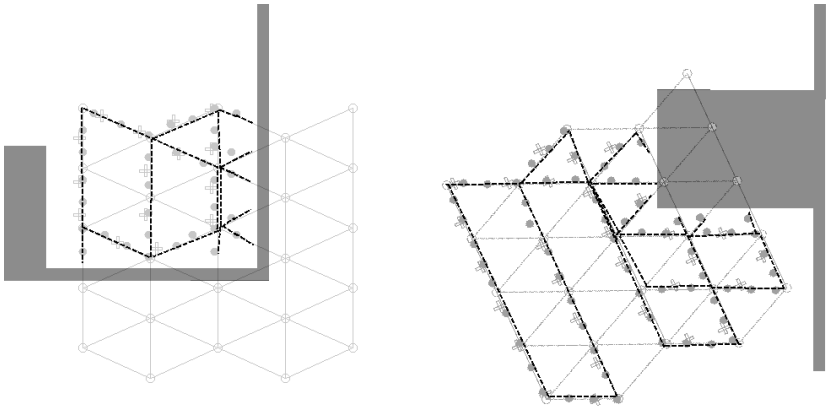
The vertical and the horizontal sampling strategies were used in alternation, producing sampling paths as exemplified in figure 4.7.



(a) Horizontal exploration (b) Horizontal exploration

(c) Vertical exploration (d) Vertical exploration

Figure 4.7: Sampling of local maps, circles are coordinate samples while crosses are four-directional image samples



(a) (b)

Figure 4.8: Example robot paths from vertical exploration

Figure 4.7 shows examples of recorded samples over two local spaces, explored by both vertical and horizontal exploration strategies. Figure 4.8 shows examples of the actual robot paths from the local spaces mapped in figure 4.7(c) and 4.7(c). Table x lists statistics over positional and angular error relative to the robots position before and after mapping of local spaces. The statistics covers all the mapping runs, 18 in total.

	Env. 1	Env 2	Env 3
Average number of local maps	10	23	27
Maximum positional error	2.1 cm	1.8 cm	2.7 cm
Minimum positional error	0.3 cm	0.2 cm	0.3 cm
Average positional error	0.6 cm	0.6 cm	1.0 cm
Maximum angular error	20°	17°	23°
Minimum angular error	3°	5°	2°
Average angular error	12°	9°	13°

Table 4.5: Positional and angular error over local maps

To describe how the inclusion of local maps and the positional error resulting from this affected the accuracy of the global map distance statistics for the global map is listed in table 4.6.

	Env. 1	Env 2	Env 3
Approximate actual distance	461 cm	909 cm	987 cm
Maximum distance traveled	465 cm	922 cm	1004 cm
Minimum distance traveled	450 cm	902 cm	977 cm
Average distance traveled	451 cm	910 cm	984 cm
Best-case positional error	4 cm	4 cm	7 cm
Worst-case positional error	11 cm	13 cm	17 cm
Average positional error	7 cm	8 cm	11 cm

Table 4.6:

Discussion, local maps and the impact on global map accuracy Before presenting the remaining results related to local mapping I discuss the impact on the accuracy of the global map in light of the inclusion of local maps, and the accuracy of sampling in local maps.

During exploration of local spaces surrounding global nodes the robots motion is driven by path-integration. Table 4.5 show statistics over the positional error acquired after local map exploration. After a local map is fully explored the robot returns to the transit position to continue global mapping. Location error resulting from the local space exploration process may result in the robot positioning itself in a position not equal to the transition position from which local mapping was initiated and this error would then be propagated into the global map.

Table 4.5 shows the worst case location error acquired from local space exploration to be 2.7 cm, which is not a large number considering that navigation is driven by path-integration. The reason why local map navigation seems to be so accurate could be due

to two factors. The first and obvious of which is the small size of local maps, limiting the distance over which path-integration is performed and thus limiting the accumulated positional error. However, the worst-case errors are still surprisingly low compared to the large errors resulting from path-integration driven global mapping exemplified in figure 4.3 and table 4.5, even though the distances traversed in those cases far exceeds that of the local maps. The explanation for the dampened accumulated error found in local maps could be explained by, together with smaller distances, the way in which path-integration is performed in local maps. The path-integration driven mapping exemplified in figure 4.5 was performed by a continuous estimation of position, allowing for calculating the coordinates of an arbitrary path. In local mapping however the robot navigates from node to node in the rigid map, rotating in place and moving in straight lines only. This form of path-integration require fewer position estimates than what is needed when estimating arbitrary paths and essentially it separates turning operations from moving operations. This method of path-integration does not require a continuous partition of robot motion into small discrete movements as is the case when estimating arbitrary paths. These attributes seems to aid local map accuracy considerably in concert with the limited distances of local map navigation.

Table 4.6 shows error statistics over the global maps when local maps are generated and anchored to global nodes. The results indicate that the inclusion of local maps and the path-integration performed within these does not have a large negative impact on the accuracy of the global map. In fact, the accuracy of the global map has in some cases improved fractionally. The inclusion of local maps was expected to have a negative influence on global map accuracy to some degree. The reason that this has not happened can be explained by the minute accumulated error resulting from path-integration over local maps. The local maps are separated and anchored to global nodes too support limited navigation by path-integration while avoiding accumulated error. The results indicate that the separation of metric maps accomplishes this and that the hypothesis that path-integration over small distances yield negligible accumulated error holds.

Metric and perceptual SOM training The metric and perceptual samples recorded in the local spaces of global nodes were presented as input to the metric SOM and the perceptual SOM constituting the local maps. The varying parameters tested for the SOMs are listed in table 4.7.

Metric SOM	
Horizontal nodes	3, 5, 10
Vertical nodes	3, 5, 10
Initial weights	uniform*

Perceptual SOM	
Horizontal nodes	5
Vertical nodes	5
Initial weights	$< -0.3, 0.3 >$ **

Table 4.7: SOM parameters. * Initial weights are distributed evenly over the size of local spaces. ** Initial weights are randomly chosen from the range.

Training of the metric and perceptual SOMs were performed over two configurations described in table 4.8.

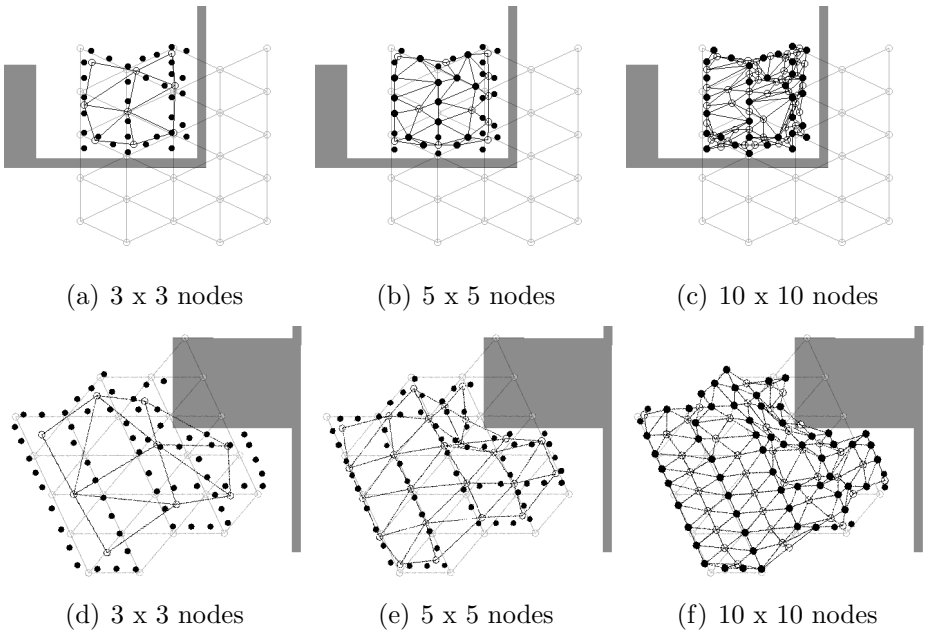
Parameters	Metric SOM		Perceptual SOM	
	Config. 1	Config 2	Config 1	Config 2
Ordering phase T	1000	1000	500	500
Initial learning-rate α_0	0.4	0.1	0.7	0.3
Initial neighborhood radius σ_0	-	3	-	3
Minimum learning-rate α_T	0.01	-	0.01	-
Adoption gain	linear	p.s. *	linear	p.s. *
Neighborhood function	bubble	Gaussian	bubble	Gaussian
Average delta value	10^{-8}	10^{-8}	10^{-6}	10^{-6}

