

BUILDING TOPOLOGICAL MAPS FOR ROBOT NAVIGATION USING NEURAL NETWORKS

Philippe Künzle

Department of Computer Science
McGill University, Montréal

February 2005

A thesis submitted to McGill University in partial fulfillment of the requirements of the
degree of Master of Science

© PHILIPPE KÜNZLE, MMV

ABSTRACT

Robots carrying tasks in an unknown environment often need to build a map in order to be able to navigate. One approach is to create a detailed map of the environment containing the position of obstacles. But this option can use a large amount of memory especially if the environment is large. Another approach, closer to how people build a mental map, is the topological map. A topological map contains only places that are easy to recognize (landmarks) and links them together.

In this thesis, we explore the issue of creating a topological map from range data. A robot in a simulated environment uses the distance from objects around it (range data) and a compass as inputs. From this information, the robot finds intersections, classifies them as landmarks using a neural network and creates a topological map of its environment. The neural network detecting landmarks is trained online on sample intersections. Although the robot evolves in a simulated environment, the ideas developed in this thesis could be applied to a real robot in an office space.

RÉSUMÉ

Un robot évoluant dans un environnement inconnu a souvent besoin de construire une carte pour pouvoir naviguer. Une approche est de créer une carte détaillée de l'environnement avec la position des obstacles. Mais cette option peut utiliser beaucoup de mémoire surtout si le lieu à cartographier est spacieux. Une autre approche qui est similaire à comment l'humain construit une carte mentale, est la carte topologique.

Une carte topologique ne contient que les emplacements qui sont facilement reconnaissables (point de repère) et ces derniers sont connectés entre eux.

Dans cette thèse, nous explorons le problème de la création de carte topologique à partir de données de portées. Un robot dans une simulation utilise la distance à laquelle les objets autour de lui se trouvent (données de portées) et une boussole. À partir de ces informations, le robot détecte les intersections, les classe comme étant des points de repère en utilisant un neural network et crée une carte topologique de son environnement.

Le neural network détectant les points de repère est entraîné on-line sur des échantillons d'intersection. Bien que le robot évolue dans un environnement simulé, les idées développées dans cette thèse pourraient être appliquées à un vrai robot dans un bureau.

ACKNOWLEDGEMENTS

Many thanks to my advisor, Doina Precup, without whom this work would have been impossible. She provided wonderful supervision while reading numerous revisions and helped me clarifying ideas and concepts.

DEDICATION

I would like to dedicate this dissertation to my wife Corinne Künzle Saragas. She supported me and encouraged me every step of the way writing this work. I could not have done it without her. I would especially like to thank my parents who always believed in me from the beginning of my studies even when results could have been better. I also thank them for all the financial support.

TABLE OF CONTENT

ABSTRACT.....	2
RÉSUMÉ	3
ACKNOWLEDGEMENTS.....	4
DEDICATION.....	5
LIST OF FIGURES	7
LIST OF TABLES.....	9
CHAPTER 1 Introduction.....	10
CHAPTER 2 Background.....	13
2.1. Types of Sensors.....	13
2.2. Landmark Detection.....	18
2.3. Neural Networks for Landmark detection	21
2.4. Mapping	24
2.5. Navigation & Position Tracking	28
CHAPTER 3 Overview of the Simulation.....	31
CHAPTER 4 Landmark Detection	35
4.1. Range Data.....	35
4.2. Edge Detection.....	36
4.3. Neural Network.....	38
4.3.1. Input preprocessing.....	38
4.3.2. The Input.....	38
4.3.3. The hidden layers	39
4.3.4. The output	40
4.3.5. The training.....	40
CHAPTER 5 Topological Map.....	43
CHAPTER 6 Navigation for Topological Map Building	54
CHAPTER 7 Experimental Results	61
7.1. Landmark Detection with neural network	61
7.2. Topological Map Creation	68
7.2.1. Comparing different Landmark detection on different maps	68
7.2.2. Robot Slipping	72
7.2.3. Adding Noise to Range data	72
7.2.4. Sensor Failure	75
7.2.5. Human robot control vs Auto navigation.....	76
CHAPTER 8 Conclusions and Future Work	79
REFERENCES	81
Appendix A.....	84

LIST OF FIGURES

Figure 1 : Sonar Error Simulation in a Simple room [3]	15
Figure 2 : Low level sonar traces in with an ideal isolated obstacle [3].....	16
Figure 3 : Low level sonar traces in a realistic environment [3]	16
Figure 4: Laser Range Finder Device position with mirror	17
Figure 5 : Distinctive Places [Kuyper and Byun]	19
Figure 6 : Extracting the topological graph [8].....	26
Figure 7 : Exploration results with systematic and 10% random error [14].....	27
Figure 8 : Small Range Data Limit	32
Figure 9 : Range Data Representation on a Small Map.....	35
Figure 10 : Range Data in a Hallway.....	36
Figure 11 : Example of a Training Map.....	41
Figure 12 : Neural network Architecture 3-3-2-4-1	42
Figure 13 : Demo Map 1	43
Figure 14 : Ideal Topological Map for Demo Map 1.....	43
Figure 15 : Actual Topological Map by the Robot for Map Demo 1	44
Figure 16: Merging with existing node.....	45
Figure 17: Link Crossing	45
Figure 18: Legend	46
Figure 19: Erroneous position of nodes due to drift and slippage	48
Figure 20: Robot Range Data at wrong location & Local Range Data	50
Figure 21: Range Data Comparison.....	50
Figure 22: Direction decision function.	51
Figure 23: Range Data representation on simple map	53
Figure 24: Simple topological map.....	54
Figure 25: Bell Curve.....	56
Figure 26 : 1.Range Data and 2. Weighted Range Data with obstacle in front of the robot	57
Figure 27 : 1.Range Data and 2. Weighted Range Data with no obstacle in front of the robot	58
Figure 28: Evolution of the Mean Squared Error over Time during Training.....	62
Figure 29: Average Mean Squared Error and Standard Deviation Per Map	63
Figure 30 : Neural Network 9 on Map 16.....	64
Figure 31: Average Mean Squared Error and Standard Deviation per Neural Network ..	67
Figure 32: Demo Map 1	69
Figure 33: Topological Map of Demo1 using Edge Detection.....	69
Figure 34: Topological Map of Demo1 with Neural Network 4	69
Figure 35: Topological Map of Demo1 with Neural Network 9	69
Figure 36: Topological Map of Demo2 with Edge Detection	70
Figure 37: Topological Map of Demo2 with Neural Network 4	70
Figure 38: Demo Map 2.....	70
Figure 39: Demo Map 3.....	71
Figure 40: Demo Map 4.....	71
Figure 41: Topological Map of Demo 3	71

Figure 42: Topological Map of Demo 4	71
Figure 43: Demo Map 5	72
Figure 44: Topological Map created without slipping.....	72
Figure 45: Topological Map created with slipping.....	72
Figure 46: Average Mean Squared Error and Standard Deviation with Different Max Noise Values	73
Figure 47: Topological Map created without noise	74
Figure 48: Topological Map created with 0.5 unit maximum noise.....	74
Figure 49: Topological Map created with 1 units maximum noise	74
Figure 50: Topological Map created with 5 units maximum noise	74
Figure 51: Average Mean Squared Error and Standard Deviation with Failing Sensors .	75
Figure 52: Topological Map with 3 sensor failing.....	76
Figure 53: Map of the 3rd floor of McConnell Building	77
Figure 54: Topological map created with human control of the robot	77
Figure 55: Topological map with robot self navigation.....	77
Figure 56 : Training Maps	84
Figure 57 : On Training “Testing Maps”	85
Figure 58 : Testing Maps	86
Figure 59 : Program Organization	88
Figure 60 Flowchart of the Robot Update function	89

LIST OF TABLES

Table 1: Detailed Results for Neural Network 3

Table 2: Mean Squared Error per Maps & per Neural Network Data

Table 3: Evolution of the Mean Squared Error over Time during Training Data

Table 4: Comparing all Neural Networks on Two Different Maps

CHAPTER 1

Introduction

Mapping an unknown environment is a popular subject in robotics and artificial intelligence since it is often the first requirement for an autonomous mobile robot. In order for a robot to build a map, it must translate input information, which can be images from cameras, distances to surrounding obstacles or data from other sensory devices, into an abstract representation of the environment. This translation can be troublesome since input information is often noisy and inconsistent. A robot is frequently used in areas that a human cannot reach like a sewer or a distant planet. Human intervention in case of a problem is usually impossible. The map created by the robot must therefore be robust, predictable and reliable.

Mapping the environment is crucial. A robot's task could be to gather information, to carry objects from an origin to a destination, or to perform other tasks in the same manner as a human. For many of these tasks, the robot must have a map of the known area in order to navigate and it must be able to expand the map as needed when discovering uncharted areas. Without a map, an autonomous robot cannot choose an itinerary to accomplish a task.

Building a map is not a trivial problem. The robot must translate its first person view of the environment into an abstract top down view. Depending on the sensory device used, it can detect the relative position of obstacles through a sonar or a laser range finder, it can find obstacles by touching through a tactile sensor or identify obstacle from images received from a camera at regular time intervals. But all this data is relative to the robot's current position and must be processed to create a map. The main problem encountered is the noise in inputs due to the limited precision of the hardware used. A sonar for instance cannot perceive an obstacle that is too close, and it has a limited range of action.

Translating the data from the input devices into a map is the second problem encountered especially when no global positioning device is used. A robot may have problems tracking efficiently its position due to imprecision in odometry. On any surface a robot will drift and slip which results in inaccurate estimates of distance traveled. Cameras present another problem, the detection of separate objects in an image and the estimation of distances to that object is troublesome. Different techniques for distance estimation exist but they require a lot of information about the lighting condition and the environment.

There exist two main approaches to mapping an environment in robotics. Metric maps are a spatial representation of the environment describing the distances between objects. They are easy to build from range data but can be difficult to use for navigation. Topological maps on the other hand are a more abstract representation in which distinctive places are connected. Those connections represent existing paths from one place to the next. They can be built from an existing metric map or on the fly by detecting the distinctive places as the robot discovers the environment.

This thesis shows how a robot can autonomously create a topological map of an indoor environment using range data and compass as inputs. A formerly trained neural network is used for landmark detection. We chose to use a neural network because it can be trained online to recognize different sensory situations as landmarks. Also, neural networks are known for their robustness to noisy data. We are building a topological map since the latter require less memory than a metric map and is easily used for navigation. Navigation on a metric map requires complex image processing algorithms in order to extract an obstacle free path. Since the topological map is already an abstract version of the environment, navigation only requires following the linked distinctive places using a shortest path algorithm. The creation of the topological map on the other hand is more complex than that of a metric map. We developed a simulation in order to test our concept. All results are gathered from the simulation. This work is a proof of concept.

The thesis is organized as follows. In the second chapter, we will look at the background of map creation in robotics, the hardware requirements and limitation and different

approaches solving this problem. The third chapter will go over the simulation we created in order to test the concept. The next three chapters review how we accomplished landmark detection, topological map creation and navigation of the robot. The seventh chapter shows the experimental results of the landmark detection, topological map creation and navigation under different conditions. Finally the last chapter presents conclusions and directions for future work.

CHAPTER 2

Background

Mapping an unknown environment is a crucial task for most autonomous robots. It requires processing sensor and/or camera information in order to build a map, which is then used by the robot in order to localize itself. The map building algorithm must be flexible enough to compensate for hardware errors, such as sensor errors and position errors due to slippage.

There have been many different approaches for building maps from sensor data over the past few years. Based on the types of maps being constructed, these can be classified as building metric maps [16], topological maps [7, 5, 14] or both types of maps [6, 8, 10]. A metric map represents the distances between obstacles. A topological map is an abstraction of the environment which connects distinctive places or landmarks.

Connections between landmarks represent an obstacle-free path between two landmarks. In this chapter, we will look at sensor types and then review different approaches in landmark detection, map building and navigation.

2.1. Types of Sensors

The two most common sensor devices used on mobile robots are cameras and range sensors. Cameras give more information about the environment, but it is much harder to extract information from images. In order to create a map of the environment, we must extract distances from the robot to the surrounding obstacles. With those distances, we can then reconstruct the map. If the robot uses only one camera, it is very hard to determine how far objects are. “Shape from shading” [1] is a standard technique for this problem, but it requires precise information about lighting conditions in order to work properly. If the robot uses two cameras positioned a few centimeters apart, it is easier to determine the distance to obstacles using stereovision techniques [1]. The two images captured at one point in time are similar but not identical due to the difference in position

of the two cameras. The robot needs to match each pixel of the image captured by the right camera to each pixel of the image taken by the left camera and then uses geometric properties to infer distances. Another difficulty related to cameras consists in finding objects. Even with depth information, discerning objects from background is a complex problem. So cameras can provide much information about the surrounding but deciphering this information is not a trivial task.

Range data given by infrared, sonar or laser rangefinder sensors contains explicitly the distance to an obstacle. With multiple sensors, the robot can obtain the distance to all obstacles at the current location [2]. Range data provides less information, which facilitates parsing and interpretation, especially for the task of finding landmarks in an indoor environment.

A comprehensive description of the different types of range sensors is given in the book “Computational Principles of Mobile Robotics” by G. Dudek and M. Jenkin. We summarize here the information relevant to this thesis.

The infrared proximity detector, although subject to environment interference (sunlight), gives a fast and inexpensive solution to range data. The distance is estimated from the strength of the infrared signal sent out and received back after bouncing on an obstacle. Sonars, which have a long history in the underwater world, use sound to determine the distance to an obstacle. Distances are estimated using time of flight, phase and the attenuation of the reflected signal.

A transducer using ultrasound, such as the Polariod SensComp 7000, has a range from 0.15 meters to 10.7 meters with a resolution of ± 3 mm to 3 meters [23]. Sonar transducers are widely used in robotics because of low cost [3].

In the case of a sonar, the transducer sends a sonar “chirp” that will be reflected onto obstacles in front of the transducer. Ideally, the reflected “chirp” should be received by the sonar as a single clear sound at a later time. Knowing the speed at which the sound travels, one can compute the distance at which the obstacle is from the time it took for the “chirp” to travel to the obstacle and back.

There are multiple problems inherent to the system that generate errors in the results. First, once the sonar emits a pulse, it goes through a transition phase during which it is unable to listen for the returning pulse [3]. If an obstacle is too close to the transducer, the reflected

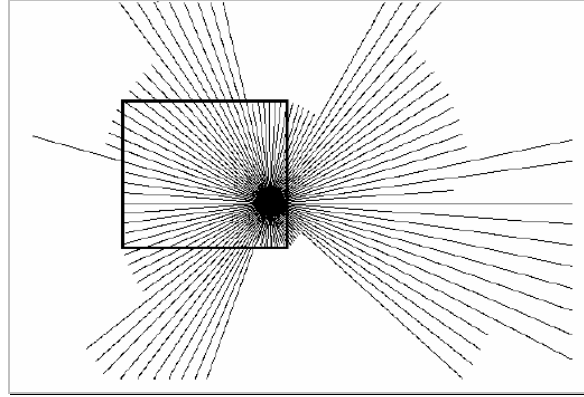


Figure 1 : Sonar Error Simulation in a Simple room [3]

signal will return during this transition phase and will never be “heard” by the sonar. As we can see on Figure 1, when the wall is too close to the sonar, the line representing the estimated distance is very long because the sonar never “heard” the reflected signal.

Secondly, “the sonar chirp is not an infinitely narrow beam” [3]. The off-axis sonar chirp can be a source of error in the reflected signal. Thirdly, the reflected signal heard by the transducer may have been reflected on multiple surfaces, hence it may not always represent a linear path between the robot and the obstacle.

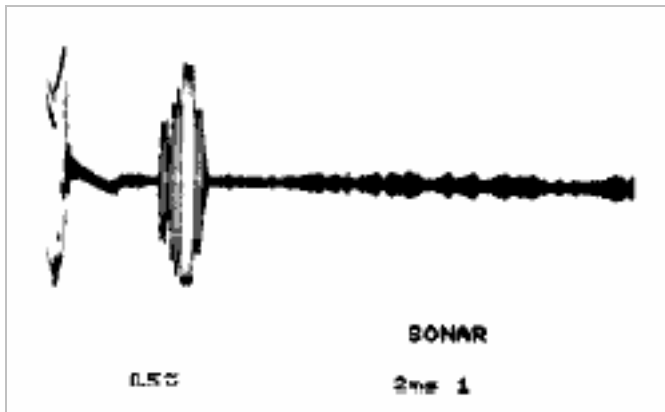


Figure 2 : Low level sonar traces in with an ideal isolated obstacle [3]

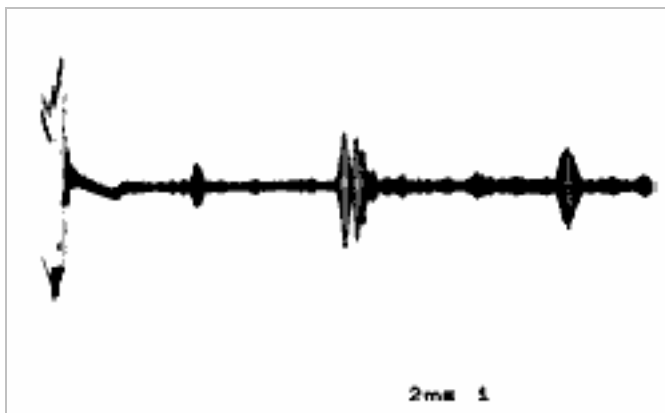


Figure 3 : Low level sonar traces in a realistic environment [3]

Figures 2 and 3 taken from Dudek and Jenkin [3] show low level sonar traces after sending a pulse. We see on both figures the initial pulse on the left hand side of each figure. In Figure 3, the sonar pulse takes multiple paths and we get multiple reflected signals while in Figure 2 we see one large reflected signal which corresponds to one obstacle.

Radars work the same way as sonars but use radio waves instead of sound waves. Laser rangefinders use one of three methods to find the distance of an obstacle: triangulation (geometric relationship between emitted light beam and reflected light beam), time of flight (time between emitted light beam and reflected light beam) or phase difference (difference between the phase of the emitted light beam and the reflected light).

Laser rangefinders can be found in a wide variety of ranges; some have a 2 – 600 meters range with ± 15.3 cm accuracy [22], others have been used to compute the distance from the earth to the moon [24]. Laserrange finders are more accurate, but since they are more expensive and present health hazard to the eyes, they are less commonly used on robots. Lasers have a wide variety of range capability depending on what technique is used. A laser using time of flight can measure from a few meters up to tens of kilometers with an accuracy of 1 mm [24]. The time of flight method is very similar to the one discussed for sonars. The distance is estimated using the time it took the light to travel to the obstacle and back. Knowing the speed of light, the distance traveled by the light can be computed very accurately. Unlike sonars, lasers do not have the problem that the signal may take multiple paths. In fact, it is possible to have dense environment range data. [18]. One can get an accurate distance to obstacles for every pixel of a digital picture.

In order to measure distances in different directions with only one laser, one can use a scanning mirror [18]. Moving a mirror instead of the entire laser is faster, demands less energy and does not affect the result.

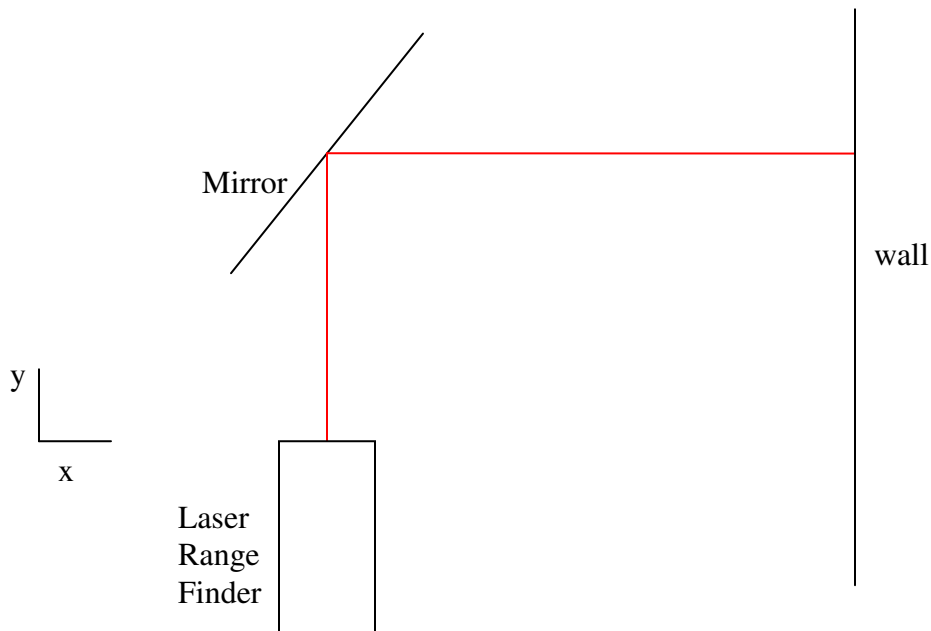


Figure 4: Laser Range Finder Device position with mirror

In Figure 4, the laser is pointing toward the ceiling and a mirror placed at a 45 degree angle reflects the laser horizontally. By rotating the mirror around the y axis, one can obtain range data all around the robot using only one laser rangefinder device.