Table 4.8: SOM configurations. * Power series

Learning-rate functions	
Linear	$\alpha_0 - (\alpha_0 - 0.1)t/T, t < T$ $0.01, t > T$
Power series	$\alpha_0 \left(\frac{\alpha_T}{\alpha_0}\right)^{t/1000}$ $0.01, t > 1000$

Neighborhood functions	
Linear, N_i	i + 6 neighbors of i, $t < 1000$ $i, t > 1000$
Gaussian	$\exp(-(\frac{\|\|r_c - r_p\|\|^2}{2\sigma^2(t)}))$

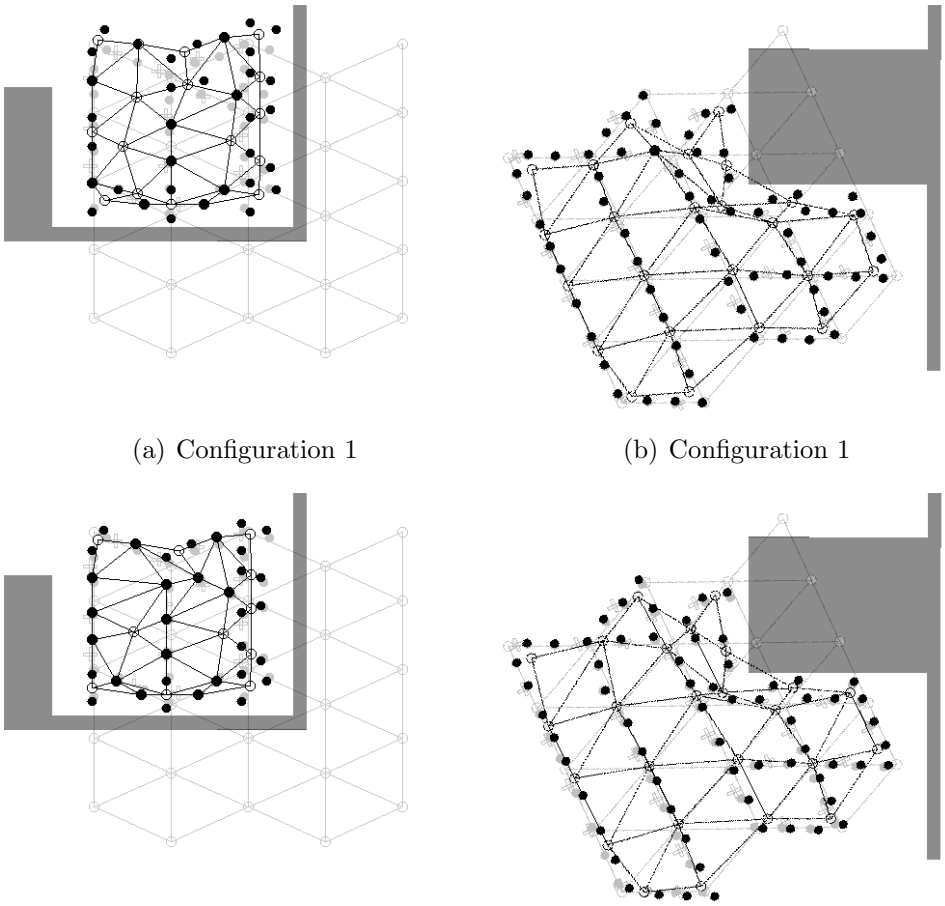
Table 4.9: Description of learning-rate functions and neighborhood functions



(a) 3 x 3 nodes (b) 5 x 5 nodes (c) 10 x 10 nodes

(d) 3 x 3 nodes (e) 5 x 5 nodes (f) 10 x 10 nodes

Figure 4.9: Local metric SOMs of varying node size after training



(a) Configuration 1 (b) Configuration 1

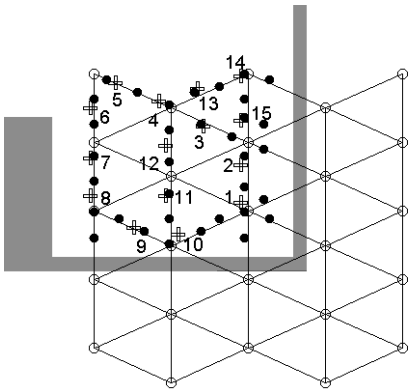
(c) Configuration 2 (d) Configuration 2

Figure 4.10: Example local metric SOMs after training

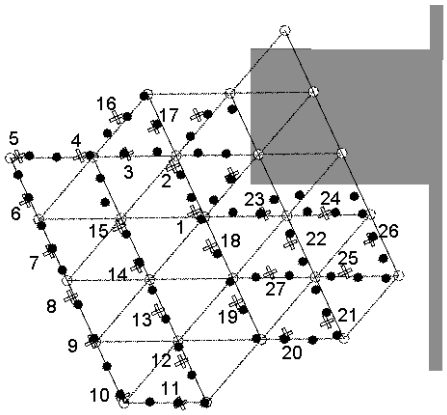
Metric SOM training iterations		
	Configuration 1	Configuration 2
Max iterations	2890	4345
Min iterations	1850	3640
Average iterations	2308	4460
Average time usage	2.0 sec	4.0 sec

Perceptual SOM training iterations		
	Configuration 1	Configuration 2
Max iterations	1241	1609
Min iterations	816	1235
Average iterations	954	1444
Average time usage	6.5 min	8.0 min

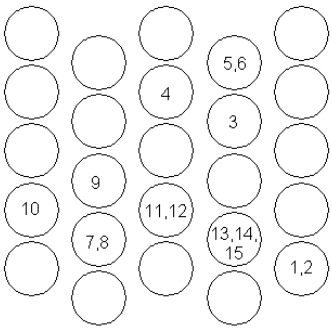
Table 4.10: Statistics over the number of iterations used in each configuration to reach the error threshold



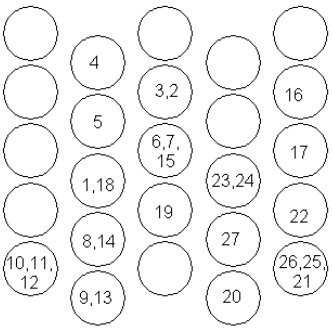
(a) Example 1



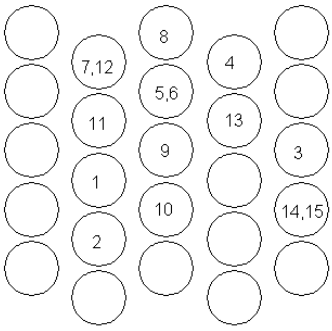
(b) Example 2



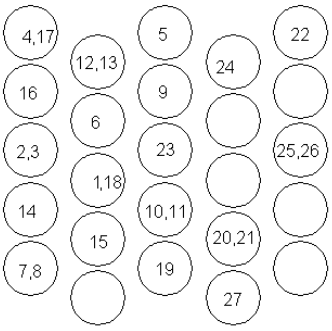
(c) Configuration 1, example 1



(d) Configuration 1, example 2



(e) Configuration 2, example 1



(f) Configuration 2, example 2

Figure 4.11: Example perceptual SOMs after training. Nodes are linked with the coordinates of the visual samples identified by numbers within the nodes.

Discussion, local metric and perceptual SOMs The metric SOMs and perceptual SOMs were tested over two parametric configurations each; this was done to compare the two methods of SOM training implemented in the system. Configuration 1 has a linearly decreasing learning rate and a bubble neighborhood function while configuration 3 has a power series learning rate and a Gaussian neighborhood function, both of which are described in table 4.9. Examples of metric SOMs in a trained state is shown in figure 4.10. These figures examplify the similar mapping resulting from both training configurations. Both configurations yielded topologically ordered nodes, mapping the recorded coordinate samples in a fairly precise manner.

Several metric SOM setups were tested, to find the best node resolution. Three different amount of nodes were tested, a 3x3 map, a 5x5 map and a 10x10 map. Examples of these variations are shown in figure 4.9. As can be seen from the figure the larger the map resolution the larger the accuracy of the mapping. Accuracy is certainly a favorable attribute of a local metric map, but other factors must also be taken into account. A 3x3 node resolution is clearly too small for accurate local space representation with a significant amount of free-space and topographic complexity. Figure 4.9(d) exemplifies this, the map resolution is simply too small to map the environment to any significant accuracy. Indeed, large portions of the sampled free-space are entirely left out of the trained metric SOM. The largest resolution tested, 10x10 nodes, consistently yielded very accurate maps. However, even in the local space with the largest amount of actual sampled coordinates and the most complex topography, a large percentage of the metric SOMs nodes were redundant, leading to unnecessarily large maps, relative to the number of nodes. An example of redundant nodes are shown in figure 4.9(f) and 4.9(c). A resolution of 5x5 nodes consistently yielded fairly accurate maps while at the same time avoiding redundant nodes in all but the local spaces with a small amount of free-space. An example of such a local space is shown in figure 4.9(b). Thus a resolution of 5x5 nodes in metric SOMs seems to be a good compromise between accuracy and map size.

The metric SOMs were additionally tested over two different parametric configurations, configuration 1 has a linearly decreasing learning rate and a bubble neighborhood function while configuration 3 has a power series learning rate and a Gaussian neighborhood function, both of which are described in table 4.9. Examples of metric SOMs in a trained state is shown in figure 4.10. These figures exemplify the similar mapping resulting from both training configurations. Both configurations yielded topologically ordered nodes, mapping the recorded coordinate samples in a nearly identical manner. Figure 4.9(a) shows statistics over the iterations used in the two configurations. Configuration 2 is shown to be slightly more time-consuming than configuration 1, although both configurations reach equilibrium within few seconds. This can be explained in part by the more gradual tuning of weights in configuration 2, resulting from the Gaussian neighborhood function used in this approach, even though all the parameters of the configurations impact the gradient of the weight tuning. A more gradual weight tuning should produce more accurate maps, however the results show no apparent increase or decrease in accuracy when configuration two is employed compared too when configuration 1 is employed.

Figure 4.11 shows two perceptual SOMs after training over perceptual samples recorded from two different local maps. Figure 4.11(c) and 4.11(d) show example SOMs trained over configuration 1 while figure 4.11(e) and 4.11(f) show example SOMs trained over

configuration 2. The SOMs produced using both configurations were shown to be consistently accurate, in that visual samples captured at similar locations in the environment maps to similar nodes, and in some cases the same node, in the perceptual SOMs. Differences in the degree of accuracy were notable between the two configurations used in training. Configuration 2 consistently produced more accurate SOMs, in that that the visual samples were associated with a larger number of nodes after completion of the training process. This is clearly favorable, as it allows for a more accurate localization over visual samples when the perceptual SOM is used in position correction. The reason for the better accuracy of configuration two could, as in the case of the metric SOMs, be explained in part by the more gradual tuning of weights resulting from the Gaussian neighborhood function used, compared to the bubble function. The use of a Gaussian neighborhood function seems to have a significant impact on the resulting SOMs. Table 4.9(b) shows that configuration 2 is slightly more time consuming than configuration 1, though the training process of perceptual SOMs are extremely time consuming in either configuration. The reason for this is explained by the number of weights related to each node of the perceptual SOMs, one for each pixel in the visual samples. The visual samples used in the experiments described in this section consisted of four camera images, each having a resolution of only 16x16 pixels, merged into one image sample of 256x256 pixels resolution. Thus each node of the perceptual SOMs has 1024 weights each. As a result perceptual SOM training is a time-consuming process. Early experiments included smaller image samples, and although this improved time-consumption considerably it lowered the accuracy of the SOMs to un-operational values.

4.3 Global and Local Navigation

4.3.1 Global Navigation

The global navigation algorithm was tested in isolation. The robot navigated over three of the maps created in the experiments described in section 4.2.2. The maps used where those with a an error closest to the average positional error among the maps created, the actual errors are listed in table 4.11. The robot was programmed to navigate each map, by sequentially moving from the first to the last node of the map and back again five times over in each run. Maps for each environment were navigated in this fashion five times over.

	Env. 1	Env 2	Env 3
Positional error in the maps	7 cm	8 cm	11 cm

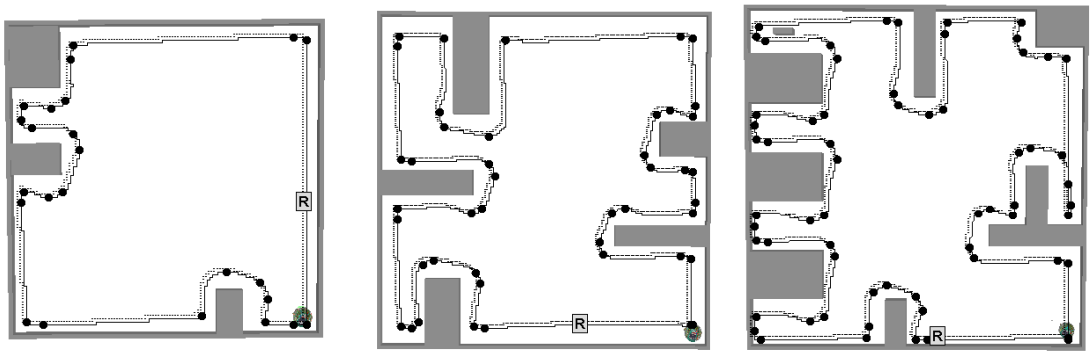
Table 4.11:

The parameters used in the wall-following behavior was identical to the ones used in map creation, defined in table 4.1.

Discussion, positional error Figure 4.12 show the worst case positional error after map navigation for each of the environments and table 4.12 show positional error statistics over all the runs. An average error of 49 cm after traversing the largest map five times,

	Env. 1	Env 2	Env 3
Approximate actual distance	23.05 m	45.0 m	49.35 m
Best-case distance error	17 cm	25 cm	26 cm
Worst-case distance error	41 cm	65 cm	71 cm
Average distance error	32 cm	46 cm	49 cm

Table 4.12:



(a) (b) (c)

Figure 4.12: Worst case positional error

constituting a total traveling length of almost 50 meters, seems to be fairly good in the absence of periodic position correction over landmarks. The reason for this limited positional error even after traversal of such long distances could be explained by the way in which global navigation is performed. As in the case of environment exploration, the robot moves by engaging a wall-following behavior for certain distances to navigate from node to node. Because no heading information is needed, the error from wheel-slippage and sensor noise is not allowed to accumulate unboundedly as is the case in path-integration driven navigation. Even though limited, the average errors listed in table 4.12 demonstrates the need for additional techniques for localization. Errors of this magnitude may be tolerable in a system purely meant for global navigation, but it is far to large for accurate navigation in local maps anchored to global nodes. The reasons for this are discussed in the next section.

4.3.2 Local Navigation

Local navigation was tested by navigation over a set of three maps created in the experiments described in section 4.2.2. Two best-case maps were used for environment 1 and 2 while an average map was used to navigate environment 3. The errors related to each map are listed in table 4.13.

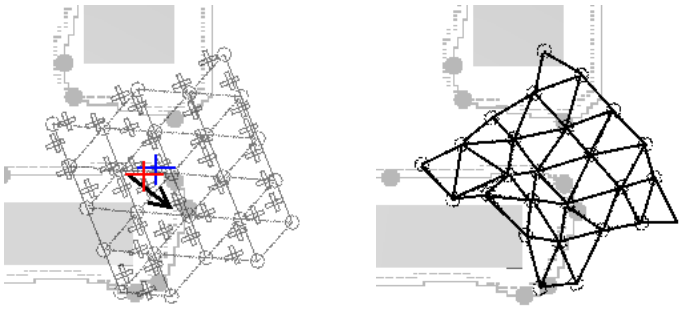
The global nodes of the maps were traversed five times per run for five runs, and every other local map was navigated in their entirety at the two first and the two last traversals of each map. The reason for this was to test navigation of a local space when that space was approached from both types of wall-following behaviors. Additionally testing local navigation at both the first two and the last two traversals was done to test the algorithms in the presence of varying positional error, as this error was assumed to be

smaller at the beginning compared to the error resulting from traveling the full distance of the runs.