2.2. Landmark Detection

The term “landmark” does not have an official definition, but everyone seems to agree that it is a “reference point” that helps a robot recognize its current location [7]. While some researchers have used artificial landmarks such as ultrasonic beacons, reflective tape or visual patterns to help robot localize itself [19, 20], others have tried to use natural landmarks such as intersections [14], signs meant for people [15] or let the robot choose its own natural landmarks [7]. This latter type of landmark is what we are using in this thesis.

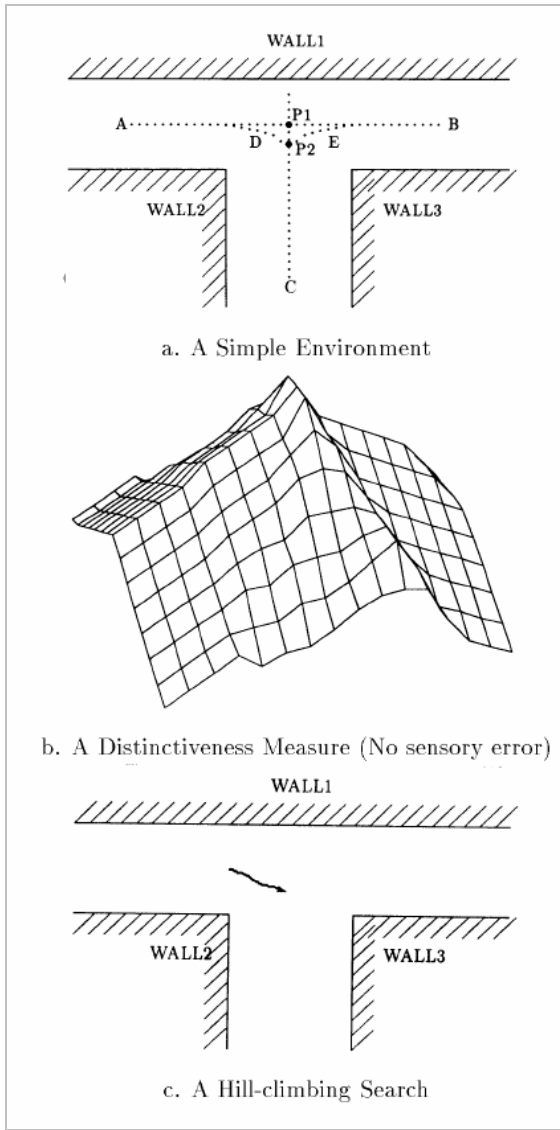


Figure 5 : Distinctive Places [Kuyper and Byun]

used to find the closest distinctive place as shown in Figure 5 c. When traveling from one distinctive place to the next, the robot uses a similar geometric feature. It follows the midline of a corridor. When the robot enters a larger room, it follows the wall. The hill-climbing search for distinctive place on continuous sensory feedback is a robust algorithm even with sensory and movement errors.

Rizzi, Maio and Golfarelli use a set of landmark templates and match them to real-time inputs as the robot evolves in an environment. Every sonar pattern is stored in a high resolution occupancy grid. But because the matching algorithm has prohibitive

Kuipers & Byun use the distinctiveness of the surrounding to find their landmarks (called Distinctive Places [14]). Their distinctive places are closely related to the topological map the robot is creating since each node in the topological map corresponds to an intersection. Each place has to be locally distinctive within its immediate neighborhood. When considering a two dimensional map of a local neighborhood (Figure 5 a), “the most distinctive points occur where the lines intersect, near the center” [14]. Those lines are defined by the geometric feature Equal-Distances-to-Near-Object (Figure 5 b). Once the robot recognizes that it is in the neighborhood of a distinctive place, a hill-climbing strategy is

computational costs[11], they introduce a second, low resolution grid which is a “weighted average of the occupancy previsions for all the corresponding cells in the first occupancy grid” [11]. When a region of interest is found on the low resolution occupancy grid, the matching template algorithm is performed on the high resolution occupancy grid but only in the region of interest. The actual matching is done by “comparing the occupancy prevision gradients for each pair of corresponding cells” [11].

Sim and Dudek explore another aspect of landmark detection [32]. They present a method that addresses the problem of learning a set of models of visual features that are not affected by scale or translation. This approach can be used for initial localization, position tracking or other visualization tasks. The learning process is divided in four parts: Feature Extraction, Feature Tracking, Generative Feature Model and Model Evaluation. Potential features are first extracted from training images using the SIFT feature detector [33]. The area around every point detected by SIFT feature detector is presumed important. From the initial training image, nearby training images are inserted while the algorithm tries to track the feature detected initially. In the generative feature model stage, the robot must learn the features from images taken from known camera positions.

The observation of a feature is defined as

$$z = \begin{bmatrix} s \\ u \end{bmatrix}$$

where s represents the scale of the feature and u represents the position of the feature in the image. The observation z is a vector-valued function of the pose of the camera. In order to learn this function, the robot models each element of z as a linear combination of radial basis functions (RBF).

$$z = \sum_i^n w_i G_i$$

where n is the number of training poses, w are weights and G is an exponentially decaying RBF. Optimal weights are computed as the solution to the linear least squares problem. Finally the quality of models is evaluated using leave one out cross validation.

Data from sensors needs to be interpreted in order to detect landmarks. Landmark detection can be seen as a binary classification problem. Such a problem can be approached in many different ways. There exist many different classification algorithms such as decision trees, support vector machines or neural networks to name just a few options.

Any classification algorithm could potentially work in our problem. We chose neural networks because they can naturally be trained online (which is not the case for other classification algorithms). Also neural networks are known to work well with noisy or occluded data. An example in chapter 7 of the Neural Network Design textbook [4] shows how well a neural network is able to recover patterns of which 50% have been occluded.

2.3. Neural Networks for Landmark detection

A neural network is a collection of simple computational units called neurons, which are organized in layers. There is an input layer, one or more hidden layers, and an output layer. Each layer can have multiple neurons. In the feed-forward architecture, the output of every neuron of one layer is connected to the input of every neuron of the next layer.

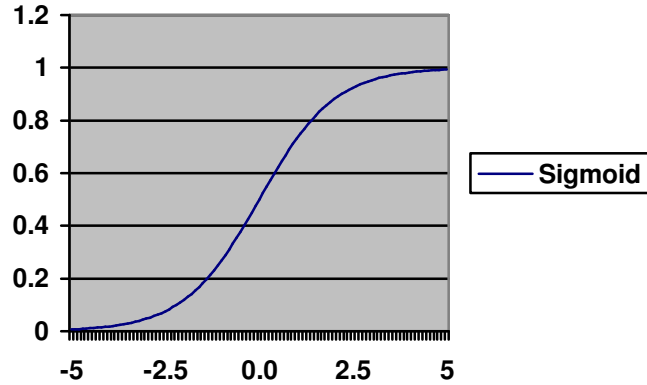
The output signal of a neuron is the weighted sum of its inputs plugged into a transfer function [4]. The transfer function modifies the signal to emphasize whether the neuron is mostly on, or mostly off.

So the output vector of the layer i which has n neurons would be computed as follows:

$$a^i = f^i(W^i \times a^{i-1} + b^i)$$

where i is the layer number, W is an m by n matrix of weights, a^{i-1} is the vector of inputs of size m which are outputs from the previous layer (for all layers except the first), b is the vector of bias weights of size n and f the transfer function. In other words, for every neuron, the output is obtained as a linear combination of its inputs, modified by a transfer function. Typically, the transfer function for a backpropagation neural network is a Sigmoid:

$$f(n) = \frac{1}{1 + e^{-n}}$$



Neural networks have already been used for landmark detection. S. Thrun used a neural network in order to have the robot detect self-selected landmarks [7]. In his approach, the robot has a camera that can tilt and pan (active perception) and a sonar. Both sonar readings and camera image are used for landmark detection. The neural network is trained to minimize the approximate Bayesian localization error which we explain next.

At any time, the robot has some belief about its location. It keeps a probability density over all locations l , $\hat{P}(l)$. As the robot moves or rotates, the certainty decreases since robot motion is subject to slippage and drift. The motion command a is described by a transition density, $P_a(l|\tilde{l})$. This is the probability that the robot is at location l knowing that it was at location \tilde{l} after executing the command a . In order to make sure that the robot maintains a high certainty of its position, it queries its sensors at regular intervals to check if any landmarks can be observed. Obviously, the probability to observe a landmark f_i depends on location l . $P(f_i|l)$ is the probability that f_i is observed knowing that the robot is at location l . $P(l)$ denotes the prior belief as to where the robot might be.

Each landmark found is used to improve the precision of the density function. Maximum likelihood or Bayes estimation can be used to specify a single location instead of a density function. The density function is updated as follows:

1. Initialization $\hat{P}(l) \leftarrow P(l)$

2. For each observed landmark vector f do:

$$\hat{P}(l) \leftarrow P(f | l) \cdot \hat{P}(l)$$

$$\hat{P}(l) \leftarrow \hat{P}(l) \left[\int_L \hat{P}(l) dl \right]^{-1} \quad (\text{normalization})$$

3. For each robot motion a do:

$$\hat{P}(l) \leftarrow \int_L P_a(l | \tilde{l}) \cdot \hat{P}(\tilde{l}) d\tilde{l}$$

The algorithm needs to minimize the Bayesian a posteriori error E . But the true Bayesian localization error can not be computed because the probability $P(f_i | l)$ is unknown.

So the neural network landmark detector is trained so as to minimize an approximation of the error, \tilde{E} , based on examples. The set of example is defined as $X = \{ \langle l, s \rangle \}$, where s is the sensor measurement and l is the location where the sensor measurement was taken.

The samples are used to provide an approximation of $P(f_i | l)$ and the approximated error is computed as:

$$\tilde{E} = \sum_{\langle l, s \rangle \in X} \sum_{\langle \tilde{l}, s \rangle \in X} \left\| l - \tilde{l} \right\| \cdot P(l) \cdot \hat{P}(\tilde{l}) \cdot \sum_{f_1=0} \sum_{f_2=0} \dots \sum_{f_n=0} \left(\prod_{i=1}^n P(f_i | l) \cdot P(f_i | \tilde{l}) \right) \cdot \left[\sum_{\langle \tilde{l}, s \rangle \in X} \left(\prod_{i=1}^n P(f_i | \tilde{l}) \right) \hat{P}(\tilde{l}) \right]^{-1}$$

where $\| \cdot \|$ denotes a norm²

The neural network is trained with gradient descent to minimize \tilde{E} . The weights and biases $w_{i\mu\nu}$ are iteratively modified as follows:

$$w_{i\mu\nu} = w_{i\mu\nu} - \eta \frac{\partial \tilde{E}}{\partial w_{i\mu\nu}} \text{ with } \eta \text{ being the learning rate.}$$

This neural network is not trained with supervised learning; “no target values are generated” [7]. It is trained with gradient descent to directly minimize \tilde{E} . The landmark detection “emerges as a side effect of minimizing E” [7].

Our approach is similar to Kuipers & Byun’s method, but instead of using a hill-climbing algorithm to detect landmarks, we are using a neural network. Unlike Thrun, we use a standard supervised learning setting, in which the neural network is trained with previously labeled data. Although Kuipers & Byun’s algorithm is robust, we believe that a neural network is more flexible. The neural network can be trained to recognize any kind of sensor situation depending on the environment it will be mapping. Also, as mentioned earlier, neural networks work well with noisy data, and can potentially find intersections no matter how the robot approaches them. We are exploring how well a neural network is capable of matching a pre-learned set of templates to places in an unknown environment.

2.4. Mapping

Maps representing an environment can be classified in two groups: metric maps and topological maps. There has been a lot of work in robotics and artificial intelligence using either or both of these methods to map an unknown environment.

The most common metric map, called grid map is a two dimensional array of values where each value represents the occupancy [7] corresponding to obstacles in the environment. It has been used in many robotics projects [7, 16, 26, 27, 28]. The robot creates an occupancy grid map of the location of walls and obstacles corresponding to the environment it is exploring. This kind of map will keep more information and therefore, it can use an excessively large amount of memory depending on the necessary resolution. Chalita and Laumond proposed different types of metric maps using sets of polyhedra

[17]. This alternative approach is still subject to the same size issue as other types of metric maps.

A topological map only keeps key elements of the environment (landmarks) and their relative location to each other. It is usually represented as a graph. Landmarks are represented as vertices and the direct paths between them as arcs.

Topological maps are much easier to use since they are at a higher level of abstraction. Finding a path from one point to another on the map only requires finding the shortest path between two nodes following the arcs. Different topological map representations can be found in many robotics projects [14, 8, 10, 5, 30].

People interested in robotics have mainly studied metric maps while people working in AI seem to be mainly interested in topological maps [9]. More recently, some have been interested in using both because each type of map has something to offer. Grid maps are easily built and facilitate computation of shortest paths while topological maps allow efficient planning, accurate positioning is not required and they are convenient to represent symbolic planner & language interface [8].

Building a metric map from sensors is fairly easy [8]. One approach is to build the topological map from the metric map. Once the metric map has been built, free space in the metric map is partitioned into a small number of regions. From the free space in the metric map, Thrun and Bücken draw the Vornoi diagram, find the critical points (points

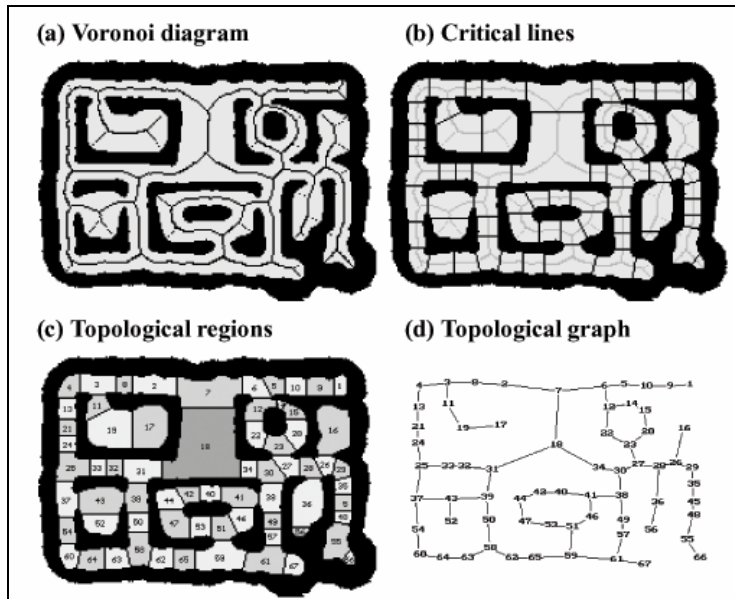


Figure 6 : Extracting the topological graph [8]

[8]. Each region becomes a node of the topological map. If two regions are next to one another, this will correspond to a link connecting the two nodes representing the two regions.

Figure 6 shows the different steps from the Vornoi diagram to the topological graph. Fabrizi and Saffioti have used a similar approach to build topological maps from metric maps. They use the watershed algorithm [13] to partition the free space into a set of connected regions [10].

Those two approaches create topological maps after creating the metric maps. Although the methods are interesting, they do not allow having the topological map created on the fly and therefore, the robot cannot take advantage of the topological map as it is discovering the environment. They also require more memory in the robot to store both types of maps. Even if the metric map is discarded after building the topological map, the robot needs to have enough memory to have both types of map in memory at the same time.

on the Vornoi diagram that minimize clearance locally [8] and deduce the critical line “obtained by connecting each critical point with its basis points” [8]. Each critical point must be part of the Vornoi diagram and the clearance of all points in an ϵ -neighborhood of each point in the free space is not smaller. The lines partition the free space into regions

Some other approaches successfully created topological maps without creating a metric map of the environment first. U. Zimmer experimented with a robot equipped with light sensor, tactile sensor and an odometer [5]. From those inputs only, the robot inserts nodes in an initially empty topological map using two growing strategies: spontaneous insertion [5] and statistical insertion[5]. The first strategy inserts a node when the distance between the previous node and the current estimated position exceeds a certain threshold. Statistical insertion adds nodes every time the classification error exceeds a threshold. Each inserted node is “classified” in the topological map and has a “classification error attached to it.

Zimmer’s method creates the topological map on the fly, but the nodes of the topological map do not represent a strategic place, they are placed regularly to keep track of the current position for localization. In this thesis, we explore the use of landmarks as distinctive navigational place such as intersections, turns, or rooms. This approach is motivated by our personal observations on how humans seem to memorize paths from one place to another. We seem to remember important places where a decision has to be taken, as well as how long it takes to go from one important place to the next.

Kuipers & Byun have developed a method for topological map creation by finding landmarks at intersections [14]. Their robot keeps metric information at landmarks and along the arcs leading from one landmark to the next. The robot can then recreate a metric map from the topological map. An illustration of this approach is presented in Figure 7. As we can see, the large room is not detected as a landmark, but as an intersection at each hallway. The landmarks only represent intersections and cannot

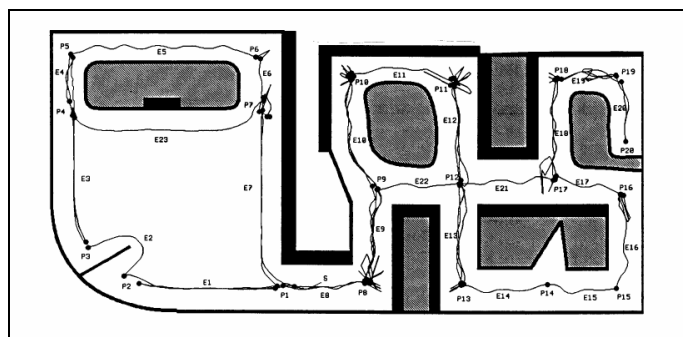


Figure 7 : Exploration results with systematic and 10% random error [14]

represent a room. This approach does not allow choosing what a landmark should be. We believe that being able to mark a room as one unique node makes the topological map closer to how a human would conceptualize a map, it simplifies the topological map and makes it easier to use for navigation. In our approach, we are creating similar maps using a different, more flexible, landmark detection technique where we can decide to have a room as a landmark by adding a room to the neural network's training set.

2.5. Navigation & Position Tracking

Building a map requires the robot to approximate its current location and have a strategy to explore the entire environment. Those two requirements are mandatory in order for a robot to create a map without human intervention.

Localization “involves determining the position of the agent in the environment from a set of sensor readings”[31]. Position tracking, which is a type of localization, “refers to the problem of compensating slippage and drift while the robot is moving”[7].

A robot needs to know its current position and orientation on a map of the environment in order to navigate to a goal. There are two main methods used in robotics for localization. One uses a form of global positioning device such as Global Positioning System (GPS) or the new European system: Global Navigation Satellite System (GNSS) [21, 25].

Although global positioning simplifies greatly the task of positioning, it is not always a viable choice. GPS may not work in the environment in which the robot is evolving (underground, Mars, Moon). The other option often chosen in robotics [14, 8] is the odometer with a compass. Knowing how much the robot traveled and the direction in which it traveled lets the robot estimate the new position with respect to its previous position. But odometers are subject to errors due to drift and slippage so position tracking becomes crucial for this method to work well.

The data gathered by a robot has a certain degree of uncertainty which in some cases can add up to return large errors. For instance, when the only method for estimating the robot's current position is the distance traveled and orientation, errors due to slippage can

have a devastating effect. After 15 minutes of robot operation, the position error can attain 11 meters [8]. When this is the case while building a metric map, the resulting map looks distorted: wall positions and orientations are inaccurate.

Thrun and Bucken integrated three source of information to resolve this problem: wheel encoder, map correlation and wall orientation. Wheel encoders, which measure the number of revolutions of the robot's wheels, can be used to reduce the error. According to [8], odometry based on their measurements is very accurate over short time intervals. Every time the robot interprets a sensor reading, it creates a "local" map. Map correlation is used to compare the local map with the global map providing a second source of information on the robot's position. Finally, checking the wall orientation solves the problem related to the rotational error.

We address this issue by checking the position of the robot every time it passes through a known node of the topological map. We do not have to take rotational error into consideration since our robot has access to a compass at all time. This will be further explained in chapter 5.

We separate the navigation in two level of abstraction. At a higher level, the robot needs to know where it goes on a map and what path it must take to reach its goal. At a lower level, the robot must follow the path decided by the higher level of abstraction while avoiding collision with obstacles.

For the higher level of abstraction, P. Gaussier and S. Zrehen have studied an approach of navigation imitating the functioning of animals [12]. Using a neural network, they use information from the environment and needs such as hunger, pain and tiredness as input for a neural network trained using unsupervised learning. The robot is able to make decisions based on current needs (e.g. hunger), memorized location (e. g. position of food), and current information (e.g danger or obstacle to be avoided). They use a simple neural network to accomplish robot navigation and goal management. This approach, although conceptually interesting, is too complex for the task we want to accomplish.

Kuiper & Byun use an exploration agenda [14]. Their robot keeps a list of all places visited where an unexplored direction still exists. Each time the robot leaves a place

where there exists at least one more direction in addition to the one it is taking, it adds the place and the direction to the list. In order to delete direction from the list, the robot has to reach the place again and follow the unexplored path. Our method for map building is similar, but instead of keeping an extra list of unexplored directions and places, we keep this information in the topological map at each node. The robot will recursively search the topological map until it finds the next closest unexplored direction.