	Env. 1	Env 2	Env 3
Positional error in the maps	4 cm	4 cm	11 cm

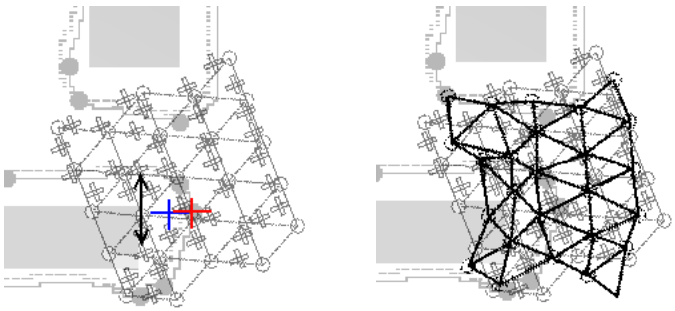
Table 4.13:

The position of the robot was corrected over landmarks at every global node visit, regardless of whether a local map was explicitly navigated to test local map navigation or not.



(a) Position estimate, env. 2 (b) Map orientation and position

Figure 4.13: Localization, and local map error, map traversal 1

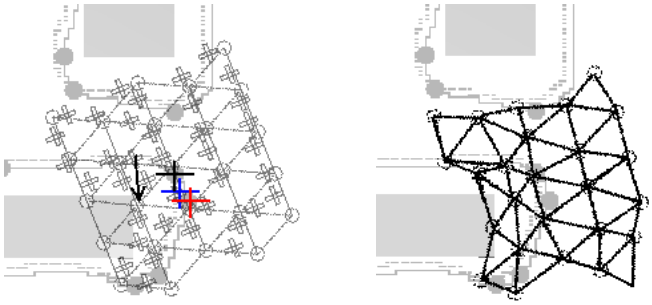


(a) Position estimate, env. 2 (b) Map orientation and position

Figure 4.14: Localization and local map error, map traversal 2

Figure 4.13,4.14 and 4.15 show position updates in a local map at different traversals of a map over environment 2 in this example.

The figures contain a set of crosses of different colors. The blue colored crosses show the actual position of a robot as it assumes the transisiton position of the local map. Ideally, this position should of course be equal to the middle point of the local map, constituting the coordinate of the transit pose. The red colored crosses show the coordinates retrieved from the perceptual SOMs. These coordinates constitute the updated position estimate of the robot.



(a) Position estimate, env. 2 (b) Map orientation and position

Figure 4.15: Localization and local map error, map traversal 4

The arrows in the figures show the heading of the robot as it assumes the transition pose. In the figures where bi-directional arrows are pictured, the robot approaches the local map from the opposite direction of that used in map creation, and must thus turn 180° to assume the estimated transition pose.

Additionally the local metric maps are shown in the positions and orientations estimated after robot localization is performed.

Successfull runs		
Env. 1	Env 2	Env 3
4/5	3/5	0/5

Table 4.14:

Global accuracy			
	Env. 1	Env 2	Env 3
Approximate actual distance	23.05 m	45.0 m	49.35 m
Best-case distance error	6 cm	4 cm	-
Worst-case distance error	11 cm	8 cm	-
Average distance error	7 cm	6 cm	-

Table 4.15:

Discussion, global and local accuracy The experiments described above provided ambivalent results. As table 4.14 implies, not all rounds were successfully completed, in fact none of the maps were successfully completed in all five runs, and none of the runs over environment 3 were successful. By unsuccessful it is meant that the robot was completely lost during the run and was unable to complete the run. Table 4.15 shows statistics over the accuracy of the successful runs, in great contrast these statistics show that the navigation was very accurate during the successful runs, with an average positional error of just 7 cm and 6 cm. The reason for these apparently mysterious results can be found by analyzing the position correction function.

Local accuracy			
	Env. 1	Env 2	Env 3
Average pre-correction error	2 cm	2.1 cm	-
Average post-correction error	1.0 cm	1.1 cm	-
Average angular error	18°	21°	-

Table 4.16:

The position correction function is described in section 3.5.2. When the robot during global navigation reaches a global node it terminates the wall-following behavior. To correct its position against the perceptual SOM of the current global node it samples a four-directional image. This image sample is run over the perceptual SOM, and coordinate associated with the node having the strongest reaction pattern to the sample is returned to the robot. The robot now believes it is positioned at the returned coordinate in the local space, and corrects its position by navigating to the center node.

The position correction function takes two critical assumptions; it is assumed that the robot actually finds itself within the mapped local space it believes it has reached, and that it is rotated correctly according to the rotation of the local space. The experiments above, shows that these assumptions cannot be made in all circumstances. Figure 4.16 illustrates a situation in which these assumptions lead a robot astray.

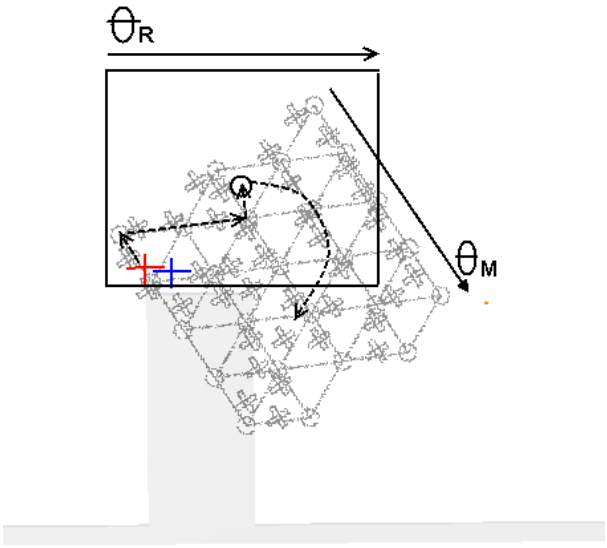


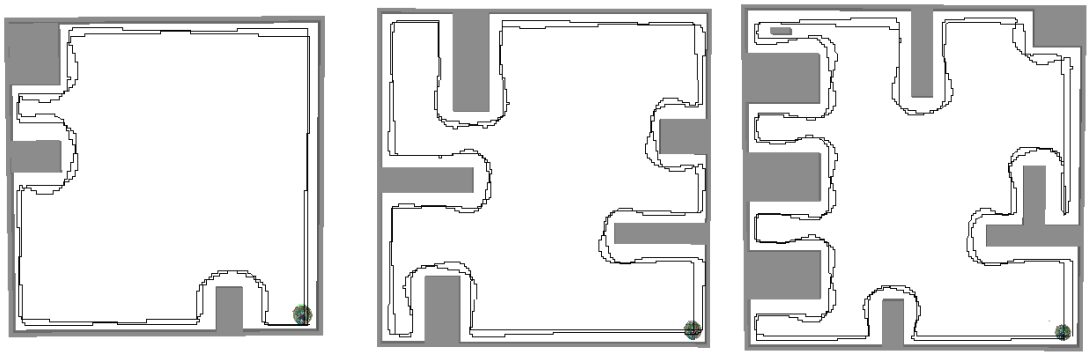
Figure 4.16: Corrupting position estimate

The blue cross in the figure shows the position of the robot as it believes it has reached a global node. The red cross is the coordinate returned by the position correction function, it is in this example nearly perfectly accurate. The heading of the robot Θ_R however is very inaccurate compared to the orientation of the local space connected to the global node the robot is visiting. Because of the faulty heading the robot navigates to a faulty location instead of to the global node before it rotates itself into its initial heading before continuing navigation. As the robot is positioned with an error of approximately 7 cm, in this example, the robot will if behaving ideally, terminate wall-following behavior 7

cm before reaching the actual middle position of the next global node. If its heading is equally faulty at the next node, the new position estimate will most likely be even worse than the one before.

It is easy to see that a faulty position estimate such as that pictured in figure 4.16 can throw the robot off course, as the error is propagated to the next position estimation. To understand why 3 of 5 runs were very accurately completed over environment 2, even with the existence of a fragile position estimation function, the figures 4.13,4.14 and 4.15 must be investigated. The figures show that the position estimates are consistently accurate through the traversals of the map. The heading of the robot is seen to vary slightly from that of the orientation of the SOMs, but as it is so close to the middle node, the error in heading seems to close enough. This reflects the accurate positions of the robot at the end of the navigation process described in table 4.15. The results can be explained by the way in which position estimates are periodically corrected over landmarks. Positional error between pairs of global nodes are shown to be small, due to the way in which global navigation is performed by avoiding the use of a heading value, and as the robots position is updated at short distance intervals, any positional error due to i.e. wheel slippage or sensory noise is corrected.

The noise related to the maps must be considered to properly explain the ambivalent results. No runs was successfully completed over environment 3, the map used in navigation over this environment contained an error of 11 cm, doubly that of the other maps. Even so, this error will be spread approximately evenly over the map, and as the accurate results in table 4.15 shows, the position correction function clearly functions well when initial pose of the robot is fairly close to the middle node or within the map and in an accurate heading. It can be argued that inconsistent wall-following behavior also can corrupt the initial pose of a robot The consistency of the two types of wall-following behaviors can be measured in the way they align. A robot engaging absolutely inverse wall-following behaviors, left-side and right-side, should have exactly the same position and exactly the inverse heading at the same point in the environment, depending on the behavior type used.

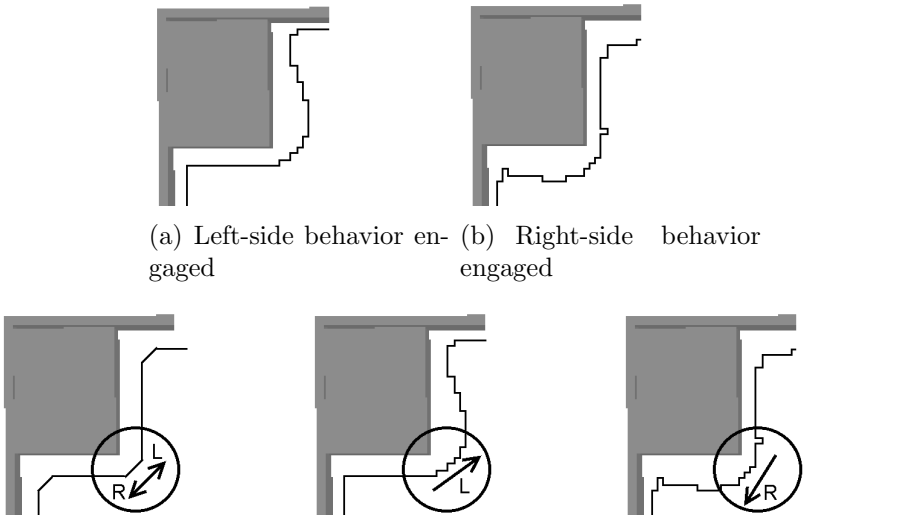


(a) (b) (c)

Figure 4.17: Both left-side and right-side exploration paths

Figure 4.17 shows two example mapping runs for each of the three environments, one explored by engaging left-side behavior and one explored by engaging right side behavior overlaid each other. The figure clearly illustrates that there is some divergence between

the two types of behavior. Especially true is this in "open corner" situations. This is situations in which the robot detects no walls in its vicinity. In such situations the robot will turn gradually to its left or right, depending on the wall-following behavior used.



(a) Left-side behavior engaged (b) Right-side behavior engaged

(c) Ideal corner paths (d) Left-side behavior (e) Right-side behavior

Figure 4.18: "Open corner" situations

Figure 4.18 shows how the robot path varies slightly when navigating an "open corner" using the two wall-following behaviors. The lack of consistency in navigation of "open corners" could to some degree corrupt the initial heading and consequently the navigation of local maps. This is because the entry heading of a local map anchored to a global node is defined by the entry heading of a robot as it created the global node. When navigating the map engaging another wall-following behavior than that used in map creation, the heading of the robot as it enters a global node is assumed to be the exact inverse of the orientation of the local map. Figure 4.18(c) illustrates how an ideal robot path would look, exhibiting the exact inverse heading during navigation.

Navigation of local maps suffers to a certain degree from the lack of a method of robot heading correction. With this taken into account, the average initial angular errors shown in table 4.16 was still small enough for a partially successful navigation of the local maps. Partially successful because the majority of local space was still covered by the metric maps, despite a degree of error in map orientation.

The reason no runs were completed successfully over environment 3 is likely to be a combination of the noise present in the map, inconsistent wall-following behavior and the length of the map, increasing the chance for a faulty initial pose to be assumed.

4.4 Loop Closing

The loop closing algorithm was tested in the mapping experiments of the three environments described in section 4.2.2, and produced an additional set of maps. The parameters for mapping in the loop-closing experiments was thus equal to the parameters used in standard environment mapping, listed for global mapping in table 4.1 and for local mapping in table 4.4.

A set of three distance thresholds used in defining the similarity requirement for a loop to be closed was tested. These thresholds were related to the accuracy of the perceptual SOMs contained in the initial global node of the map over which testing was done. The three thresholds were estimated by first deciding the average distance D_a between the visual samples used to train the perceptual SOM and the nodes most responsive to each visual sample in:

$$D_m = \sum_{i=1}^{k} \min(||\vec{\xi_i} - \vec{w_j}||), j = 1, 2, 3.., N$$
$$D_a = \frac{D_m}{k} \tag{4.1}$$

where $\vec{\xi_i}$ is the i'th input sample, K is the number of input samples, w_j is the weight vector of node j and N is the total number of perceptual nodes in the first global node of the map.

Threshold 1 T_1	$\frac{D_a}{2}$
Threshold 2 T_2	D_a
Threshold 3 T_3	$D_a * 2$

Table 4.17: Distance thresholds used in the loop closing experiments

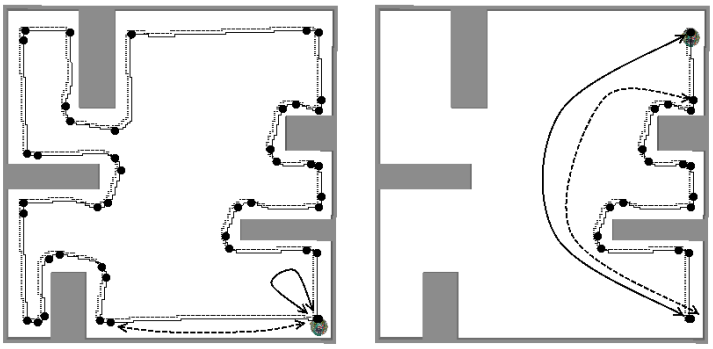
Figure **??** shows an example of a loop being correctly closed in environment 2. The loop closing algorithm returns an Euclidian distance below the current threshold, threshold 2 in this example, between the camera image, or input vector, sampled at the location of the robot and the weight vector of the winning node of the perceptual SOM in the initial node. Thus, instead of creating a new global node at this location, the previously create node, marked by a dotted arrow in the figure, is connected to the initial node and the loop is closed and mapping is terminated.

In figure **??** the robot mistakenly believes it has returned to the initial node as the Euclidian distance between the visual sample and weight vector of the winning node of the perceptual SOM in the initial node is below the threshold, threshold 3 in this example. Consequently, a new global node is not created, but the previously created global node is wrongly connected to the initial node and mapping is terminated prematurely.

Table 4.18 lists the success rate of the loop-closing algorithm over the three threshold values.

Discussion, loop-closing The results show that only one of the tested threshold values, threshold 1, consistently avoids erroneous loop-closing. The loop-closing algorithm using threshold 2 managed to correctly close the largest number of loops in all environments. However, it was also shown to consistently perform incorrect loop-closing as seen in table 4.17(b), while in the case of threshold 3, a large amount of incorrect loop-closing was observed.

Because of the way in which threshold 1 was estimated to be only half of the average error of the visual samples that was used to train the perceptual SOM, this threshold was expected to be to small for operational position estimation. When using this threshold



(a) Correct loop-closing (b) Incorrect loop-closing

Figure 4.19: Example correct and incorrect closing of a loop

Threshold 1			
	Env. 1	Env.t 2	Env. 3
Incorrect loop-closing	0	0	0
Correct loop-closing	1	0	0
No loop-closing	5	6	6

Threshold 2			
	Env. 1	Env.t 2	Env. 3
Incorrect loop-closing	1	1	2
Correct loop-closing	4	3	2
No loop-closing	1	2	2

Threshold 3			
	Env. 1	Env.t 2	Env. 3
Incorrect loop-closing	4	5	6
Correct loop-closing	2	1	0
No loop-closing	0	0	0

Table 4.18: Loop closing success rates

the visual samples captured at the global nodes must match the weight vectors of the perceptual nodes of the initial node doubly as precise as the average of the visual samples over which the nodes were actually tuned, clearly a restrictive limitation assuming accurately trained perceptual SOMs with weight vectors closely resembling the training patterns. The results clearly reflect that this is the case in a large majority of the runs. Interestingly, the results show that one loop was correctly closed using threshold 1. This can be explained by inaccuracies in the perceptual SOM of the initial node, in turn increasing all threshold values.