CHAPTER 3

Overview of the Simulation

For the purpose of illustrating the use of neural networks for mapping, we will use a simulated environment in which the robot can gather data at will. We simulate navigation through building environments, and the sample maps we use are presented in the Appendix.

The robot navigates either in manual or in automatic mode. When the robot is in manual mode, the user controls the robot. Range data and the position on the map are displayed on the screen. When the robot is in automatic mode, it tries to explore the environment without human intervention. The simulated robot handles the topological map building in both modes. The only knowledge of the environment is through its sensors which are: laser range finder, compass and odometer. From those inputs, the robot detects landmarks, updates the topological map each time a landmark is detected and controls its acceleration and turn when in automatic mode. The distance traveled by the robot depends on the time step chosen for the simulation. Note that the robot movement has its own parameters, and it does not depend on the computer performance.

$$dist = \Delta t \cdot speed$$

We simulated a basic laser range finder because laser range finders are more accurate and are simpler to simulate. We provide a couple of parameters in the simulation to set the maximum range limit and the number of directions that will be covered. The maximum range limit simulates the fact that all range finders have a distance limit. Any obstacle that is further than that threshold will be seen as if it was at that threshold. Since the range data will be sent to a neural network and neural networks work better with values between -1 and 1, the range data is then normalized between 1 and -1. So for every value in the range data array we have:

When $rangedata[i]$ is larger than $threshold$

$Rangedata[i] = threshold$

$$In[i] = \left(\frac{2 \cdot rangedata[i]}{threshold} \right) - 1$$

The threshold represents the maximum range limit. Note that choosing a threshold that is too small would blind the robot because all obstacles would appear at equal distance. No actual obstacle would be reached by any sensor in any direction. The threshold should be chosen realistically, based on the capabilities of actual sensors.

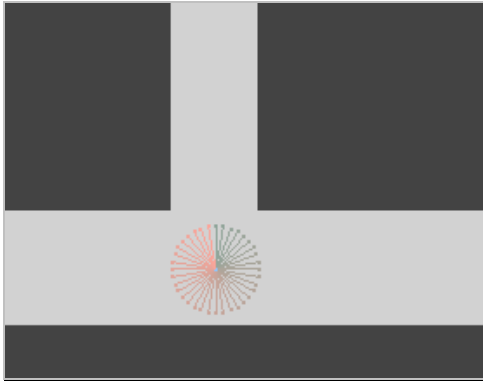


Figure 8 : Small Range Data Limit

Choosing a threshold that is too large introduces other problems. Because the input is normalized; the largest value will always become 1 and the smallest -1. If the variation between the largest distance and smallest distance is important, all details in the range data will be smoothed out. Two similar T-intersections except one of them with a long hallway will generate very different input to the neural network because of that normalization. If a large threshold is desired, a different way of encoding the inputs for the neural network may be necessary. We do not explore this further in the thesis.

Hallways in the maps we created are between 4 and 10 units wide and up to 128 units long. The width of a real hallway is about 2 meters. As mentioned earlier, some laser range finders can estimate distances up to 600 meters. We chose to simulate a laser range finder that has a limit of 25 meters, which means choosing a threshold of 50 units. This

threshold is a reasonable choice because it will catch the details of hallways without creating problems for the neural network input.

The number of directions covered is set by the number of laser range finders we desire to use or by the number of direction that one laser range finder can cover. The number of input neurons of the neural network is then set to match the number of directions that our device can cover.

In order to simulate environments, we load grid maps into the program. The program is using information from a metric map to simulate the sensor readings. However, the robot does not have a global knowledge of the metric map nor does it create one. It is only getting information about the surroundings through the simulated sensors.

We simulate three types of sensor error: range data noise, errors in odometry due to slippage, and sensor failure. Uniform noise is added to the range data values to simulate error in range data due to hardware precision. A parameter of the simulation controls the maximum noise. Sensor failure sets the value of certain directions in the range data to 0 as if the sensor was not working. This allows us to see how the robot reacts to this situation.

Error in odometry is based on a simplified simulation of slippage. The error in odometry is computed as follows:

$$\text{slipping} = (\text{previous_speed} - \text{current_speed}) * \text{slipping_factor} * \text{elapsed_time}$$

The value slipping is added to the distance traveled by the robot at that particular time. The slipping factor is an arbitrary number between 0 and 1. The odometry error is computed every frame. For example, if the slipping factor is 1, *previous_speed* is 0, *current speed* is 5 units/sec, and the *elapsed_time* is 0.5 second, the robot should move 2.5 units but *slipping* will be equal to -2.5 and therefore the robot will not move. Next frame, if the robot's speed is still 5 units/sec and the elapse time since last frame is again 0.5, the robot will move 2.5 units since the slipping will be null. This simplified simulation of slippage will introduce a small error in odometry for every acceleration or

deceleration of the robot. Even though this is not a realistic model of slippage and drift, it adds enough odometry error to test the robustness of our learning approach.

We didn't simulate any rotational error because the robot has access to a compass at all time. The compass error should be minimal and therefore should not affect the creation of the topological map or navigation.

We note that other standard robotic simulators such as CARMEN [34] or MMRA [36] which includes Robodaemon [35] could be used just as well for the experiments. We used our own simulator mostly out of convenience.

CHAPTER 4

Landmark Detection

The robot needs to decide on each step whether it should have a landmark at that location or not. Once a landmark is discovered, it is sent to the function that handles the updating of the topological map. Hence, the topological map is built incrementally, during the exploration of the environment. We consider two different algorithms for landmark detection: an edge detection algorithm and neural network. We describe both of them below.

4.1. Range Data

As described in Chapter 3, we use laser range finder sensors as input for the robot. An example of a sensor setup is given in Figure 7 below. A sensor is placed at every 10 degrees and therefore, the input that the robot gets consists of 36 distances all around it. Although robots commonly have a sensor every 15 degree for a total of 24 measurements [8, 5] or sometime even less, it is not unconceivable to have a total of 36 measures, especially if we use the panning mirror technology described in Chapter 2.

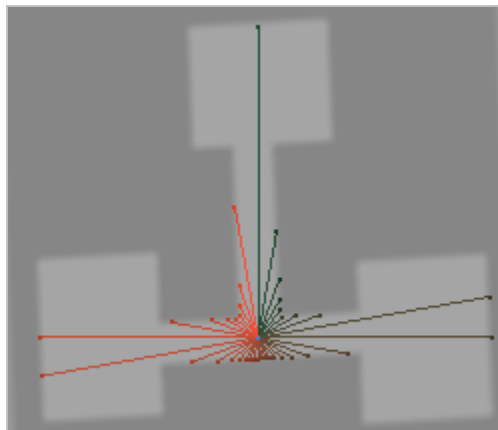


Figure 9 : Range Data Representation on a Small Map

4.1. Edge Detection

This technique is used as a reference point with which we will compare the results of the learning algorithm. The main idea is that we want the algorithm to detect a landmark when there is a change in the number of possible directions. For example, a landmark should be detected when coming from a hallway which allows for two possible directions of movement (forward and back) and getting into a T-intersection which allows three directions (back, left and right).

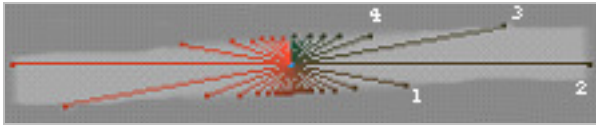


Figure 10 : Range Data in a Hallway

In the hallway situation, the distance in the directions pointing down the hall will be significantly larger than the directions pointing towards the walls of the hall. When comparing adjacent distance of the entire range data, there will be a sudden and large “jump” in size of the distances from one value to the next in the direction pointing to the end of the wall (e.g. distance 1 & 2 in Figure 8). By counting the number of large differences between adjacent values in the range data and dividing it by 2, we have an estimate of the number of possible directions in which the robot can travel.

Unfortunately, this way of counting may miss one count and find a odd number (distance 2, 3, 4 may not be counted as 1 large difference because the distance 2 and 3 and the distance 3 and 4 may be under the threshold). In order to handle this problem, if the division gives a float, we truncate the floating point and add one.

A landmark will be detected when the number of possible paths changes. When the robot moves from a hallway to a T-intersection, the number of counted directions for a hallway is two, and the number of directions of a T-intersection is three. That change will be detected as a landmark.

This method of landmark detection is rotational invariant as long as we have a reasonable amount of sensors around the robot. In Figure 10, if the robot were to rotate clockwise, the sensor labeled '3' would measure the distance to the end of the hall while the sensor labeled '2' would now measure the distance to side of the wall. It would not affect the number of possible paths and therefore the algorithm would not detect a landmark as a result of this rotation.

This simple algorithm will give us more landmark than desired because when the robot will move from the T-intersection back to a hallway, it will detect an extra landmark that may not be desired. But this gives us a quick and simplified approach to landmark detection.

4.2. Neural Network

The neural network used in this project is based on a back propagation neural network with some modification. It has 2 hidden layers and the input layer has an input history that is also fed as input into the neural network.

4.2.1. Input preprocessing

The range data gives us the distance at which there is an obstacle in certain directions. The distance in this program is given in pixels since the map is loaded from a bitmap. Depending on the size of the map, the range data may give us input that have a value of 50, 100 or more. The maximum range is determined by the sensor. By default, our sensors have a maximum range of 50 units. A neural network works better if the input is between -1 and 1 so we normalize the input to match neural network requirements as explained in Chapter 3.

4.2.2. The Input

The number of input (I) depends on how many sensors around the robot we are using. As explained in the previous paragraph, we are going to have a sensor every 10 degrees which gives us 36 inputs for the neural network. This gives us enough precision on the environment without having too many inputs in the neural network. Too many input neurons would slow down the computation done by the neural network that must be done in real time. In this project, it would drop the frame rate; on a real robot, it would lower the input rate and therefore the reaction time of the robot.

Each input keeps track of the history of input for a certain number of times (H). For the landmark detection neural network, we used 6 history of inputs. We determined experimentally that 6 history of inputs gave enough history information of approaching an landmark to the neural network while keeping a good frame rate in the simulation. If this landmark detection was used on a real robot, the number of history of inputs should be determined by the hardware used so that the computer still has time to execute the landmark detection algorithm in between each range data query.

So the actual neural network has $I \cdot H$ number of input neurons. So we have the input of each neuron at time t , the input of each neuron at time $t-1$ and so on until the input of each neuron at time $t-H$.

In order to illustrate the architecture of the neural network, we give the example (Figure 12) which has 3 sensors with 3 steps of history in memory which gives us 9 inputs to the neural network. It is important to note that there are no weights associated with the links that creates the history of input. When we are entering a new input to the neural network in our example, every $t-2$ values are discarded, $t-1$ values are copied onto $t-2$ values, t values are copied onto $t-1$ values and the new entry is stored in t values. The actual neural network inputs are the input with memory layer.

The number of inputs grows really fast each time we decide to have more history or more sensors and we have to decide carefully how much history and sensory data is really needed.

With this history of inputs or memory of inputs, the neural network will make a decision based on where it is now and where it was the passed couple of steps. Using this technique, the neural network should recognize the “approach” towards a landmark and drop a landmark only when it is closer to the center of the intersection.

4.2.3. The hidden layers

There are two hidden layers. The first hidden layer (Time dependent layer) has an unconventional connection to the input layer. This layer has (H) group of neurons with (T) neurons per group. Every neuron of the first group are connected to all neuron of the input layer that corresponds to time t , every neuron of the second group are connected to all neurons of the input layer that corresponds to time $t-1$, etc...

The network at this layer can take decision based on a time independent input. At this level, we are allowing the neural network to make computation at each time step without have other time step interfere with the computation. Each group of neuron makes a decision based on the information of the range data at one point in time. The second hidden layer (Normal layer) is just a normal neural network layer where every neuron of this layer is connected to every neuron of the previous layer. At this level, the network blends the different history inputs. The previous layer made decisions based on the range

data at each time step; we now need to take those decisions which are related to the range data of different time step (t, t-1, t-2) and take into account all the time step.

There are 2 main motivations in such architecture for the hidden layers. First of all, by grouping the neurons depending on time, we are forcing the neural network to “compress” the input at each time step before making a decision. It seems a reasonable architecture since we want the neural network to take into account the environment at each time step. Second of all, by not connecting every neuron of the first hidden layer to the input layer, we are reducing the number of computation that has to be done by the neural network and therefore making it a little faster.

4.2.4. The output

The point of this neural network is to know whether or not we should add a landmark at a specific location based on (H) history input. So the output is a single neuron that gives us a true or false output. The output is actually a floating point, so if the output is above 0.5, the output is true; if it is below 0.5, the output is false.

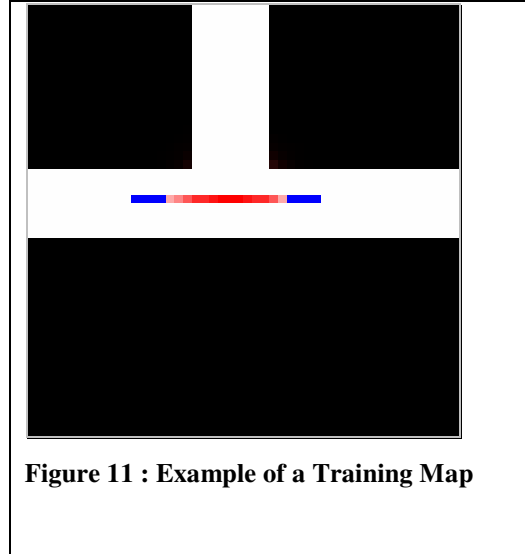
4.2.5. The training

We are using a supervised learning algorithm to train the neural network. The error correction is the one of a back propagation neural network. After some experimentation with the learning rate, a learning rate of 0.02 gave a stable learning process. For better flexibility, noise is added to the input to the neural network. The architecture of the neural network is as follows: 36 inputs corresponding to the 36 distance sensors data with 6 input history for a total of 216 actual input to the neural network, 12 neurons per group with 6 groups (1 per input history) for a total of 72 neurons at the first hidden layer, 24 neurons for the second hidden layer and 1 output. We designed different map with different intersection and make the robot walk back and forth on those map following a path. On each step on the path, we have a target value that is used to train the neural network.

The map below (Figure 11) is an example of maps we use to train the neural network. Obstacles (walls) are represented in black. The white area is where the robot can travel. The robot we are training will follow the blue and red line. The color code tells the network what is the desired response (target value). Blue (RGB = 0,0,255) means no landmark; it trains the network to respond 0 and the darker the red the more there should be a landmark up to Red (RGB = 255,0,0) which trains the network to respond 1.

As the robot walks along the line, the neural network receives information from the sensors and returns a value (TRUE / FALSE), and the color coded line is used as target value to train the network using back propagation. For this example (Figure 11) we have another training map in APPENDIX A, where the line is vertical

and stops at the intersection so that the neural network would also recognize this intersection approaching from the top. This method to train the neural net is very flexible, we can easily modify the desired target value or the kind of intersection the neural net will encounter. If the neural network does not handle a certain kind of intersection very well, we can very easily add a map with this kind of intersection to the list of training maps and retrain a new neural network.



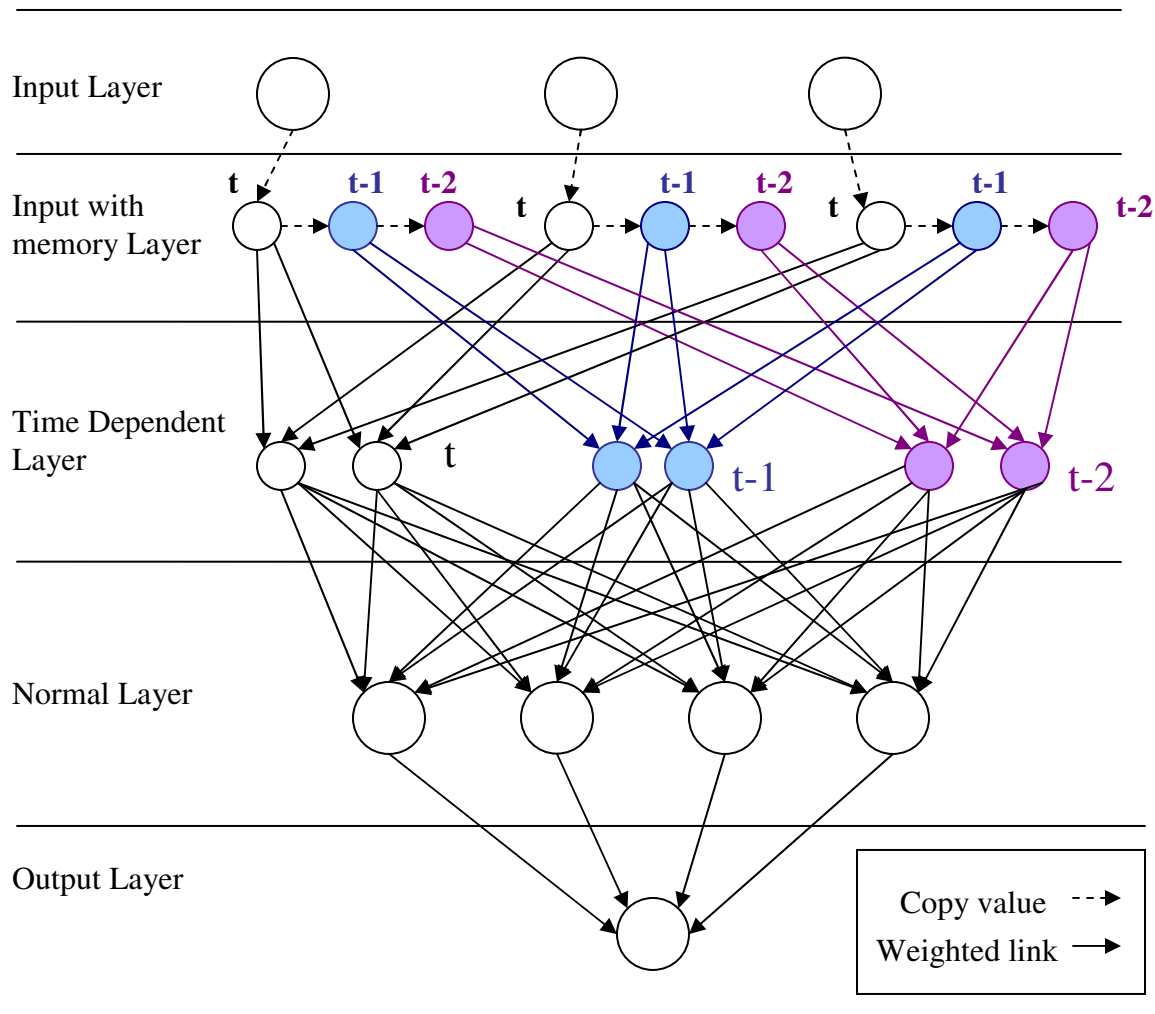


Figure 12 : Neural network Architecture 3-3-2-4-1

Using this architecture, we allow the neural network to sort information at each time step then make a final decision based on a history of input which allows it to recognize the fact that it is “approaching” a landmark and drop a landmark only where needed and not before or after. Since we are training the neural network online approaching each intersection from any side, it will detect an intersection no matter where it is coming from. The neural network has no knowledge of the position of each intersection. It is designed to recognize intersections that are similar to what it was trained on. Also, a neural network is much more flexible to noise that would occur in a real robot as we will show in the results section.

CHAPTER 5

Topological Map

Every landmark found by the algorithm described in the previous chapter is to be added to the topological map if necessary. In an environment such as Figure 13, we want the topological map to resemble Figure 14 where the T-intersection and every room entrance and room is represented by a node.