Although further experimentation with other threshold values could produce better results than those presented here, the results indicates that the method used is to fragile for operational use. Significantly small threshold values seems to ensure that no loops are mistakenly closed but it also seems to severely limit the performance of the algorithm as it requires an overly large similarity between a visual sample and the weight vectors of the initial SOM. Larger thresholds lead to an increase in both incorrect and correct loop-closing occurrences. The problem is then that of perceptual aliasing, where physically distinct locations have similar perceptual signatures. One possible method of improving the robustness of the loop-closing method, without algorithmic change would be to increase the resolution of the image samples used in perceptual mapping. An increased input pattern resolution would further distinguish perceptually similar locations in the environment. This would in turn allow for increased threshold values without an increased risk of incorrect position estimates. An increase in the resolution of visual samples would however affect the already large time consumption of perceptual SOM training. Another method could be to diversify the perceptual sampling of local spaces, by mapping i.e. distance values from distance sensors in addition to visual samples.

A third method could be to extend the existing method by sampling the entire local space of a potentially existing global node in loop-closing situations. A sampling of the perceptual space in the entire local space of a potential global node could be compared to the mapping already existing in local map of the initial global node. Such a comparison would be more informed than the single sample approach currently used.

Chapter 5

Conclusion

I conclude the thesis by providing a summary of the motivations and the central issues underlying the research done, in section 1. Section 2 reviews the results and summarizes the key points from the discussion. Finally, in section 2 I review the approach presented and reiterate how the central elements of the approach answers to the research issues and hypotheses formulated in the introduction chapter.

5.1 Motivations

This thesis introduced a hybrid robotic mapping system, merging the topological and the metric paradigm in a single unified representation. Topological and metric approaches both exhibit favorable attributes for use in autonomous robotic mapping, but few of these are shared between the two paradigms. The central hypothesis motivating the work in this thesis was that by combining elements from a metric and a topological representation in a hybrid representation, the positive attributes of purely metric and purely topological approaches could be obtained in one system, while the negative attributes traditionally associated with the two paradigms in isolation could be omitted.

Purely metric approaches represent an environment as a grid of evenly spaced cells. The cells of the grid represent fractions of coordinate space and are set in an absolute coordinate system. Purely topological approaches represent an environment as a set of nodes in a topological graph, and each of these nodes represent a location in the environment. The nodes of topological maps contain no positional information; instead the map is defined by how the nodes are connected.

These types of environment representation dictates the strengths and weaknesses traditionally associated with the two paradigms. Metric approaches provide geometrically accurate representations of environments. Accuracy is naturally a positive attribute by itself, but a large accuracy also leads to a large increase in memory usage and computational efficiency. Conversely, topological approaches do not need to map an environment in the complete way found in metric approaches, but generally seek to map the overall topography of an environment. Topological approaches therefore tend to keep compact representations of the environments. As a result topological approaches demand less memory than metric approaches as well as being more computationally efficient. These positive attributes are reflected by the obvious disadvantages associated with a less precise representation.

Another significant factor separating the two paradigms is the way in which navigation is performed. In metric approaches the prime method of navigation is to retain odometry data acquired from wheel-encoders and use this information to estimate the pose of the robot within the map. Because wheel-encoders are subject to noise and imperfect environments, the use of odometry data, referred to as dead reckoning, for position estimation alone will lead to incorrect estimations over time. This is because positions estimated by virtue of dead-reckoning depend on the string of previous position estimates, and position error will as a result accumulate unboundedly. Because sensory noise and factors like wheel-slippage cannot be avoided in practice, odometry based position estimation is characterized by large errors. Topological map are defined by the nodes and the way in which they are connected alone. The nodes of a topological map are not bounded in a coordinate system and do not depend on global consistency as does metric approaches. Topological systems are characterized by a reluctance to involve metric data in defining the relation of nodes. Topological maps that avoid the inclusion of a heading value in navigation do not suffer from accumulated position error.

A hybrid approach using a topological map to represent the overall topography of the environment and individual local metric maps to represent specific fractions of the environment was hypothesized to constitute a more robust representation than purely metric or purely topological approaches in isolation. By combining the paradigms, the robust and compact natures of topological maps are proposed to provide a robust and computationally efficient global representation, while local metric maps of limited size provide metric accuracy without introducing significant accumulated positional error.

5.2 Experimental Results

The approach was tested thourhougly in a range of simulations. The different elements constituting the complete system were tested in isolation and in concert over a range of experiments. The approach was tested on a simulated e-puck in the Webots simulator, a 3d simulator supporting the adding of Gaussian sensory noise to provide realistic simulation.

In a first range of experiments the global mapping part of the system was tested in isolation. The algorithm was tested in a series of runs over the three environments. The robot also estimated its position by path-integration in all the runs; this was performed so that the position error resulting from path-integration could be compared to the error estimates of the global mapping algorithm.

The results from the first experiments showed that the global mapping algorithm was significantly more accurate than the positions estimated by path-integration, though some error was observed. This error was caused by noise from the wheel-encoders used to estimate the distance between global nodes.

In the following experiments the global mapping algorithm was tested again, but this time local maps were created as well. The approximate angular and positional error from the sampling of local maps was observed to be negligible. The error after sampling of local maps was expected to be limited due to the small size of local maps, but the results exceeded expectations. The reason for the small error is likely a combination of the local spaces' limited size, but also due to the way in which path-integration is performed in

local spaces by moving from node to node in straight lines, and only rotating in place. This method of path-integration is more resistant than when an arbitrary path is followed, and this seems to have had a positive effect on the navigation in local maps.

The impact of local mapping on the global map accuracy was shown to be negligible, as the errors were very similar, and even better in some cases, than that estimated in the experiments were the global mapping algorithm was tested in isolation. The reason for this small impact can be explained by the accurate performance of local map navigation.

The perceptual and metric SOMs of the local maps were trained in varying configurations, both of which produced fairly accurately tuned SOMs. The nodes of the perceptual SOMs was consistently associated with correct coordinates, as visual samples captures at physically close locations in the local maps stimulated the same or adjacent nodes in the perceptual SOMs. The use of SOMs to both map the coordinate space and the perceptual space was thus showed to work well in that regard. The metric SOMs were instantly trained, but the perceptual SOMs spent almost 10 minutes adjusting in each local space. The reason for this is explained by the large number of weights in each node of the perceptual SOM, increasing the time used in the training of perceptual SOMs significantly.

A third series of experiments tested robot navigation over maps created in the experiments described above. To test the navigation algorithms properly maps for two of the environments was chosen that only were shown to inhabit small errors from the mapping process itself, while a map having an averagely large error was chosen for the last and biggest environment. This was done to test the navigation algorithm over maps with a small amount of noise but also over a map with a considerable amount of noise.

The experiments were first run over the global navigational algorithm in isolation. The positional error at the end of the runs was shown to be significant, but surprisingly small in light of the distances traveled with an average positional error of about 40 cm. The same experiments were run again, but this time the position of the robot was corrected against the perceptual SOMs of the global nodes when these were encountered.

The results from these runs were ambivalent. For the two first maps with the smallest inherent errors a majority of the runs was performed with very good accuracy, while a few runs did not complete successfully. This was also the case for the largest map with the largest inherent error. It was observed that position correction over perceptual SOMs alone was not enough to accurately estimate the position of a robot in all scenarios, as erroneous initial headings of the robot combined with significantly large errors in the initial positions of the robot could lead to a propagation of error, quickly leading to the robot getting lost.

In a final range of experiments the loop-closing abilities of the robot was tested, but the results showed inconsistent loop-closing performance, implying that the loop-closing method is too simple to operate functionally.

5.3 The Hybrid Approach

The approach presented in this thesis use a topological map to represent the overall topography of an environment while metric maps are anchored to locations in the topological map. A topological map used as the global map is favorable for several factors.

A global topological map allows for a compact representation of the overall environment, compared to a metric representation. Additionally, topological maps allows for the inclusion of non-metric data in navigation. By including non-metric data, navigation need not depend on dead-reckoning, and accumulated positional error is avoided. This is ensured by relating the nodes of the topological map with wall-following behaviors and metric distance values. As no angular heading value is used in global navigation, the global topological map can be expected to provide robust navigation.

The topological map was contained in a dynamical graph. An exploration strategy employing a reactive wall-following behavior was implemented. This wall-following behavior was also a vital part of the global map itself, as wall-following behavior was used in concert with a distance metric to define links between the topological nodes. A reactive wall-following behavior was considered to be important for robust navigation of the global map. This was justified by the consistency expected from a simple reactive behavior.