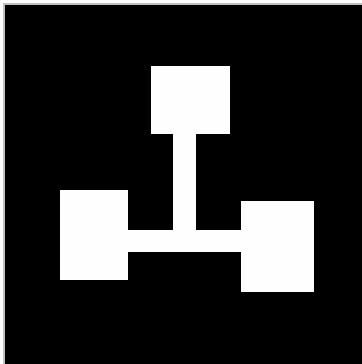


Figure 13 : Demo Map 1

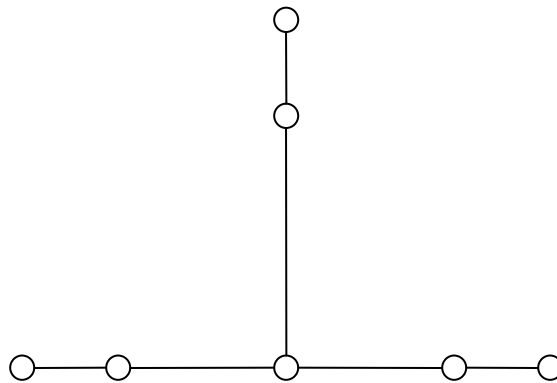


Figure 14 : Ideal Topological Map for Demo Map 1

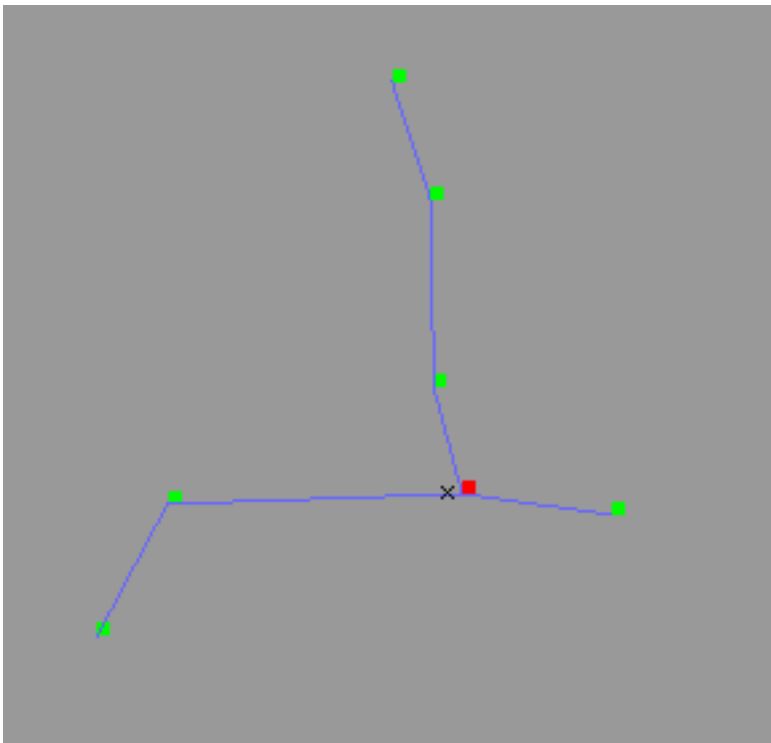


Figure 15 : Actual Topological Map by the Robot for Map Demo 1

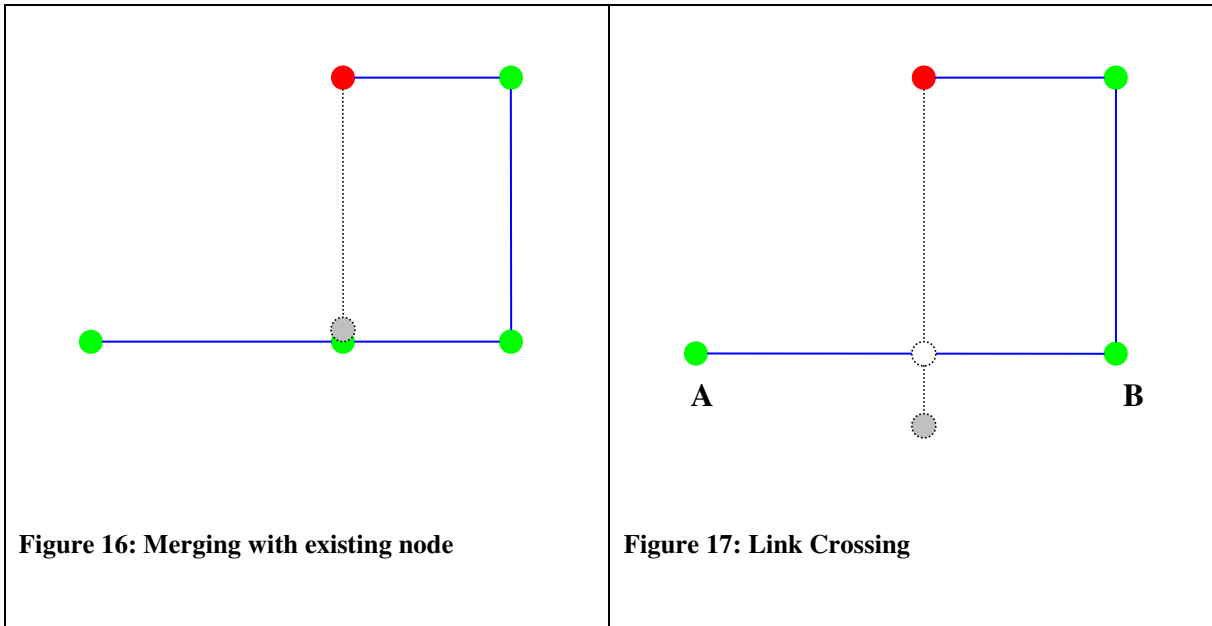
The topological map created by the robot (Figure 15) has most of the features we are looking for. The green points are nodes, the red point is the closest node of the current location of the robot, the x show the current location of the robot and the blue lines represents the links between the nodes which are obstacle free paths from one landmark to the next.

Node creation

The new node can be added in an unexplored area where it will have only one link connecting it to the previous node, but it can also be added in between two nodes. The landmark detection algorithm may miss an intersection going one way and find it coming back. Therefore, it is necessary to consider the option of a node added in between two existing nodes.

Allowing the creation of loops within the topological map is important for more complex environment. A loop could exist within a room going around objects or in a building that has a hallway that loops around. At every node creation, we check the topological map

recursively up to a limited depth. The depth limit is a variable that the user can set. We are checking for two possibilities. Either the node we want to add is within the minimum distance of some other node within the topological map in which case, we simply link the previous node (P) to that node we found to be close to the robot as shown on Figure 16, or the link connecting the current position (C) to the previous node (P) is crossing some other link (linking node A and B) in the topological map in which case, we create a node that is linked to node A, B and P and then create a node at the current position (C) as displayed on Figure 17. It is important to note that we do not need absolute coordinates in order to check for loops, we use relative coordinates from the robots current position. With the estimated distance traveled gives by the robot's odometer and the recorded distance and direction of each link, we can find loops.



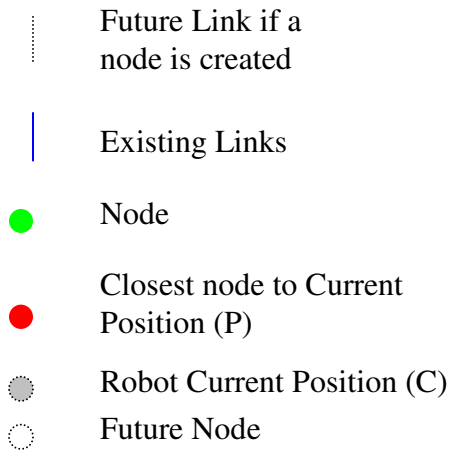


Figure 18: Legend

Since each intersection should be represented by a unique landmark, each node has a minimum distance within which no other node should exist. This minimum distance doesn't have to be the same for every node since it depends on how much obstacle free space surrounds the node. It is computed from the range data. We find the smallest value $\text{RangeData}[i] + \text{RangeData}[i+180\text{degree}]$ and multiply this value by a user defined parameter r to compute the minimum distance value. The parameter r allows the user to have more flexibility on how many nodes will be created in the topological map. The lower r is, the more nodes will be created since they will be allowed to be created closer from one another. We determined experimentally that setting r to 0.9 gives good results. It means that each node will cover 90% of the largest obstacle free radius around the node. This value works for all tested environment and would be recommended if used on a real robot.

Because of this method, each landmark found doesn't necessarily translate into a new node in the topological map. There must not be any node within the minimum distance of the node we are adding and the new node must not be within the minimum distance of another node close by. The landmark detection algorithm may find multiple landmarks within an intersection. This will be translated into a unique node in the topological map.

Each time a node is created; the robot copies the values of its current range data and saves them in the current node. For node created at intersections, the robot needs to travel to this intersection in order to save the current range data. When the robot comes back

at the position of this node, it can check if the range data stored is similar to what it can see now. We will describe how this comparison is done and how the robot relative position is updated in the next paragraph.

Position Tracking & Localization

At all times, the robot knows which node in the topological map is the closest to its current position. We will refer to this node as the “current node” from now on. Any newly created node is set as the current node, but as the robot travels in a part of the environment that has already been mapped, it is keeping track and updating the current node for localization. In addition to the current node, the robot keeps track of an estimation of the distance in x and y to the current node.

But because the distance to current node is estimated based on odometry which is subject to errors due to drift and slippage, the robots belief of the location of the current node can be erroneous and therefore the position of every node in the topological map becomes erroneous since the position of each node is relative to the position of neighboring nodes. Figure 19 illustrates this problem: the actual topological map is displayed in grey and the erroneous topological map is displayed in color. The position of the current node is the red point. As we can see, because of the error due to the drift & slippage, the topological map doesn't represent the actual location of landmarks. The node representing the T-intersection is within an obstacle.

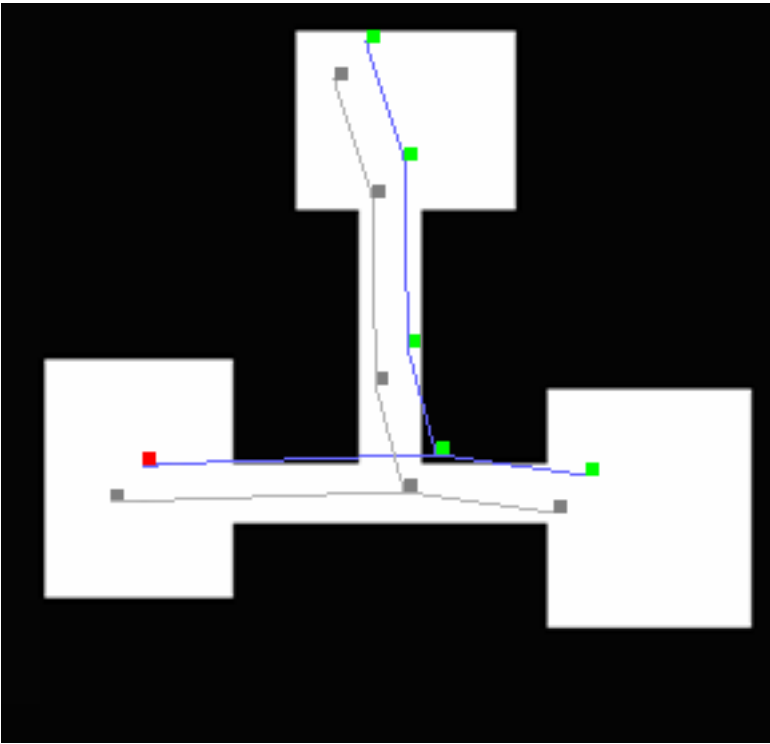


Figure 19: Erroneous position of nodes due to drift and slippage

Because of this problem, the robot needs to perform another type of localization every time it believes that it is over the current node.

This localization consists of correcting the distance to the current node by comparing the current range data gathered by the robot (Robot Range Data) and the current node's saved copy of the range data (Local Range Data)

Because the range information is taken at certain angle and not continuous, the robot will rotate so that one of its range data value points towards the north (orientation 0).

The valid directions for comparison are $\frac{360}{RangeDataSize}$.

The comparison is also done considering the current orientation of the robot. If the range information is taken every 10 degrees and the robot is pointing toward 90 degrees (East), we will compare the value of the Local Range Data of the 0 degree (North) to the value of the Robot Range Data of the degree 270 which corresponds to the North.

If the average difference between each values of the Robot Range Data and its corresponding value of the Local Range Data is larger than a threshold, the robot tries to

correct the error by modifying the its relative position to the closest node. We compute the error as shown in the following equation:

$$Error = \frac{\sum_i |RobotRangeData[i] - LocalRangeData[i]|}{RangeDataSize}$$

The threshold is .5 by default. This value was found to be efficient by experimentation. If the threshold is too small, the robot would systematically relocalize even if it is not necessary. If the threshold is too large, the robot would have a large error before trying to correct its position and it may be too far from the real location of the node to correct its error.

The robot computes the distance to the current node using the odometer and compass. The comparison of range data explained above is done when the distance to current node is under 0.01. Because the distance to the current node will unlikely ever be null, we chose a small distance where we can consider that the distance is so small that the difference doesn't affect the result of the comparison. If using this algorithm on a real robot, this value will depend on the precision of motion of the robot and the precision of the odometer.

If the robot believes it is at the same location as the node and the Robot Range Data doesn't match the Local Range Data, it will modify this distance from current node and go to the new location of that current node. We will now explain the algorithm that modifies the distance from current when it is known to be inaccurate.

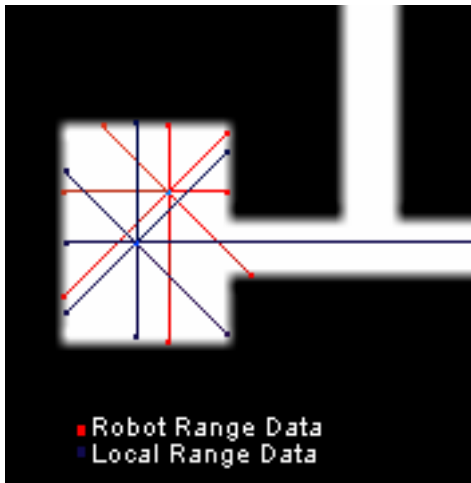


Figure 20: Robot Range Data at wrong location & Local Range Data

In Figure 20, the robot and its range data values are displayed in red while the Local Range Data taken at the time of the node creation are in blue. The robot distance to node is null, but the comparison of the range data shows that it is not at the right location. The robot must modify its relative position to the current node (blue).

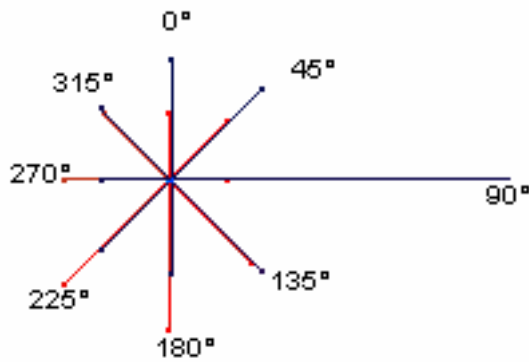


Figure 21: Range Data Comparison

The algorithm chooses for each direction if the relative position to the current node should be to the right or to the left and if it should be higher or lower. We then compute ydifference as:

$$y\text{difference} = (RobotRangeData[i] - LocalRangeData[i]) \cdot \cos(iDirection)$$

As shown in Figure 22, if *ydifference* is between .5 and -0.5 (same value as the threshold when comparing the two range data); the distances are similar enough and no modification in the relative position of the robot should be done. Otherwise, we set *ydirection* to -1 or 1.

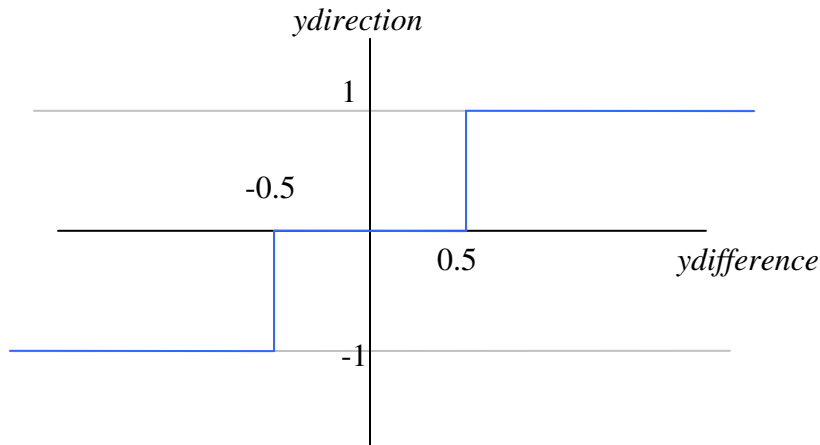


Figure 22: Direction decision function.

This is done for every direction. In the example of Figure 21, we would have 8 different *ydirection* which are averaged. The resulting value *ydirection_average* is returned and used to reposition the relative y coordinate of the robot to the current node.

This algorithm is also applied to find the relative x coordinate of the robot to the current node using this equation to find *xdifference*:

$$xdifference = (RobotRangeData[i] - LocalRangeData[i]) \cdot \sin(iDirection)$$

Basically, for every direction, we compute how much the robot seems to be off target and in which direction and “vote” if the robot should go to the right or left on the x axis and if it should go up or down in the y axis. In Figure 20, the range data from angle 90 would tell the robot to go too much to the left if the “votes” were weighted by the magnitude of the *xdifference*. This “voting” system insures that the modification of the relative coordinate of the robot will be in the right direction.

List of direction towards potential paths

Each node in the topological map has a list of directions where an obstacle free path towards another room potentially exists. This list of paths is later used by the navigation algorithm to discover unexplored areas. Once the robot explored a path in the list, a flag associated with the path is set to keep track of explored paths. Since the robot has a on board compass, the directions of paths will be based on the compass and therefore absolute.

In order to create the list of directions, the robot parses the range data array to find any distances that are larger than a threshold t . Each consecutive distance that is larger than that threshold will be listed as an unexplored path.

t is computed as follows:

$$t = \frac{\max}{k}$$

where \max is the largest value in the range data and k is a parameter set by the user.

The larger k is, the more sensitive the path detection becomes. We defined by experimentation that using $k = 2$ is an efficient value for detecting paths. When $k = 2$, any directions where the range data value is larger than half the maximum value of the range data will be add to the list of paths.

We are illustrating the process of creating the list with an example. In Figure 23, we are representing the range data array (RD) on top of the environment. The largest distance in the array is RD[0]. RD[0], RD[8] – RD[9] and RD[26] – RD[27] will be listed as unexplored paths. If the robot is in a closed room where every value in the range data is larger than variable “ \max ”, no direction will be listed as unexplored paths.

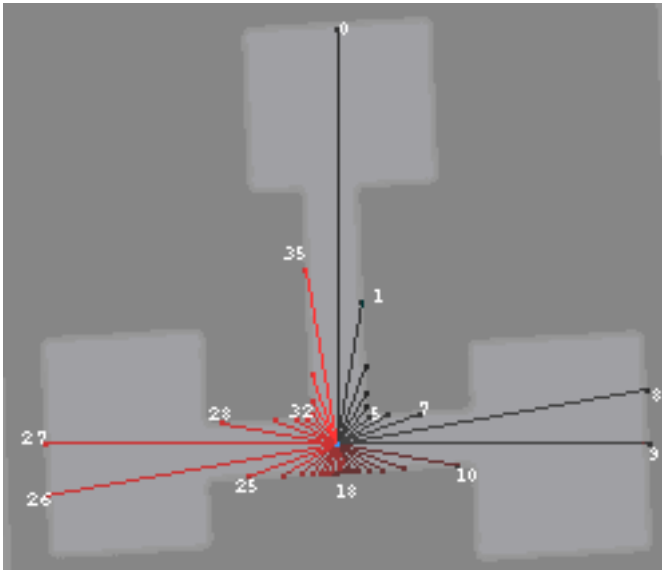


Figure 23: Range Data representation on simple map

Each links in the topological have the estimated length between the two nodes it is linking and the direction. The relative position of each node is therefore known but there is no absolute position of nodes or links.

CHAPTER 6

Navigation for Topological Map Building

In order to have the robot generating the topological map without human intervention, we need to have an algorithm for navigation. There is two level of abstraction for navigation. The robot first estimates the approximate desired direction (d) in which it needs to go. Desired direction (d), updated every time a landmark is found, is either an unexplored path taken from the current node or the direction of an existing node that leads to a node that still has unexplored paths. In Figure 24, the red point represent the robots current location, the green point represents explored nodes without unexplored paths and the blue point represents an explored node with still unexplored paths. The robot cannot go directly towards the blue point since there could be obstacles. It needs to go through the T-intersection in order to reach the unexplored path and finish the topological map. The desired direction (d) in this case would point towards the T-intersection.

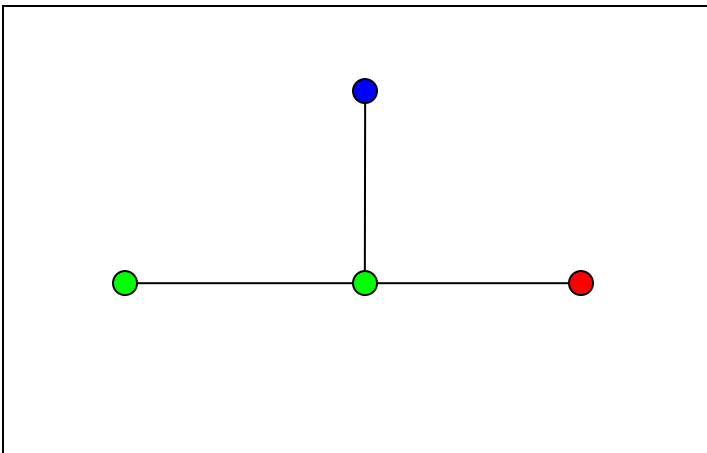


Figure 24: Simple topological map

For the second level of abstraction, we developed an algorithm that allows us to merge two variables: following the desired direction (d) set by the first level of abstraction while avoiding immediate collision with potential obstacle. Standard obstacle avoidance doesn't easily let us merge those two variables.

We modify the range data by weighting each of its values with a bell curve so that the maximum weight is applied in the desired direction (d). We then make the robot turn towards the angle corresponding to the largest value of this weighted range data. By distorting the range data, we emphasize the desired direction while keeping detailed information given by the range data and therefore merging the desired direction and obstacle avoidance.

Every cell of the range data is multiplied by a y value computed using the following bell curve equation.

$$y = \frac{1}{\sigma\sqrt{2\cdot\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

With

$$\sigma = \frac{1}{RangeData[0] + 0.05}$$

$$\mu = 2$$

$$x = \frac{angle - d + 180}{360} \cdot 4$$

σ let us dynamically set the strength of the weight depending on the distance of the obstacle in front of the robot $RangeData[0]$. We add a small value to $RangeData[0]$ to avoid division by 0 when the robot is touching a wall which would result in $RangeData[0]$ being null. μ and x allows us to map the desired direction (d) to the strongest weight of the bell curve. $angle$ is the reading from the compass of the robot.