Local maps were implemented, consisting of a pair of self-organizing maps (SOMs). A metric SOM was implemented to map the unoccupied space surrounding a global node. The metric SOM was used to approximate the metric local space. A SOM was used because of the unsupervised learning process employed and the noise tolerance expected. The unsupervised training process allows SOMs to produce maps that retain the topography of the supplied training patterns. The unsupervised training allows an accurate mapping of the coordinate space surrounding global nodes without the need for any intervention by a human operator, ensuring robot autonomy and robustness. Additionally SOMs trained over coordinate samples could be used directly in later navigation of the local space and path finding was easily done because of the topological properties of the SOM.

Perceptual SOMs were used to map the perceptual space of local maps in the form of camera images. As with the metric SOMs, this allowed an unsupervised topography preserving mapping of perceptual space. After training these maps exhibit reaction patterns when supplied with input samples. The image samples were presented to the SOMs after training was completed, and the most reactionary nodes of the SOM were associated with the coordinates at which the samples were captured in the metric map. The perceptual SOM therefore could be used for periodical position correction in later navigation, adding to the robustness of both local map navigation and global navigation as errors in the distance values and wall-following behaviors could be corrected at short distance intervals, ensuring that no error was allowed to accumulate in any direction.

The implementation of the hybrid topological and metric approach presented in this system was motivated by the potential of merging the topological and the metric paradigm into one representation to draw on the strengths and lessen the weaknesses of the representations in isolation.

The global topological map provided a very compact presentation, mapping the topography of the environment. Self-organizing maps proved to be very well suited for both metric and perceptual mapping. The open space surrounding global nodes were consistently mapped with great accuracy. The topology preserving nature of SOMs provided maps suited for navigation. The perceptual SOMs showed great robustness in operation, consistently mapping the perceptual space accurately.

Bibliography

[1] Elfes, A. (1987), "Sonar-based real-world mapping and navigation", IEEE Journal of Robotics and Automation, RA-3(3):249 - 265, June 1987

[2] Elfes, A. (1989), "Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation", PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1989.

[3] Moravec. H.P. (1988), "Sensor fusion in certainty grids for mobile robots", . AI Magazine, 9(2):61-74, 1988.

[4] R. Chatila and J.-P. Laumond (1985), "Position referencing and consistent world modeling for mobile robots", In Proceedings of the 1985 IEEE International Conference on Robotics and Automation, 1985.

[5] M. J.Mataric (1990), "A distributed model for mobile robot environment-learning and navigation", Masters thesis,MIT, Cambridge, MA, January 1990. also available as MIT AI Lab Tech Report AITR-1228.

[6] B. Kuipers and Y.T. Byun (1991), "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations", Journal of Robotics and Autonomous Systems, 8:47-63, 1991.

[7] Choset, H. (1996), "Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph.", PhD thesis, California Institute of Technology, 1996.

[8] Choset, H and Burdick, J.W. (1996), "Sensor Based Planning: The Hierarhical Generalized Voronoi Graph.", In Proc. Workshop on Algorithmic Foundations of Robotics, Toulouse, France, 1996.

[9] S. Engelson and D. McDermott (1992), "Error correction in mobile robot map learning", . In Proceedings of the 1992 IEEE International Conference on Robotics and Automation, pages 2555-2560, Nice, France, May 1992.

[10] D. Kortenkamp and T.Weymouth (1994), "Topological mapping for mobile robots using a combination of sonar and vision sensing.", In Proceedings of the Twelfth National Conference on Articial Intelligence, pages 979-984,Menlo Park, July 1994. AAAI, AAAI Press/MIT Press.

[11] D. Pierce and B. Kuipers (1994), "Learning to explore and build maps.", In Proceedings of the Twelfth National Conference on Artificial Intelligence, pages 1264-1271, Menlo Park, July 1994. AAAI, AAAI Press/MIT Press.

[12] H. Shatkay and L. Kaelbling (1997), "Learning topological maps with weak local odometric information", In Proceedings of IJCAI-97. IJCAI, Inc., 1997.

[13] H. Shatkay (1998), "Learning Models for Robot Navigation.", PhD thesis, Computer Science Department, Brown University, Providence, RI, 1998.

[14] B. Yamauchi and R. Beer (1996), "learning for navigation in dynamic environments.", IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, Special Issue on Learning Autonomous Robots, 1996.

[15] U.R. Zimmer (1996), "Robust world-modeling and navigation in a real world. ", Neurocomputing, 13(2-4), 1996.

[16] J.A. Castellanos and J.D. Tardos (2000), "Mobile Robot Localization and Map Building: A Multisensor Fusion Approach. ", Kluwer Academic Publishers, Boston, MA, 2000.

[17] M. Csorba (1997), "Simultaneous Localisation and Map Building.", PhD thesis, University of Oxford, 1997.

[18] G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba (2001), " A solution to the simultaneous localization and map building (SLAM) problem.", IEEE Transactions of Robotics and Automation, 2001.

[19] J. Guivant and E. Nebot (2001), "Optimization of the simultaneous localization and map building algorithm for real time implementation", IEEE Transaction of Robotic and Automation, May 2001.

[20] J.J. Leonard, H.F. Durrant-Whyte, and I.J. Cox. (1992), "Dynamic map building for an autonomous mobile robot.", . International Journal of Robotics Research, 11(4):89-96, 1992.

[21] P. Newman (2000), "On the Structure and Solution of the Simultaneous Localisation and Map Building Problem", PhD thesis, Australian Centre for Field Robotics, University of Sydney, Sydney, Australia, 2000.

[22] S.Williams, G. Dissanayake, and H.F. Durrant-Whyte. (2001), "Towards terrain-aided navigation for underwater robotics.", Advanced Robotics, 15(5), 2001.

[23] F. Dellaert, S.M. Seitz, C. Thorpe, and S. Thrun. (2000), "EM, MCMC, and chain ?ipping for structure from motion with unknown correspondence.", Machine Learning, 2000.

[24] S. Thrun (2001), "A probabilistic online mapping algorithm for teams of mobile robots.", International Journal of Robotics Research, 20(5):335-363, 2001.

[25] S. Thrun, D. Fox, and W. Burgard (1998), "A probabilistic approach to concurrent mapping and localization for mobile robots.", Machine Learning, 31:29-53, 1998. also appeared in Autonomous Robots 5, 253-271 (joint issue).

[26] C. Owen and U. Nehmzow, (1996), "Route Learning in Mobile Robots through Self-Organisation", Published by IEEE Computer Society, Proc. Eurobot 96. Kaiserslautern 1996.

[27] Duckett and Nehmzow (1998), "Mobile robot self-localisation and measurement of performance in middle scale environments. ", Robotics and Autonomous Systems, 24(1-2):57-69.

[28] Nehmzow, U. and Owen, C. (2000), "Robot navigation in the real world: Experiments with Manchesters FortyTwo in unmodified, large environments", In Robotics and Autonomous Systems, 33,2000.

[29] M. Mataric (1991), "Navigating with a rat brain: A neurobiologically-Inspired Model for Robot spatial representation", in Jean-Arcady Meyer and Stuart Wilson (eds.), From Animals to Animats, MIT Press 1991.

[30] N. Burgess And J. O'Keefe (1996), "Neuronal computations underlying the firing of place cells and their role in navigation", Hippocampus 7:749-762(1996).

[31] N. Burgess and J. O'Keefe and M. Recce (1993), "Using hippocampal 'place cells' for navigation, exploiting phase coding", in Hanson, Giles and Cowan (eds.), Advances in neural information processing systems 5, Morgan Kaufmann 1993.

[32] U. Nehmzow and T. Smithers (1991), "Mapbuilding using Self-organizing Network", Jean-Arcady Meyer and Stewart Wilson (eds.) From Animals to Animats, MIT Press, Cambridge Mass and London England, 1991,pp. 96-104.

[33] Tomatis, N., I. Nourbakhsh, and R. Siegwart (2003), "Hybrid simultaneous localization and map building: a natural integration of topological and metric", Robotics and Autonomous Systems, 44:3-14.

[34] Hafner V.V. (2000), "Learning Places in Newly Explored Environments", in Meyer, Berthoz, Floreano, Roitblat and Wilson (Eds.), SAB2000 Proceedings Supplement Book, Publication of the International Society for Adaptive Behavior, Honolulu.