We arbitrarily chose a bell curve that ranges between 0 and 4 and we map each value of the range data to a value on the bell curve so that the value in the range data pointing towards the desired direction (d) matches the highest value of the bell curve which is at $x=2$ as show in Figure 25. We set μ to 2 because we chose to have the largest value of the bell curve at 2.

The weight put on the importance of the desired direction depends on the distance of the obstacle in front of the robot. When an obstacle is close in front, σ is large, and the bell curve is wide. The bell curve is narrow when the obstacle in front is far. When the robot is very close to an obstacle, we cannot put as much emphasis on the desired direction we want the robot to follow; it needs to avoid the obstacle first. When there is nothing in front of the robot, σ is small, the bell curve is narrow, we put the emphasis on the desired direction we want the robot to follow all details of the environment doesn't matter. We then smooth the resulting array by averaging each array cell with its 2 neighboring cells. The robot turns in the direction of the largest distance in that weighted and smoothed range data.

This obstacle avoidance algorithm can easily be used on a real robot since scale doesn't matter. The robot will always choose to turn towards the angle with the largest value in the modified range data.

Figure 25 shows the bell curve we are using with $\sigma = .5$

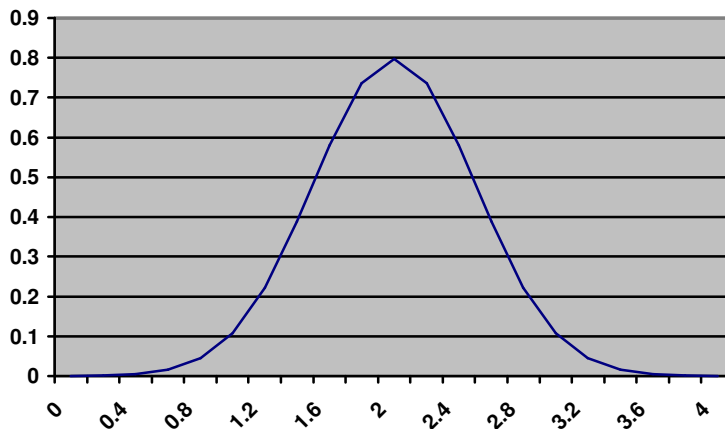


Figure 25: Bell Curve

We illustrate in the following examples how the modified range data reacts to different distances of obstacles in front of the robot. Figure 26 and Figure 27 demonstrate how the bell curve modifies the range data. Figure 26.1 shows the range data when the robot is in front of an obstacle. The robot's position is at the center of the range data represented by

a blue point. Figure 26.2 shows the weighted range data. Although the desired direction is 15 degrees to the right, the largest value of the weighted range data is at 10 degrees towards the left. The robot will turn left in order to avoid the obstacle.

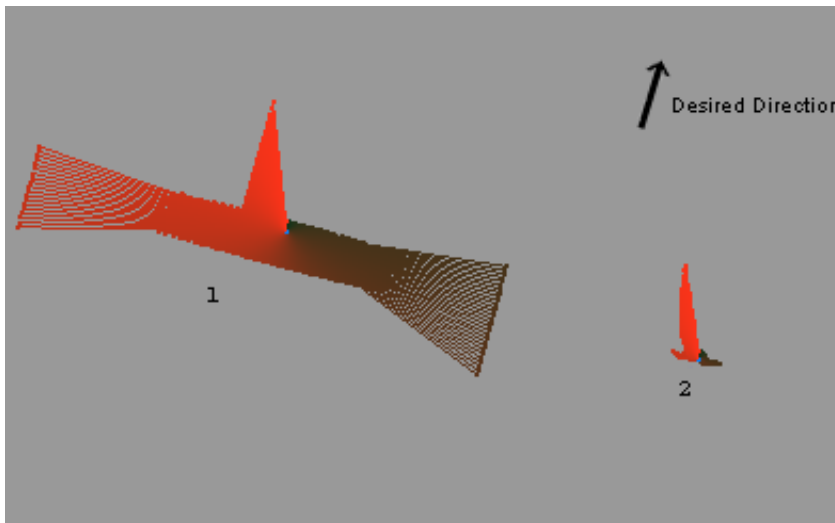


Figure 26 : 1.Range Data and 2. Weighted Range Data with obstacle in front of the robot

Once the obstacle is not in front of the robot, the emphasis is put on the desired direction. Figure 27.1 shows the range data when the robot does not have any obstacle in front of it and Figure 27.2 show the weighted range data in that same situation. As we can see, the largest value in the weighted range data is in front of the robot, and therefore, the latter will go straight in the desired direction.

Modifying the range data for navigation as we do it is very efficient. It balances the information of the range data and the direction in which we want to robot to go depending on the distance of the obstacle in front of the robot. The robot will therefore avoid any obstacle close by and still go towards the wanted direction.

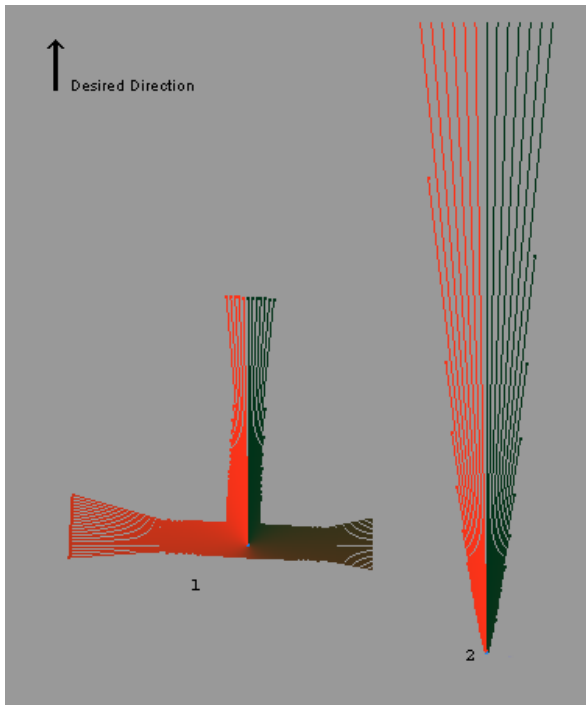


Figure 27 : 1.Range Data and 2. Weighted Range Data with no obstacle in front of the robot

In some cases, when desired direction is off centered compared to where the path really is or when the desired direction points towards a dead end, this navigation control will fail and the robot will be in a local or global minima. This happens when the landmark is detected too early when approaching an intersection or in a room that has only one exit. Those cases will be handled by the function “NextMove” which check if the robot is stuck. This function is explained in the next paragraph.

Navigation process

Figure 63 that can be found in the Appendix is the flowchart of the robot’s update function which is called every frame. Figure 28, Figure 29 and Figure 30 are the pseudo code corresponding to that flowchart.

This function is responsible for calling the landmark detection algorithm, the topological map creation, and controlling the robots motion.

Every time Update is called, it first calls “ParseRangeData”, the function that checks range data and add a node if a landmark is found. As mentioned in the previous chapter, the node is added only if it isn’t too close from the previous node. If it isn’t too close, we then check the node exists at that particular before adding the node. The node may be

created in between 2 existing nodes or as a new node in an unexplored area. When a new node is created, the new node is updated as the current closest node. The robot keeps track of the distance to current closest node. This value would be updated after the function AddNode() if necessary. If no landmark was found, the robot checks if it should update the current closest node, if it does update the current closest node, it also sets the flag Localize telling the robot that it needs to check its position (Position Tracking). The robot may be traveling towards a mapped area and getting closer to another node and the current closest node should be updated.

Before calling NextMove() which is responsible for updating the desired direction and chooses the robot's immediate response to the current situation, we check if the robot is close to current node. The fact that the robot is close to the current closest node triggers the update of the desired direction (d). NextMove function checks if the flag Localize has been set. If the flag is set, the robot will track its position as explained in previous section.

If the robot doesn't need to track its position and it is close to the current landmark, the desired direction must be updated. First it checks if there is a path to be explored at the current node. If there is an unexplored path, the desired direction is set to the direction of this unexplored path. If all paths at the current node have been explored, it checks if there still exist an unexplored path in the entire topological map.

When the robot is not close to the current node and the flag stuck has been set, the robot is forced to go back towards the current node which in turns makes it check if there are other unexplored paths in the topological map. The flag stuck is set to true when the robot range data value in front of it is smaller than 1.5. This value is a parameter that can be changed. By experimentation we realized that the value 1.5 makes the robot detect that it is going towards a dead end soon enough.

```
Update ()
    ParseRangeData ();
    Localize = CloseToCurrentNode ();
    NextMove (Localize);
    return;
```

Figure 28: Pseudo-code of the Robot Update Function

```
ParseRangeData ()
```

```

LandmarkFound = RunNN();
if (LandmarkFound)
    AddNode();
    UpdateDistanceFromCurrentNode();
else
    CheckCurrentPosition();
return;

```

Figure 29: Pseudo-code of the ParseRangeData Function

```

NextMove()
    if (Localize)
        if (OverCurrentNode())
            RangeDataIdentical = CompareRangeData();
            if (RangeDataIdentical)
                goto NormalProcessing;
            else
                ModifyDistanceToCurrentNode();
        else
            MoveSlowlyTowardsCurrentNode();
    return;
NormalProcessing:
if(CloseToCurrentLandmark)
    Direction = CheckForExpansion();
    if(Direction==NULL)
        Direction = CheckForExpansionInTopMap();
        if(Direction==NULL)
            MessageBox("Topological Map Done");
            return;
    g_DesiredDirection = Direction;
    MoveRobot(Direction);
    return;
else
    if(Stuck)
        HandleBeingStuck();
    else
        Move(g_DesiredDirection);
    return;
return;

```

Figure 30:Pseudo-code of the NextMove Function

CHAPTER 7

Experimental Results

7.1. Landmark Detection with neural network

We chose to use 36 inputs with 6 histories per input, 12 neurons per time dependent groups and 24 neurons in the Normal Layer.

We have trained 10 neural networks with 0.02 learning rate on the following training maps shown in appendix A. They cover the main intersection at different scale that we expect the robot to encounter in our environments. That is, cross intersection, T-intersection, 90 degree turns, hallways and rooms. Each map has a path that the robot follows and target values on the path using the color code explained in section “Training Neural Network” (IV, C, f)

We have saved the mean squared error as we were training the neural network in order to see how well the neural network is learning. The 10 neural networks evolve the same way while being trained. The mean squared error drops quickly on the first 400 or 500 iteration and stabilizes around 0.05. The graph below presents those numbers. We only included data after 200 iterations for clarity. The complete data including the error at 50 is included in Appendix A

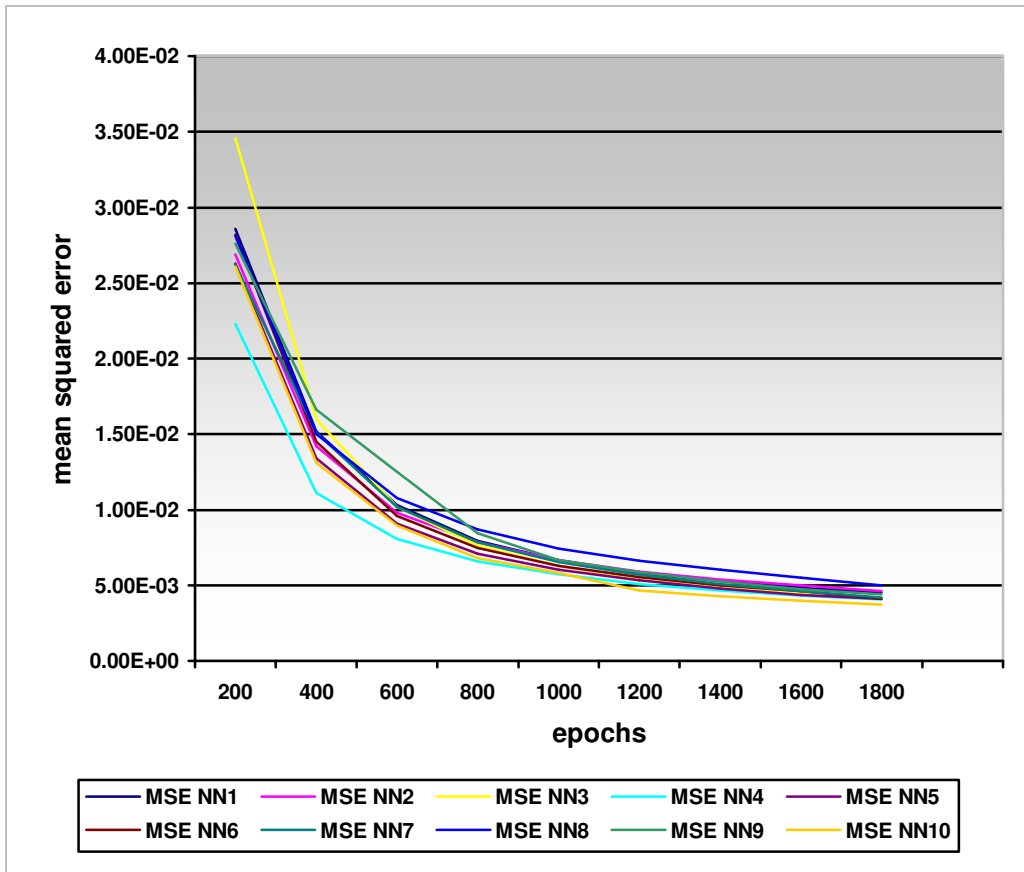


Figure 31: Evolution of the Mean Squared Error over Time during Training

As we can see in Figure 31, the mean squared error stabilizes consistently for all neural networks around 0.005 even neural networks 3 which still had a much higher error after 200 iterations. Considering the fact that those network are learning a man made target, the ending mean squared error is low.

We ran the 10 neural networks on 17 testing map. The maps & detailed results for neural network 3 are shown on the Table 1. We also show the average mean squared error and standard deviation for every map on Figure 32.

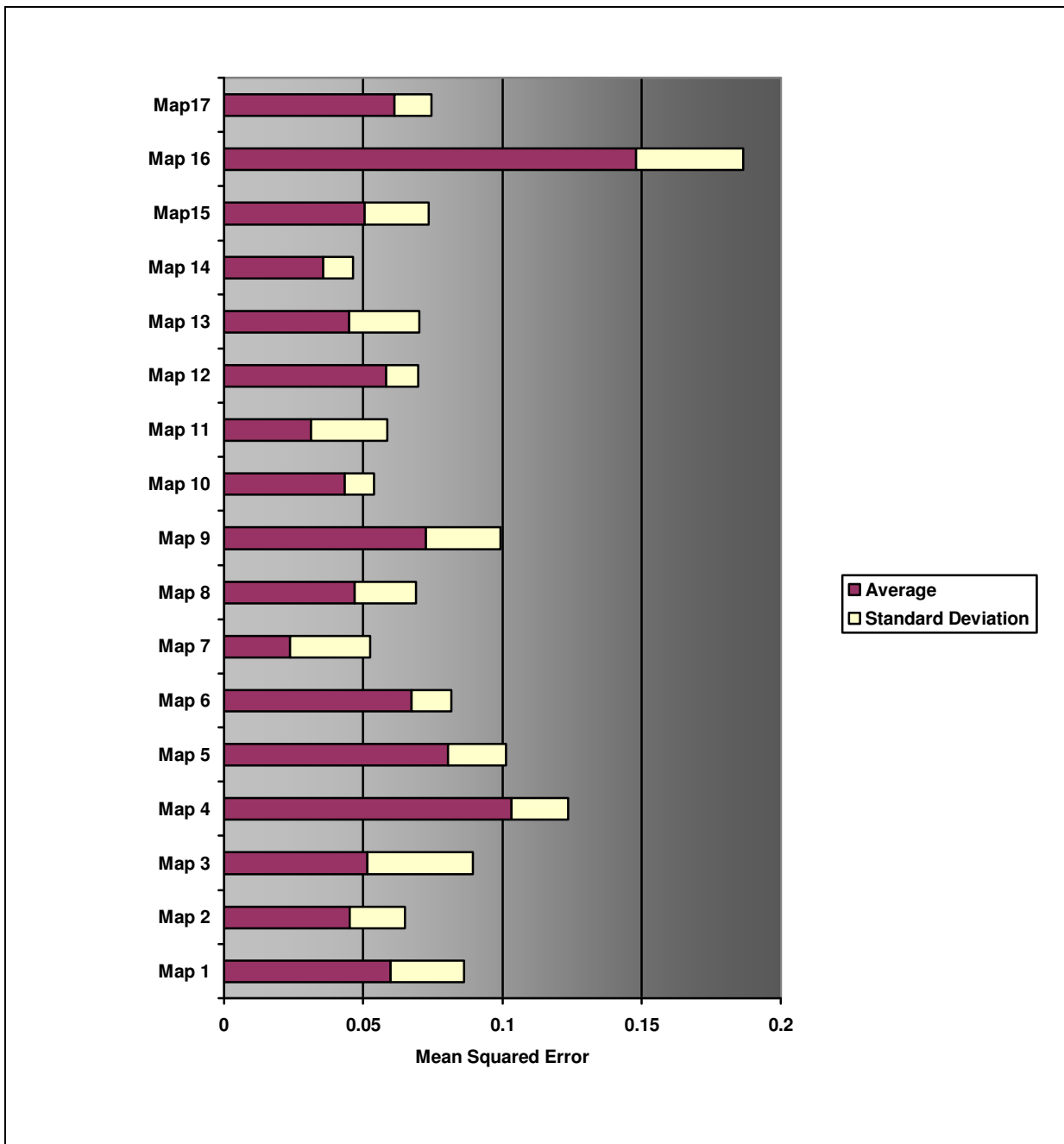


Figure 32: Average Mean Squared Error and Standard Deviation Per Map

We notice on the Figure that all networks have done particularly poorly results on map 16. As shown on the figure below, the neural network 9 finds the landmark offset compared to what we would like to have.

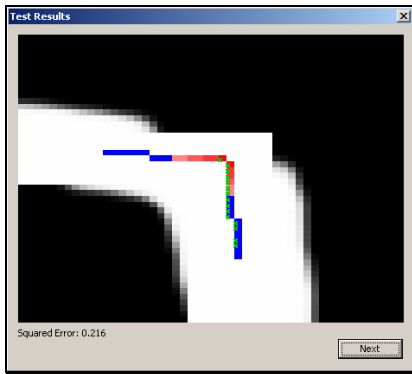


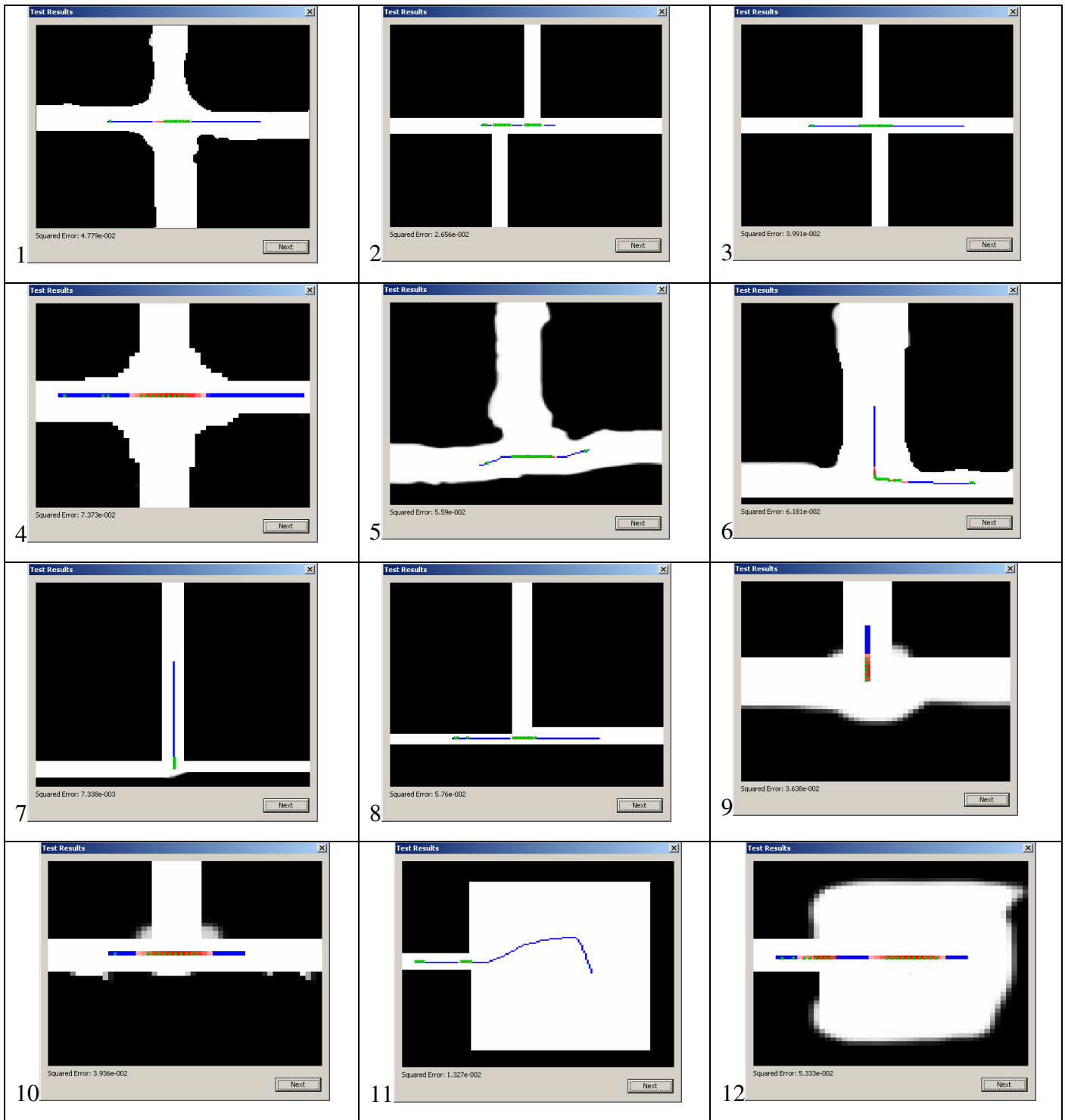
Figure 33 : Neural Network 9 on Map 16

On the other hand, all network do well on map on map 7 which is a map quite close to the basic T-intersection they were trained on.

We are also showing in the next table the detailed results and map for neural network 3. Each map shows where the network detected landmarks as it travels along the blue & red line. The green crosses are the place where the neural network detected a landmark. We are only displaying the detailed result for one neural network because the results are consistent from one neural network to another. There is not much difference on where the neural network places the landmark. We see that the network handles pretty well most maps, even those X intersection where the intersecting hallways aren't perfectly aligned. On several maps, the network detects landmark at the beginning of the path (left side of the line); we notice this behavior on map 1, 2, 3, 4, 8, 12, 13, 14. We can ignore this error due to clear history that only happen on the extremity of the path.

This is most probably due to the cleared history as it starts. As a matter of fact, for each map, we clear the history of the input neurons that we set to 0. The network detects that as a landmark.

The results on map 17 are quite interesting because the network was never trained on a 45 degree turn; it was only trained on 90 turn. Even though, the network is doing quite well on this map. It put landmark on the two turns.



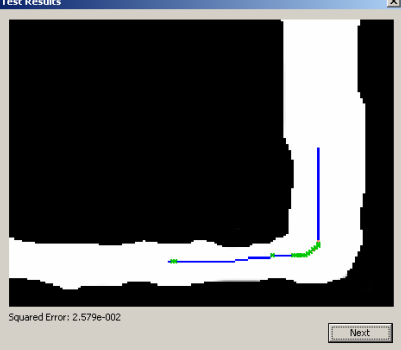
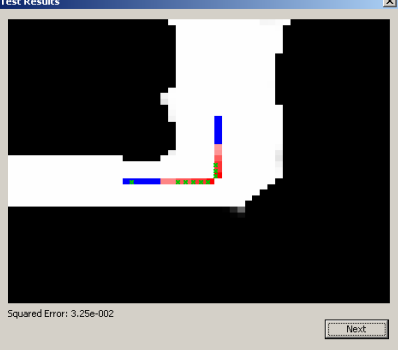
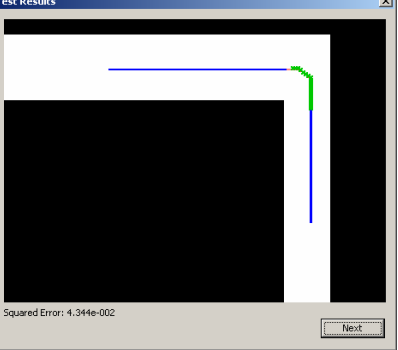
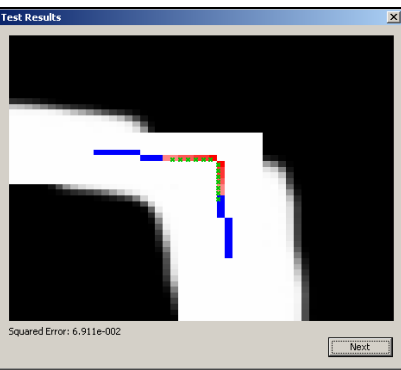
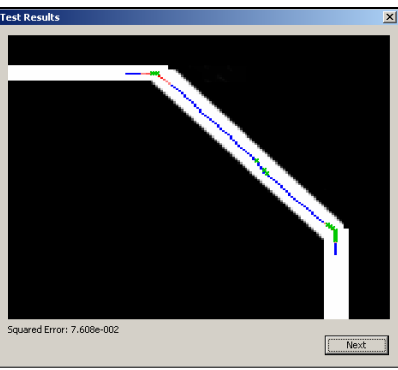
<p>13</p>  <p>Squared Error: 2.579e-002</p>	<p>14</p>  <p>Squared Error: 3.25e-002</p>	<p>15</p>  <p>Squared Error: 4.344e-002</p>
<p>16</p>  <p>Squared Error: 6.911e-002</p>	<p>17</p>  <p>Squared Error: 7.608e-002</p>	

Table 1 : Detailed Results For Neural Network 3

The results in the following table are the mean square error and standard deviation on all maps for each neural network. Neural network 9 has poor results compare to other neural nets while neural network 3 or 4 have much better results.

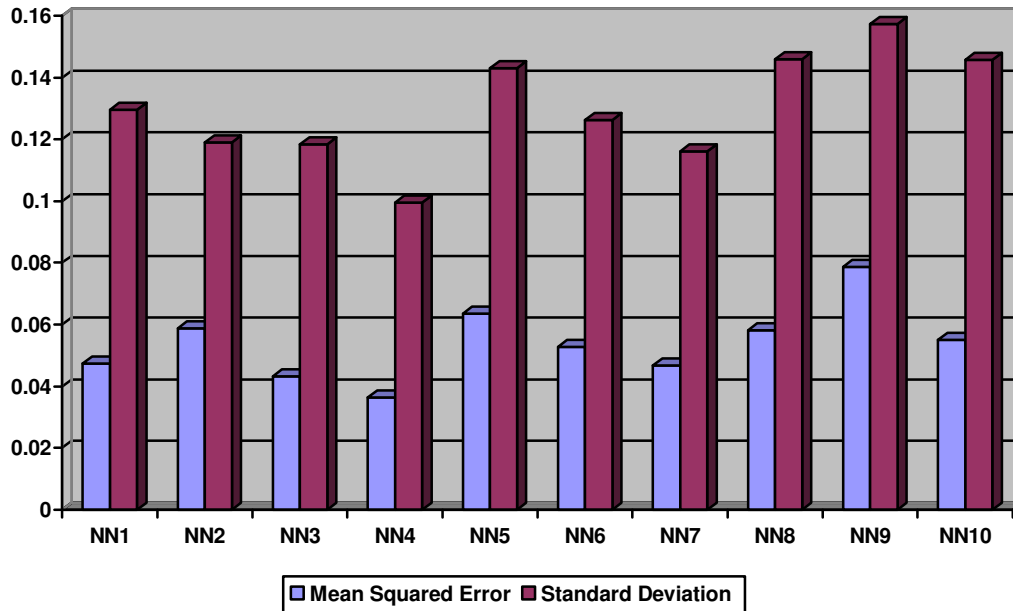


Figure 34: Average Mean Squared Error and Standard Deviation per Neural Network

In order to better compare how the different neural networks react, we ran each neural network on map 3 and map 17 and show the results in the Appendix in Table 4.

We chose map 3 because it is an off-axis cross intersection and map 17 because it is a 45 degree turn on which no neural network was trained. We ran those networks with up to 0.5 of noise on the input data to see how robust they are.

Surprisingly, Neural network 7 and 10 have a better result on map 17 than neural network 3 and 4 which have the best average mean squared error. Landmarks are only found at the 35 degree turn and nowhere else. The other neural network finds the 35 degree turns and a couple of landmark in between the 2 turns. Networks with the poor average mean squared error such as network 9 are off-centered on map 3.

The overall result is satisfactory; all neural networks find intersection consistently.

7.2. Topological Map Creation

All the following results except section 7.2.5. have been created with the robot self navigation algorithm and without human intervention.

The topological map creation depends on multiple parameters. The most important are: Path finder sensitivity (k) on page 40, Close to Current Landmark threshold on page 47, ratio of distance between nodes (r) on page 35, stuck detection sensitivity on page 46. The default values were used for all experiment. Those parameters can be modified to improve the resulting topological maps but the default values were determined experimentally to work well under most circumstances.

7.2.1. Comparing different Landmark detection on different maps

We ran the robot letting it create a topological map of the map shown in figure Figure 35.

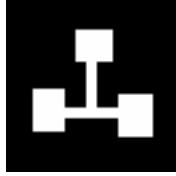


Figure 35: Demo Map 1

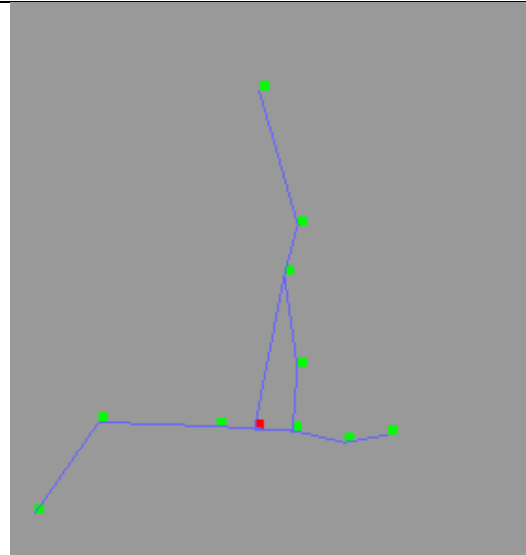


Figure 36: Topological Map of Demo1 using Edge Detection

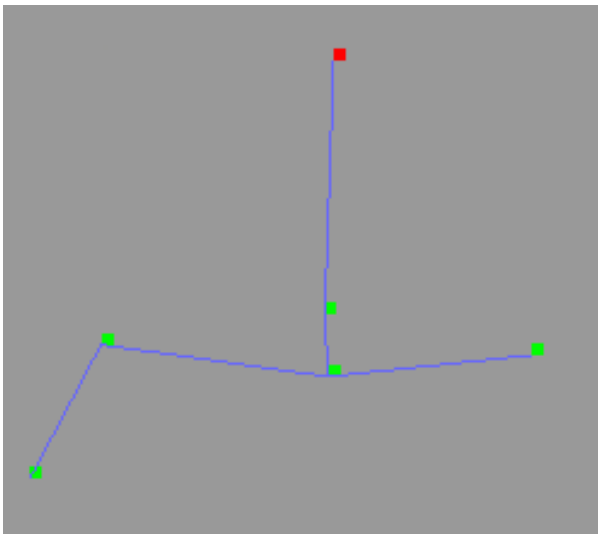


Figure 37: Topological Map of Demo1 with Neural Network 4

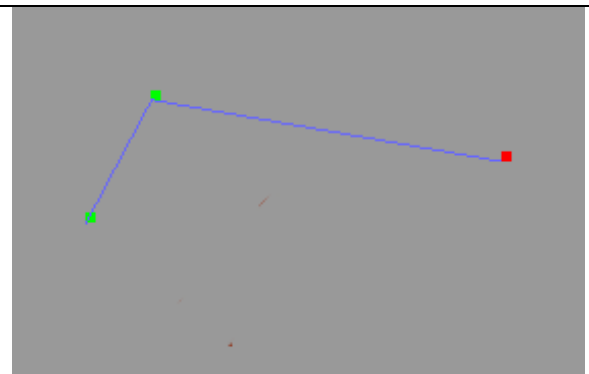


Figure 38: Topological Map of Demo1 with Neural Network 9

We first chose the Edge Detection algorithm to detect landmarks. The result is displayed on Figure 36. We notice that the Edge Detection algorithm is much too sensible, it detects too many landmarks. Figure 37 and Figure 38 show the topological map created using two different neural networks. Figure 37 was created using the neural network 4 which has the lowest mean squared error (Figure 34) while Figure 38 was created using the neural network 9 which has the highest mean squared error. The neural network 9 never detected the T-Intersection and therefore never completed the topological map correctly.

We also ran the robot with neural network 4 and with the Edge Detection algorithm on a map containing a loop show below:

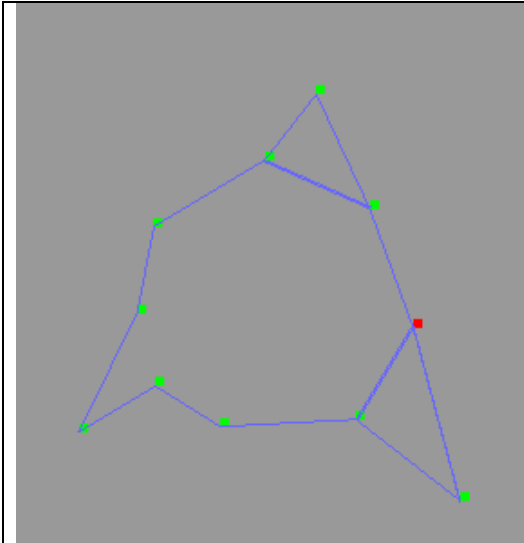


Figure 39: Topological Map of Demo2 with Edge Detection

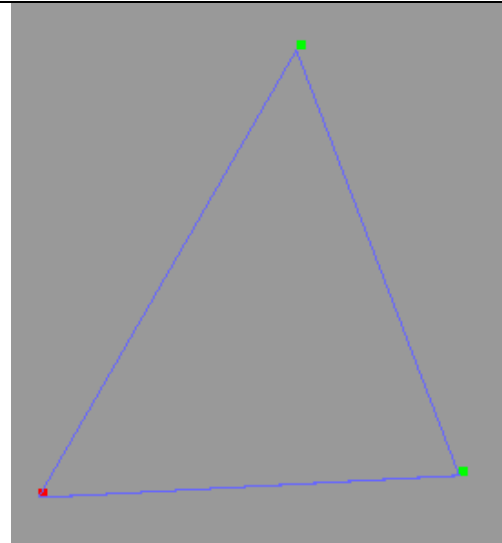


Figure 40: Topological Map of Demo2 with Neural Network 4



Figure 41: Demo Map 2

We notice the same pattern; the Edge Detection algorithm is too sensible and therefore creates too many landmarks while the neural network 4 only finds the rooms as landmarks. We can also show that the topological map creates loops accurately no matter which landmark detection technique is used.

We also ran the robot using neural network 4 to detect landmarks on larger maps and got the following results.



Figure 42: Demo Map 3



Figure 43: Demo Map 4

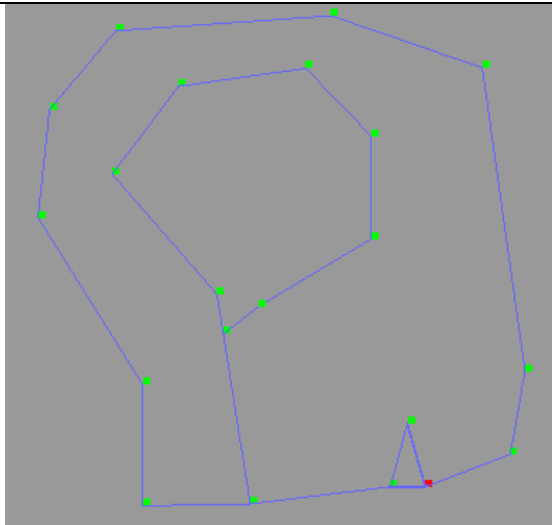


Figure 44: Topological Map of Demo 3

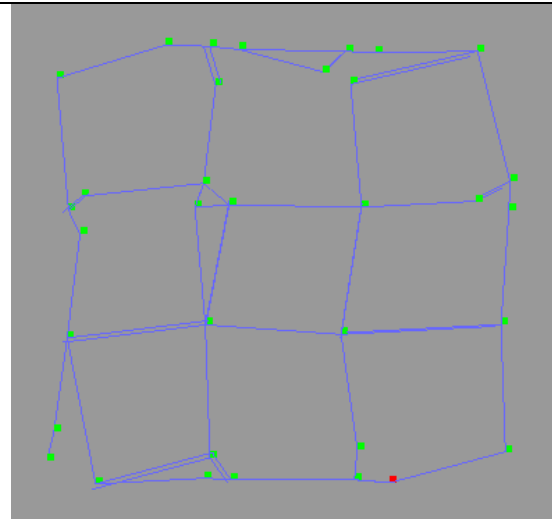


Figure 45: Topological Map of Demo 4

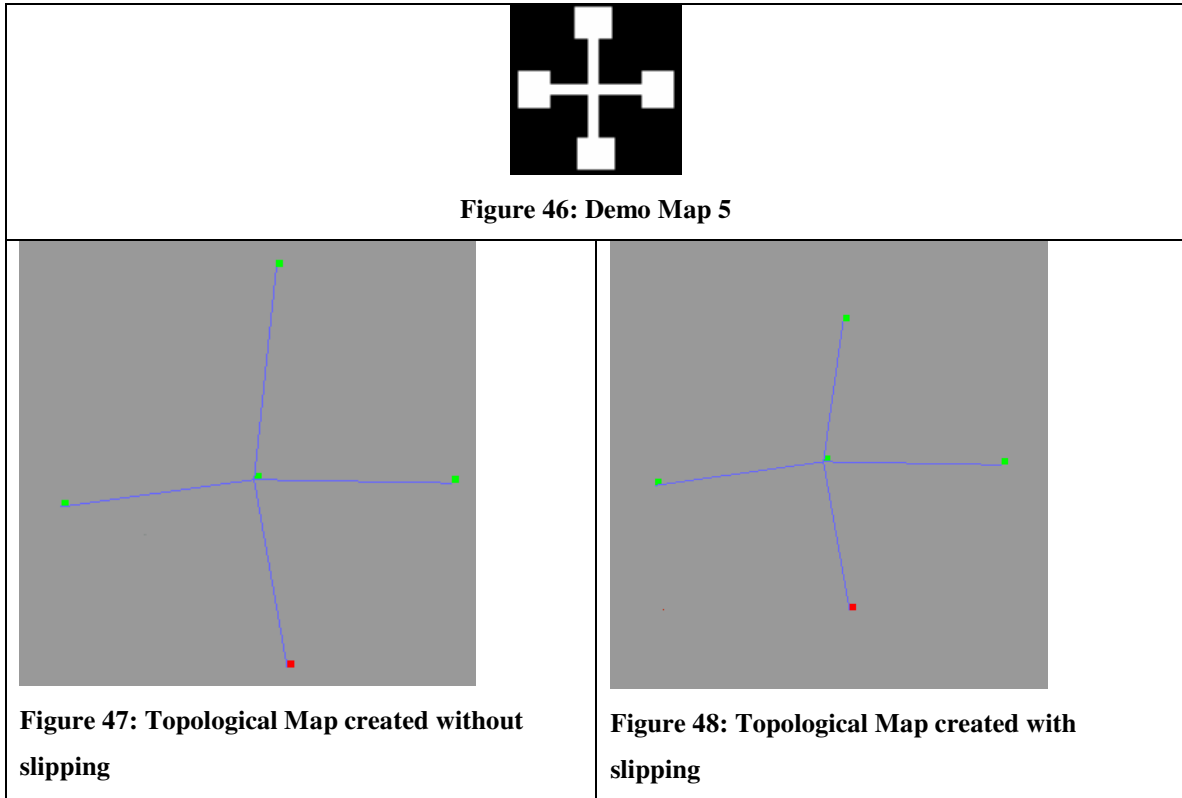
We notice that on Demo Map 3, the robot didn't detect the room on the lower left corner. Although the robot went in the room as it was creating the topological map, the neural network didn't detect a landmark while in the room. The topological map of Demo Map 4 is not as clean as some other maps, it detected multiple landmark for the same intersection on many occasions.

As we have shown on different maps, this approach creates one node per intersection or rooms. It is very similar to what Kuiper & Byun's robot accomplishes except that their approach won't mark rooms as one single node. On the other hand, on large complex map such as Demo Map 4, our approach doesn't guaranty that each intersection will have only one node. An extra step, post process, would be needed to clean up the topological map. Zimmer's approach is not comparable because his landmark doesn't have to

represent an intersection. Zimmer's topological map would have much more nodes similar to our "edge detection algorithm" (Figure 36).

7.2.2. Robot Slipping

We ran the robot on Demo Map 5 with no slipping simulated and with slipping simulated.



There is no significant difference between the two maps although the robot drifted 177 pixels while braking or accelerating on a map where the longest straight line is 61 pixels. Some of this drifting going to a room was canceled out by drifting coming back from a room. But the algorithm check the accuracy of location corrected errors when needed. Large loops may present a problem because the robot will travel a great distance before being able to check localization and therefore the error due to slipping will accumulate. When the robot closes the loop and check localization, the error can sometimes be too large to be corrected.