[35] Bosse, M., Newman, P., Leonard, J., Soika, M., Feiten, W., and Teller, S (2003), "An Atlas framework for scalable mapping.", in Proceedings International conference on Robotics and Automation. 2003.

[36] A. Tapus, R. Siegwart (2003), "Incremental Robot Mapping with Fingerprints of Places"

[37] Yeap, W.K. and Jefferies, M.E. (1999), "Computing a representation of the local environment. ", Artificial Intelligence, 1999. 107. 265-301.

[38] Yeap, W.K. (1988), "Towards a computational theory of cognitive maps.", Artificial Intelligence 1988. 34. 297-360.

[39] Chown, E., Chaplain, S., and Kortenkamp, D., (1995), "Prototypes, location, and associative network (PLAN): Towards a unified theory of cognitive mapping.", Cognitive Science, 1995. 19. 1-51.

[40] OKeefe and Nadel (1978), "The Hippocampus as a Cognitive Map", Oxford University Press

[41] J. Borenstein and Y. Koren (1991), "The vector ?eld histogram - fast obstacle avoidance for mobile robots.", IEEE Journal of Robotics and Automation, 7(3):278-288, June 1991.

[42] J. Buhmann, W. Burgard, A.B. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. (1995), "The mobile robot Rhino.", AI Magazine, 16(1), 1995

[43] J. L. Crowley (1985), "Navigation for an intelligent mobile robot", IEEE Journal of Robotics and Automation 1985.

[44] J.J.Leonard, H.F. Durrant-Whyte (1989), "A Unified Approach to Mobile-Robot Navigation", Department of Engineering Science, University of Oxford, UK., September 28 1989.

[45] A.P. Dempster, A.N. Laird, and D.B. Rubin (1977), "Maximum likelihood from incomplete data via the EM algorithm.", Journal of the Royal Statistical Society, Series B, 39(1):138, 1977

[46] Thrun, S (1998), "Learning metric-topological maps for indoor mobile robot navigation", In Artificial Intelligence 99(1):21-71.

[47] Taylor C. J. and Kriegman D.J. (1998), "Vision-Based Motion Planning and Exploration algorithms for Mobile Robots", IEEE Transactions on Robotics and Automation, Vol. 14, No 3, June 1998, pp 417-426.

[48] J.A. Castellanos, J.D. Tards, et al (1997), "Building a global map of the environment of a mobile robot: the importance of correlations", in: Proceedings of the IEEE International Conference on Robotics and Automation, Albuquerque, 1997.

[49] OKeefe J. (1976), " Place units in the hippocampus of freely moving rat", Experimental Neurology, 51:78-109.

[50] Rosen, C. A. and Nilsson, N. J (1966), "Application Of Intelligent Automata to Reconnaissance", Technical Report . Stanford Research Institute, November 1966.

[51] Brooks, R. (1986), "A robust layered control system for a mobile robot", Robotics and Automation, IEEE Journal of $[legacy, pre - 1988]$ 2 (1): 14:23.

[52] Wooldridge M (2004), "An introduction to MultiAgent Systems", 5:89-104.

[53] S. Thrun (2002), "Robotic Mapping: A Survey"

[54] H. Baltzakis, P. Trahanias (2002), "Hybrid mobile robot localization using switching state-space models", in: Proceedings of the IEEE International Conference on Robotics and Automation, Washington, DC, USA, 2002

[55] T. Kohonen (1988), "Self Organisation and Associative Memory." Springer Verlag, Berlin, Heidelberg, 1988

[56] Hart, P.E. Nilsson, N. J. Raphael, B (1972), "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", SIGART Newsletter 37: 28:29.

[57] A. Kurz (1996), "Constructing maps for mobile robot navigation based on ultrasonic range data", IEEE Trans Systems, Man and Cybernetics B, Col 26 No 2 1996

[58] Thrun, A. Bcken (1996), "ntegrating grid-based and topological maps for mobile robot navigation", in: Proceedings of the National Conference on Artificial Intelligence, Portland 1996

[59] Huang, W. H. and Beevers, K. R (2004), "Topological mapping with sensing-limited robots", proceedings of the 6th International Workshop on the Algorithmic Foundations of Robotics (WAFR 2004), Utrecht, the Netherlands, July 11:13, pp. 367:382.

[60] I. Nourbakhsh, R. Powers and S. Birchfield (1995), "DERVISH: An Office-Navigaing Robot", , Artificial Intelligence Magazine, 16(2). 1995

[61] R. Simmons and S. Koenig (1995), "Probabilistic Robot Navigation in Partially Observable Environments", Proc. of the International Joint Conference on Artificial Intelligence. 1995.

[62] M.E. Lpez et al (2003), "Visually Augmented POMDP for Indoor Robot Navigation", rom Proceeding (378) Applied Informatics, 2003

Appendix A

Algorithms

A.1 Path-integration algorithm

Algorithm 2 Path-integration algorithm

heading = 0
xPosition = 0
yPosition = 0
previousLeftSteps = 0
previousRightSteps = 0
while robot moving **do**
  $deltaLeft = encoderLeftSteps - previousLeftSteps$
  $deltaRight = encodeRightSteps - previousRightSteps$
  $deltaDistance = 0.5 * (deltaLeft + deltaRight) * distancePrStep$
  $deltaX = deltaDistance * \cos heading$
  $deltaY = deltaDistance * \sin heading$
  $deltaHeading = (deltaRight - deltaLeft) * radiansPerStep$
  $xPosition+ = deltaX$
  $yPosition+ = deltaY$
  $heading+ = deltaHeading$
  $previousLeftSteps = encoderLeftSteps$
  $previousRightSteps = encoderRightSteps$
end while

A.2 A* algorithm

A.3 Boundary-tracing algorithm

Algorithm 3 A*(startNode,goalNode)

```
closedSet = empty set
openSet = startNode
gScore[startNode] = 0
hScore[startNode] = heuristicDistance(startNode,goalNode)
gScore[startNode] = hScore[startNode]
while openSet not empty do
  x = node in openSet with lowest fScore
  if x=goalNode then
    return generatePath(historicPath,goalNode)
  end if
  remove x from openSet
  add x to closedSet
  for n in neighboringNodes of x do
    if  n found in closedSet then
      continue
    end if
    nGScore = gScore[x] + heuristicDistance(x,n)
    nScoreBest = false
    if n not in openSet then
      add n to openSet
      hScore[n] = heuristicDistance(n,goalNode)
      nScoreBest = true
    else if nGScore ¡ gScore[n] then
      nScoreBest = true
    end if
    if nScoreBest = true then
      historicPath[n] = x
      gSscore[n] = nGScore
      fScore[n] = gScore[n] + hScore[n]
    end if
  end for
  return no path exists
end while
```

Algorithm 4 generatePath(historicPath,currentNode)

```
if historicPath[currentNode] is set then
  p = generatePath(historicPath,historicPath[currentNode])
  return (p + currentNode)
else
  return empty path
end if
```

Algorithm 5 Boundary tracing(wall-following type)

```
weightsRight = [10, 10, 5, 0, 0, −5, −10, −10]
weightsLeft = [−10, −10, −5, 0, 0, 5, 10, 10]
biasSpeed = 100
wallClose = FALSE
for i = 0 to 7 do
  if distance[i] > threshold then
    wallClose = TRUE
  end if
end for
if wallClose == TRUE then
  if follow right wall then
    weightsLeft[0] = −10
    weightsLeft[7] = −10
    weightsRight[0] = 10
    weightsRight[7] = 10
    if distance[2] >300 then
      distance[1]− = 100
      distance[2]− = 600
      distance[3]− = 100
    end if
  else if follow left wall then
    weightsLeft[0] = 10
    weightsLeft[7] = 10
    weightsRight[0] = −10
    weightsRight[7] = −10
    if distance[5] > 300 then
      distance[4]− = 100
      distance[5]− = 600
      distance[6]− = 100
    end if
  end if
  leftSpeed = biasSpeed
  rightSpeed = biasSpeed
  for i = 0 to 7 do
    leftSpeed += weightsLeft[i] ∗ distance[i] >> 4
    rightSpeed += weightsRight[i] ∗ distance[i] >> 4
  end for
else
  if follow right wall then
    leftSpeed=biasSpeed
    rightSpeed=biasSpeed/2
  else if follow left wall then
    leftSpeed=biasSpeed/2
    rightSpeed=biasSpeed
  end if
end if
```