7.2.3. Adding Noise to Range data

We first ran Neural Network 4 on the testing set with different maximum noise values. The results are displayed below:

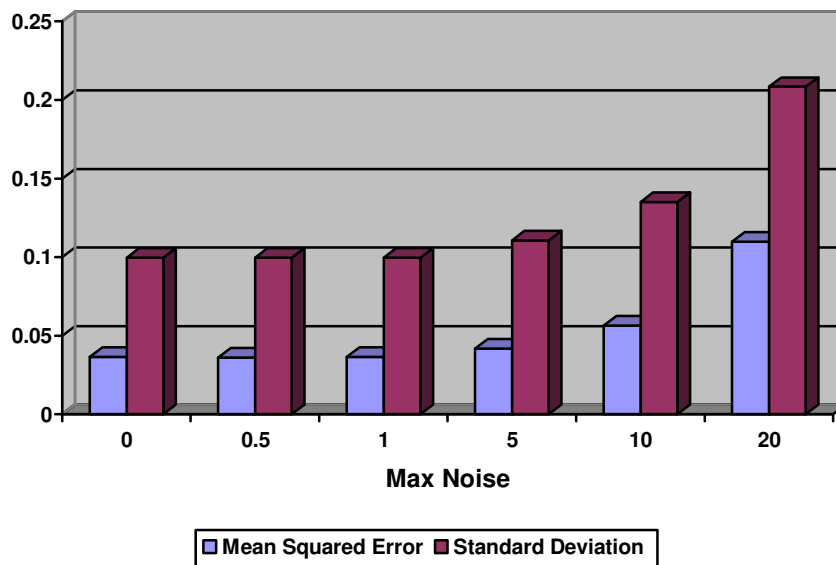


Figure 49: Average Mean Squared Error and Standard Deviation with Different Max Noise Values

On those testing maps, the hallways are between 60 and 120 units long and the width is between 6 and 12 units long. We see that the error starts to be important when the max noise is at 20 units. The error is expected to be important when the maximum amount of noise is larger than the width of the hallways.

Still using Demo Map 5, we ran the robot with neural network 4 for landmark detection with noise added to the range data. We added .5 unit maximum noise in Figure 51, 1 unit in Figure 52 and 5 units in Figure 53. In order to realize the magnitude of this noise, we would like to precise that the rooms of this environment are 12 units by 14 units and the hallways have a width of 4 units.

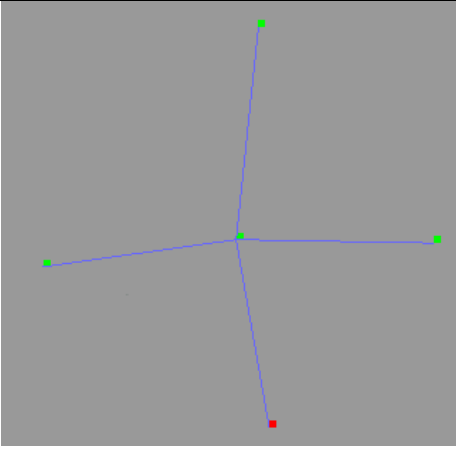


Figure 50: Topological Map created without noise

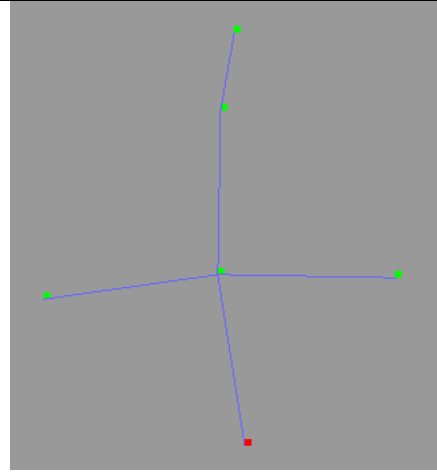


Figure 51: Topological Map created with 0.5 unit maximum noise

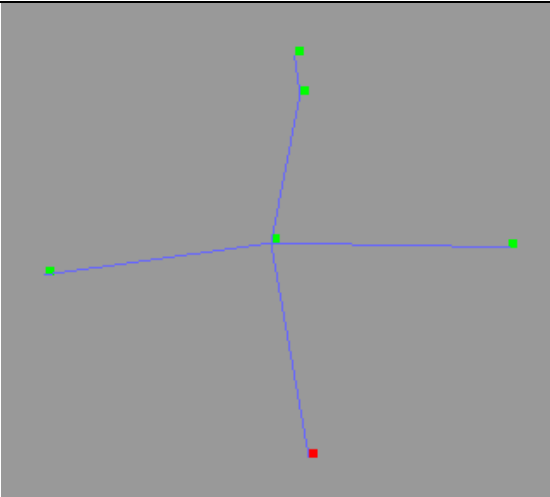


Figure 52: Topological Map created with 1 unit maximum noise

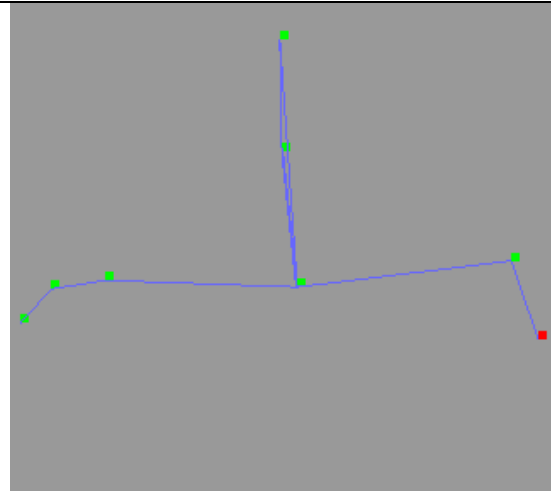


Figure 53: Topological Map created with 5 units maximum noise

The topological map in Figure 53 when the noise is larger than the width of the hallway, the map is incomplete. But with more realistic noise, the landmark creation, map creation and localization was done without error as shown on Figure 51 and Figure 52.

We show here that our approach is just as robust as Kuiper and Byun's approach when adding sensory errors. We are still able to create the topological map as long as the maximum noise is smaller than the width of the hallway.

7.2.4. Sensor Failure

Adding sensor failure is another way to test the robustness of an algorithm. We can think of situations where the robot has one or more sensor failing and the robot is inaccessible for immediate repair. It is therefore important that the robot is still properly function without all sensor working properly.

We first ran Neural Network 4 on the set of testing maps and we compiled the following results:

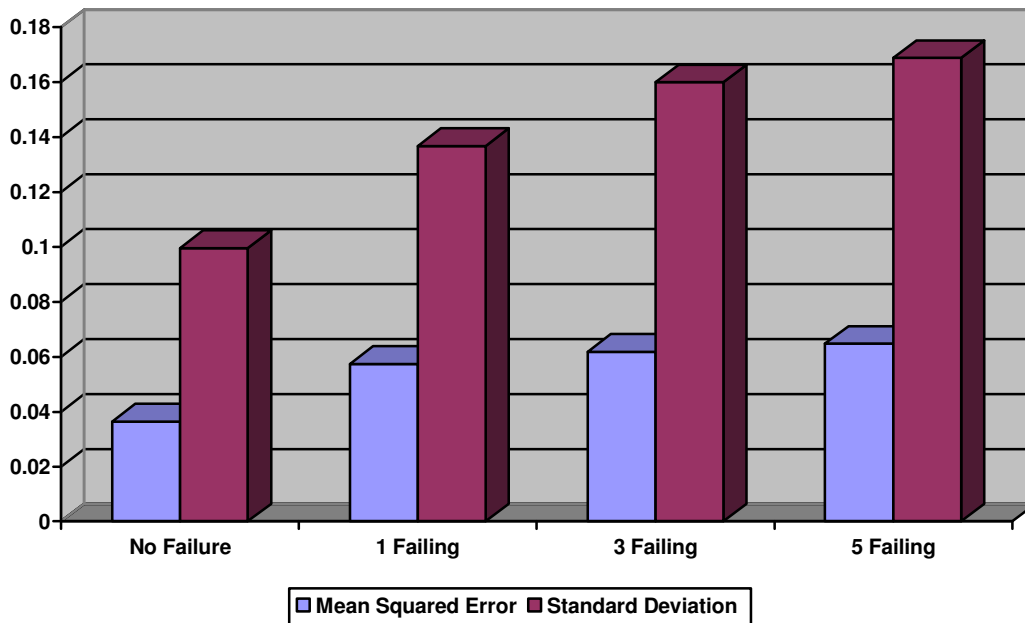


Figure 54: Average Mean Squared Error and Standard Deviation with Failing Sensors

The Neural Network 4 is still working very well even with 1, 3 or even 5 failing sensors out of 36. We see that the neural network is very robust even when we destroy some of its inputs data. But since the autonomous control of the robot also uses the range data we ran the robot with Neural Network 4 on Demo Map 5 with 3 sensor failing chosen randomly. We see in Figure 55.a the robot position with a red dot in the lower room of Demo Map 5. Figure 55.b shows the topological map created and Figure 55.c the range data at current location with 3 sensors returning 0 instead of the distance to obstacle in those directions.

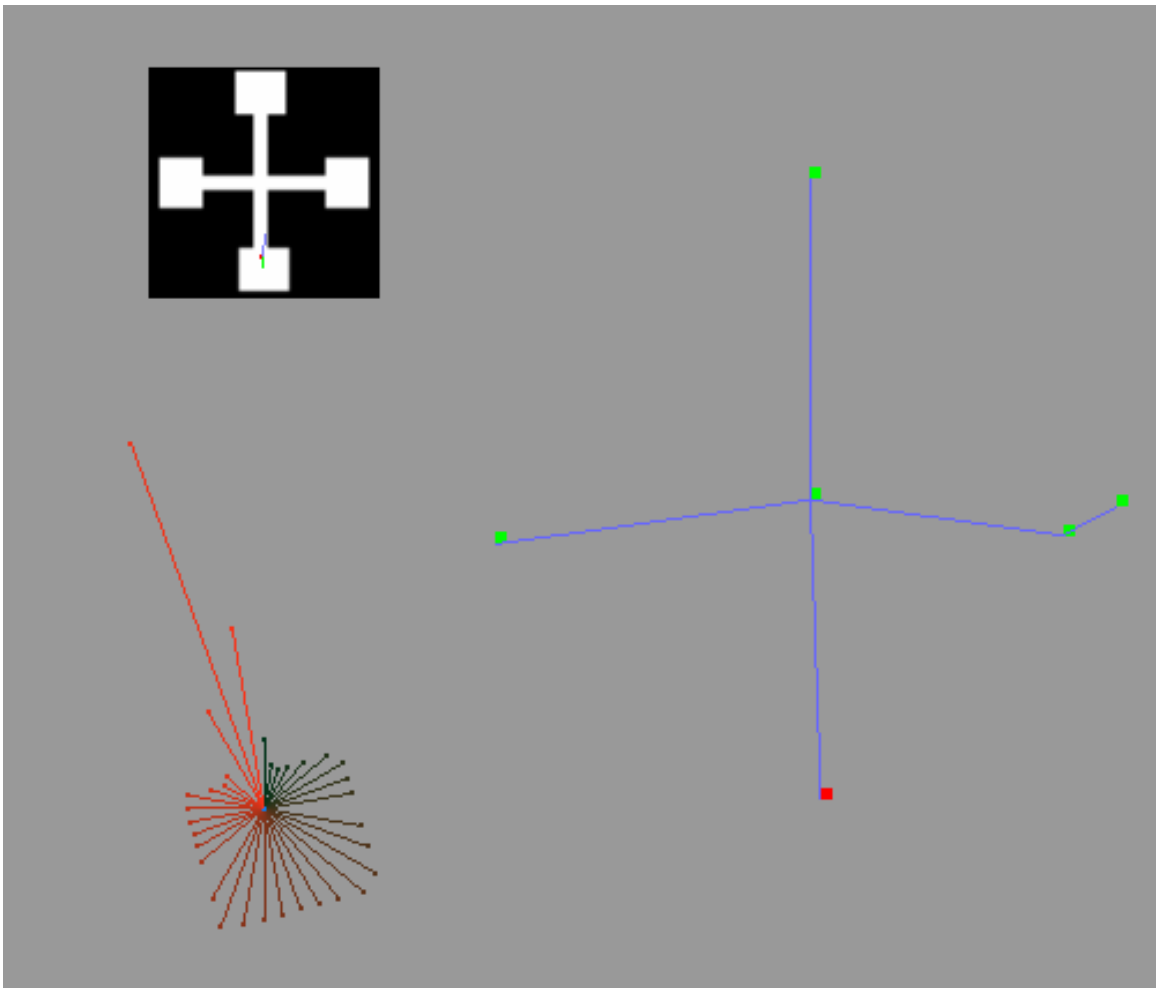


Figure 55: Topological Map with 3 sensor failing

7.2.5. Human robot control vs Auto navigation

For the last experiment, we used a map of part of the 3rd floor of the McConnell building at McGill University shown on Figure 56.

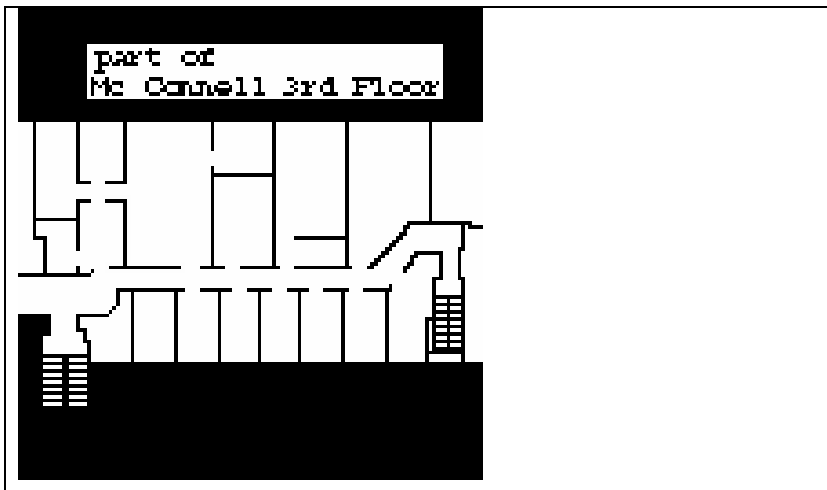


Figure 56: Map of the 3rd floor of McConnell Building

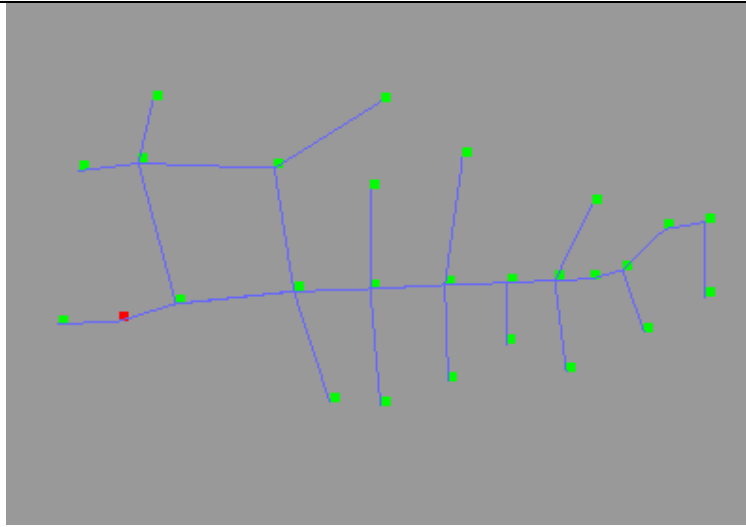


Figure 57: Topological map created with human control of the robot

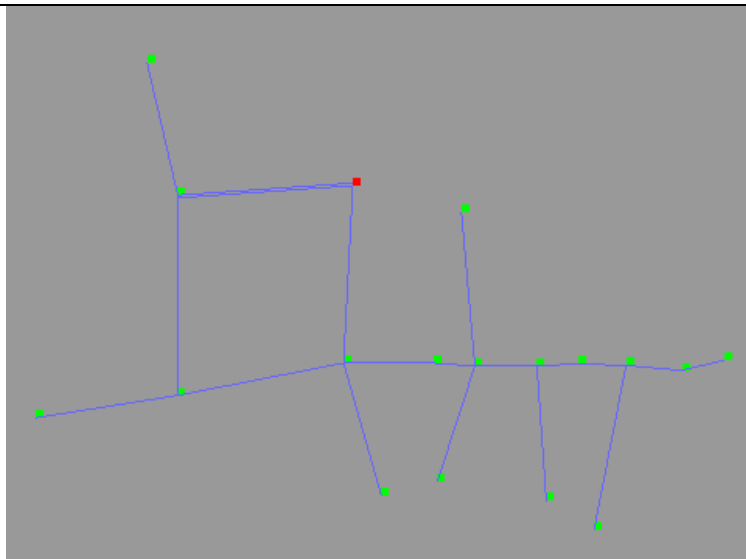


Figure 58: Topological map with robot self navigation

Figure 57 was created by controlling the robot. Although we were controlling the robot, the landmark detection was done by neural network 4 and the topological map creation was done automatically.

On Figure 58, we let the robot navigate to create the same topological map. The robot failed to find some of the rooms. The main reason for that is because of the size of the entrance to the rooms. The robot exploration path detection work, but when the robot actually turned to use the discovered path, it was perfectly aligned with the actual entrance of the room and dismissed that path as unusable. A new training set of the neural

network and an improvement of the navigation algorithm would be needed to correctly map the entire map.

CHAPTER 8

Conclusions and Future Work

This thesis describes a method for creating a topological map online using a neural network for landmark detection. The idea is to have a meaningful relation between the nodes of the topological map and the environment. Using a neural network gives us the flexibility to choose what should be used as landmark. The results illustrate the potential of our concept on a real robot for a topological map creation.

We designed and programmed the simulation for the robot considering what is realistic to expect at all times. Although we showed that this approach works in simulation, some extra work is needed to transfer our approach to a real robot.

Our program simulates multiple lasers to compute the range data. This would be expensive to implement. Using only one laser with a rotating mirror solves the problem of the price but makes the scanning time much higher. Sonars are a low cost alternative to laser range finder but some pre-computation would be needed to cleanup the range data for the neural network. A robot has limited computational power and limited memory. Some optimization of the code for landmark detection and topological map creation should be done if running on a robotic hardware. Still, we believe that transferring our approach to a real robot would be fairly straight forward.

In order to train the neural network for a real robot, an initialization of the network weight based on training in simulation would be appropriate since thousands of iterations are necessary to train the neural network. Further training could be done on the real robot after each run through the real environment by providing locations where a landmark is desired.

An extensive empirical comparison between our approach and other approaches such as Kuiper & Byun [14] should be done next in order to estimate the efficiency of our current

approach. We have discussed the conceptual similarities and differences in the thesis but, running an implementation of their approach on the same set of maps would provide a better comparison. It would be especially important to compare in terms of efficiency. Our simulated robot performs a simple tracking position which is a form of localization. We could extend this to a better tracking position algorithm and initial localization. Initial localization is important when a robot is put back in a known environment at an unknown location. This happens every time a robot is turned off and turned back on. It is not efficient if the robot has to rediscover an environment every time it is turned back on. There exist numerous algorithms for initial localization that could be applied for our purpose.

Some maps that have larger intersections will have multiple landmarks at those intersections; it may be advantageous to merge those multiple landmarks into one node in the topological map. This could easily be done if the robot checks that there are no obstacles between nodes that are close by and forming a loop. Then those nodes could be merged into one node. This would make a map that is more readable for both humans and robot.

This simulation does not take into consideration a dynamic environment. With this algorithm, the robot would not build a correct topological map when there are obstacles in motion. In order to map a dynamic environment, the robot needs to know how to deal with a changing environment for navigation and it needs to have a flexible topological map creation and modification.

REFERENCES

- [1] E. Trucco and A. Verri, “Introductory Techniques for 3-D Computer Vision”, Prentice Hall, Inc, 1998.
- [2] G. Dudek and M. Jenkin, “Computational Principles of Mobile Robotics”, Cambridge University Press, 2000.
- [3] G. Dudek and M. Jenkin, “Reflection on modeling a sonar range sensor”, McGill University, Montreal, Canada, 1993.
- [4] M. Hagen, H. Demuth, M. Beale, “Neural Network Design”, PWS Publishing Company, 1995.
- [5] U. Zimmer, “Robust World-Modelling and Navigation in Real World”, in *Neurocomputing Special Issue*, 13: 247 – 260, 1996.
- [6] S. Simhon, G. Dudek, “A Global Topological Map formed by Local Metric Maps”, in *International Conference on Intelligent Robots*, Victoria, Canada, 1998.
- [7] S. Thrun, “A Bayesian Approach to Landmark Discovery and Active Perception in Mobile Robot Navigation”, Technical Report CMU-CS-96-122, Carnegie Mellon University, Pittsburgh, PA, May 1996.
- [8] S. Thrun, A Bücken, “Integrating Grid-Based and Topological Maps for Mobile Robot Navigation”, in *AAAI/IAAI*, Vol. 2, pp. 944 – 950, August 1996.
- [9] T. Yairi, M. Togami, K. Hori, “Learning Topological Maps from Sequential Observation and Action Data under Partially Observable Environment”, in *The Seventh Pacific Rim International Conference on Artificial Intelligence (PRICAI-02)*, pp.305-314, 2002.
- [10] E. Fabrizi, A. Saffiotti, “Augmenting Topology-Based Maps with Geometric Information”, in *Robotics and Autonomous Systems*, 40(2):91-97, 2002.
- [11] S. Rizzi, D. Maio, M. Golfarelli, “A Hierarchical Approach to Sonar-Based Landmark Detection in Mobile Robots”, in *Proceedings of the 5th Symposium on Intelligent Robotics Systems*, Stockholm, Sweden, pp. 77-84, 1997.
- [12] P. Gaussier, S. Zehen, “Navigating with an animal brain: a neural network for landmark identification and navigation”, in *Intelligent Vehicles*, pp. 399-404, 1994.
- [13] L. Vincent and P. Soille. “Watersheds in digital spaces: an efficient algorithm based on immersion simulations.” *IEEE T. on Pattern Analysis and Machine Learning*, 13(6):583-598, 1991.

- [14] B. Kuipers and Y.T Byun “A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spacial Representation”, in *Journal of Robotics and Autonomous Systems*, 8: 47 – 63, 1991.
- [15] M. Mata, J.M. Armingol, A. de la Escalera and M. A. Salichs “Using learned visual landmarks for intelligent topological navigation of mobile robots”, in *IEEE International Conference on Robotics and Automation*. Taipei, Taiwan, September 14-19 (2003).
- [16] A. Elfes, “Occupancy grids: A probabilistic framework for robot perception and navigation”, DAI-B 51/02, p. 897, Aug 1990.
- [17] R. Chatila and J.P. Laumond. “Position referencing and consistent world modeling for mobile robot.” In *IEEE International Conference on Robotics and Automation*, pp. 138 – 145, 1985.
- [18] L. Nyland, “Capturing Dense Environmental Range Information with Panning, Scanning Laser Rangefinder”, <http://www.cs.unc.edu/~ibr/projects/rangefinder/>, 1999.
- [19] P. Zhang, E. Milios and J. Gu, “Underwater Robot Localization using Artificial Visual Landmarks”, in *IEEE International Conference on Robotics and Biomimetics*, Shenyang, China, Paper no 67, Aug 2004.
- [20] K. Yoon, I. Kweon, “Artificial Landmark Tracking Based on the Color histogram”, in *Intelligent Robots and Systems*, pp. 1918-1923 vol.4, 2001.
- [21] European Navigation Conference 2004. Retrieved 5 January 2005 <<http://www.enc-gnss2004.com>>
- [22] Geneq Laser Range Finder. Retrieved 28 December 2004 <<http://www.geneq.com/catalog/en/laserrf.htm>>
- [23] SensComp Global Components. Retrieved 28 December 2004 <<http://www.senscomp.com/specs/7000%20electrostatic%20spec.pdf> >
- [24] Laser Component Ltd. Retrieved 28 December 2004 <<http://www.lasercomponents.com/www/pdf/distmeas.pdf> >
- [25] European Space Agency. Retrieved 5 January 2005 <<http://www.esa.int>>
- [26] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.
- [27] B. Yamauchi and P. Langley. Place recognition in dynamic environments. *Journal of Robotic Systems*, 14(2):107–120, 1997.
- [28] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic algorithms and

the interactive museum tour-guide robot minerva. *International Journal of Robotics Research*, 19(11):972–999, 2000.

[29] M. J. Matarić. “A distributed model for mobile robot environment-learning and navigation”. Master’s thesis, MIT, Cambridge, MA, January 1990. also available as MIT AI Lab Tech Report AITR-1228.

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

[31] P. Langley, K. Pflieger and M. Sahami, “Lazy Acquisition of Place Knowledge”, *Artificial Intelligence Review*, 11(1-5):315-342, 1997.

[32] R. Sim and G. Dudek. “Learning Generative Models of Invariant Features”, in *Proceedings of Intelligent Robots and Systems (IROS)*, Sendai, Japan, 2004.

[33] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Int. Conf. on Computer Vision*, pp. 1150–1157, Corfu, Greece, September 1999.

[34] M. Montemerlo, N. Roy and S. Thrun, “CARMEN Carnegie Mellon Robot Navigation Toolkit.” Carnegie Mellon University. updated June 11 2003. retrieved 12 February 2005 <<http://www-2.cs.cmu.edu/~carmen/>>

[35] G. Dudek and Robert Sim, “Information on the RoboDaemon robot controller and simulator.” McGill University. updated January 1998. retrieved 12 February 2005 <<http://www.cim.mcgill.ca/~dudek/mobile/robodaemon.html>>

[36] G. Dudek and Robert Sim, “McGill Mobile Robotics Architecture (MMRA): A development environment for multiple robot control and simulation.” McGill University. Retrieved 12 February 2005 <<http://www.cim.mcgill.ca/~mrl/>>

Appendix A

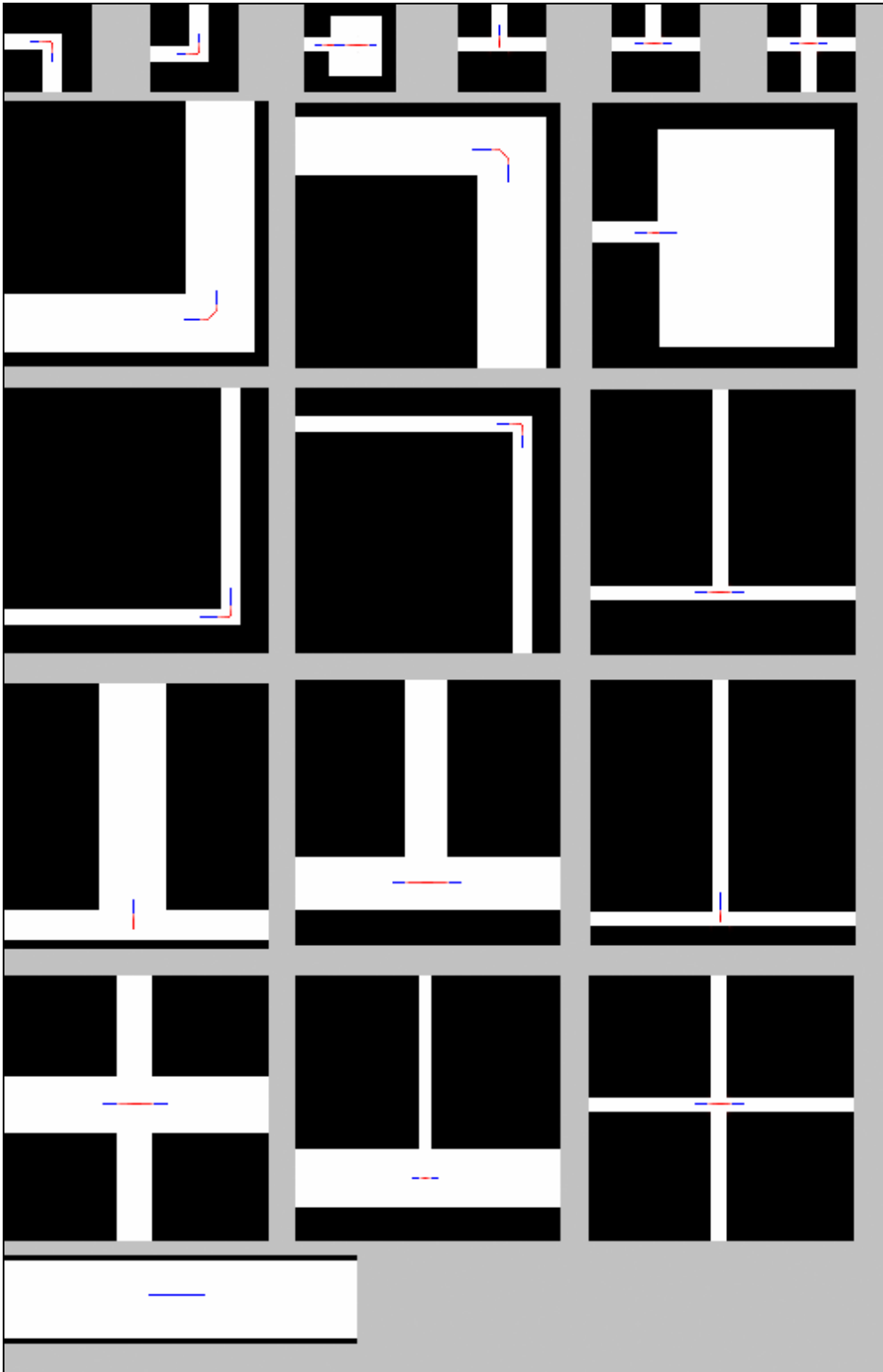


Figure 59 : Training Maps

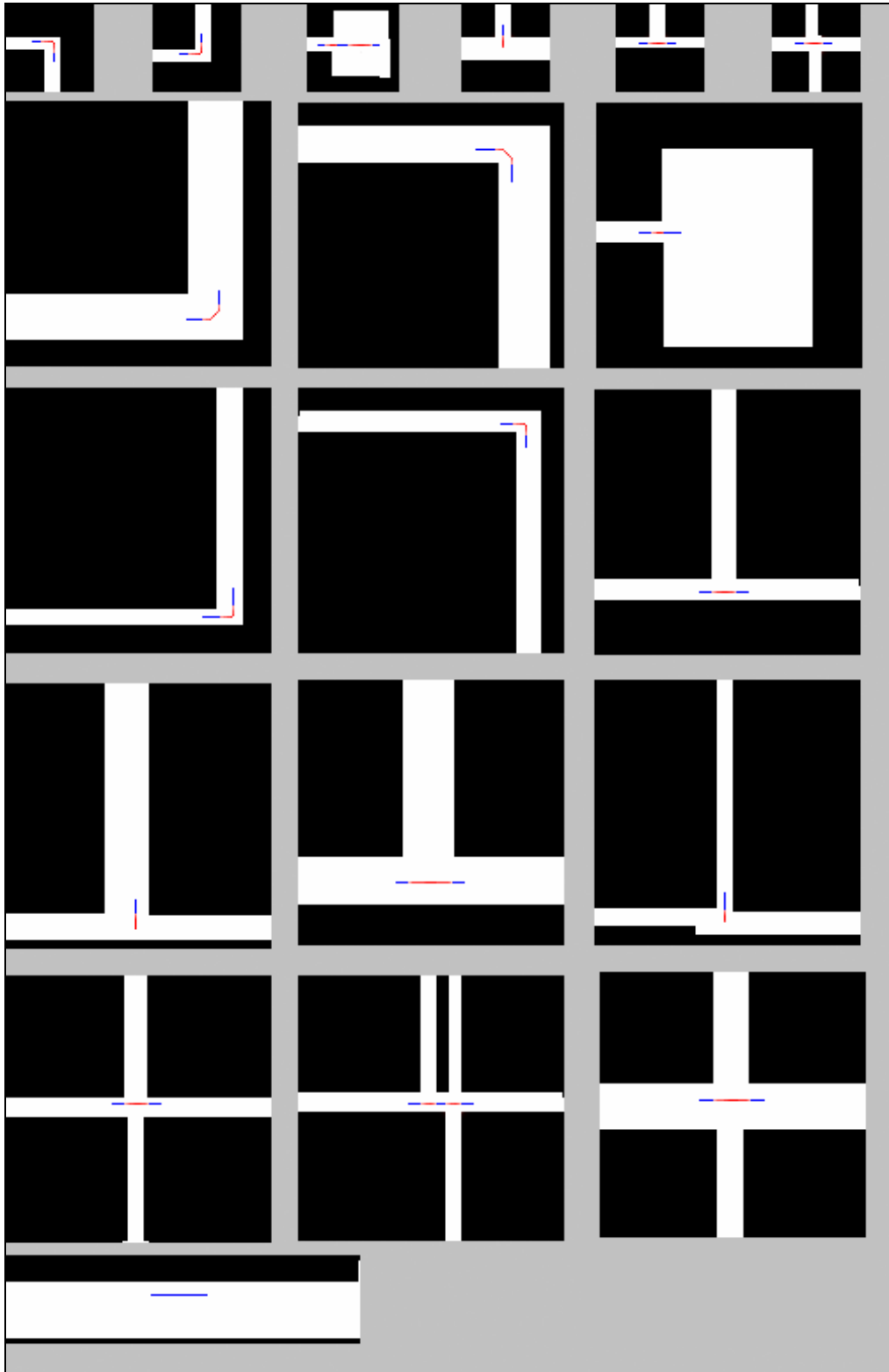


Figure 60 : On Training “Testing Maps”

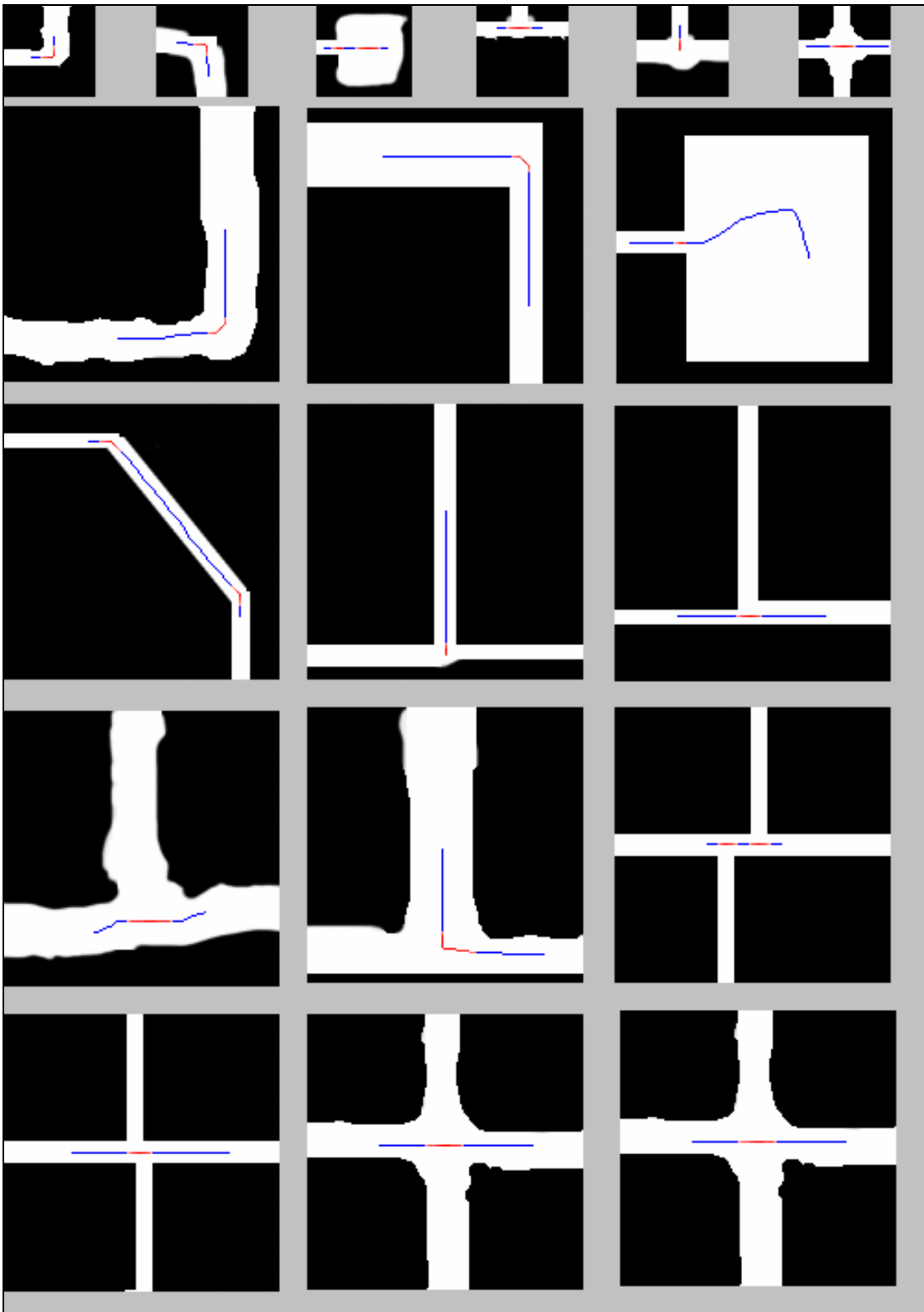


Figure 61 : Testing Maps

	Map 1	Map 2	Map 3	Map 4	Map 5	Map 6	Map 7	Map 8	
NN1	6.31E-02	3.13E-02	2.07E-02	0.1349	8.38E-02	8.91E-02	8.54E-03	2.10E-02	
NN2	3.44E-02	3.83E-02	9.11E-02	0.1211	8.95E-02	6.38E-02	7.65E-02	4.10E-02	
NN3	4.78E-02	2.66E-02	3.99E-02	7.37E-02	5.59E-02	6.18E-02	7.34E-03	5.76E-02	
NN4	2.18E-02	3.24E-02	1.86E-02	8.86E-02	8.54E-02	4.79E-02	1.02E-02	3.31E-02	
NN5	7.46E-02	8.64E-02	3.73E-02	8.54E-02	7.82E-02	7.79E-02	9.86E-03	4.02E-02	
NN6	6.53E-02	5.72E-02	3.28E-02	9.00E-02	8.66E-02	8.27E-02	9.41E-03	3.94E-02	
NN7	2.82E-02	2.37E-02	2.29E-02	0.1176	6.33E-02	5.92E-02	1.11E-02	3.88E-02	
NN8	7.68E-02	5.18E-02	5.75E-02	0.1057	5.30E-02	6.33E-02	1.48E-02	6.02E-02	
NN9	0.1017	4.05E-02	0.1396	9.11E-02	0.1259	7.97E-02	7.95E-02	0.101	
NN10	8.56E-02	6.45E-02	5.55E-02	0.1246	8.30E-02	4.79E-02	1.08E-02	3.78E-02	
	Map 9	Map 10	Map 11	Map 12	Map 13	Map 14	Map15	Map 16	Map17
NN1	7.11E-02	4.36E-02	1.05E-02	4.45E-02	3.28E-02	3.54E-02	4.65E-02	0.1365	6.30E-02
NN2	3.71E-02	3.34E-02	3.08E-02	6.46E-02	3.81E-02	2.03E-02	6.15E-02	0.1532	6.19E-02
NN3	3.64E-02	3.94E-02	1.33E-02	5.33E-02	2.58E-02	3.25E-02	4.34E-02	6.91E-02	7.61E-02
NN4	8.41E-02	4.03E-02	2.34E-02	5.90E-02	2.01E-02	3.91E-02	1.33E-02	0.1388	5.84E-02
NN5	4.60E-02	4.75E-02	1.99E-02	4.98E-02	0.1075	4.10E-02	9.12E-02	0.1488	5.29E-02
NN6	0.1146	4.49E-02	1.52E-02	5.62E-02	5.05E-02	4.26E-02	3.57E-02	0.1385	8.28E-02
NN7	0.1063	3.52E-02	0.104	6.81E-02	3.05E-02	2.82E-02	2.82E-02	0.1334	3.60E-02
NN8	7.21E-02	4.71E-02	2.35E-02	4.32E-02	5.97E-02	3.07E-02	6.34E-02	0.191	7.09E-02
NN9	7.60E-02	3.29E-02	3.04E-02	6.56E-02	5.11E-02	2.82E-02	7.55E-02	0.216	5.70E-02
NN10	8.23E-02	6.94E-02	4.28E-02	7.94E-02	3.48E-02	5.92E-02	4.68E-02	0.156	5.31E-02

Table 2 : Mean Squared Error per Maps & per Neural Network Data

Epochs	0	200	400	600	800	1000
MSE NN1	7.72E-02	2.86E-02	1.52E-02	1.03E-02	7.96E-03	6.68E-03
MSE NN2	7.14E-02	2.69E-02	1.42E-02	9.81E-03	7.88E-03	6.69E-03
MSE NN3	8.81E-02	3.46E-02	1.60E-02	1.02E-02	7.64E-03	6.28E-03
MSE NN4	7.36E-02	2.23E-02	1.11E-02	8.06E-03	6.60E-03	5.74E-03
MSE NN5	7.91E-02	2.63E-02	1.34E-02	9.09E-03	7.09E-03	6.03E-03
MSE NN6	7.56E-02	2.82E-02	1.45E-02	9.59E-03	7.48E-03	6.28E-03
MSE NN7	7.45E-02	2.62E-02	1.51E-02	1.02E-02	7.88E-03	6.56E-03
MSE NN8	7.74E-02	2.81E-02	1.50E-02	1.08E-02	8.73E-03	7.46E-03
MSE NN9	6.97E-02	2.76E-02	1.66E-02	1.25E-02	8.44E-03	6.70E-03
MSE NN10	7.43E-02	2.61E-02	1.31E-02	8.95E-03	6.82E-03	5.85E-03
Epochs	1200	1400	1600	1800		
MSE NN1	5.80E-03	5.30E-03	4.84E-03	4.52E-03		
MSE NN2	5.94E-03	5.39E-03	4.98E-03	4.61E-03		
MSE NN3	5.41E-03	4.85E-03	4.42E-03	4.13E-03		
MSE NN4	5.09E-03	4.64E-03	4.33E-03	4.04E-03		
MSE NN5	5.31E-03	4.79E-03	4.37E-03	4.12E-03		
MSE NN6	5.55E-03	5.01E-03	4.62E-03	4.10E-03		
MSE NN7	5.70E-03	5.08E-03	4.61E-03	4.20E-03		
MSE NN8	6.65E-03	6.04E-03	5.49E-03	5.00E-03		
MSE NN9	5.86E-03	5.23E-03	4.75E-03	4.40E-03		
MSE NN10	4.64E-03	4.27E-03	3.97E-03	3.71E-03		

Table 3 : Evolution of the Mean Squared Error Over Time during Training Data

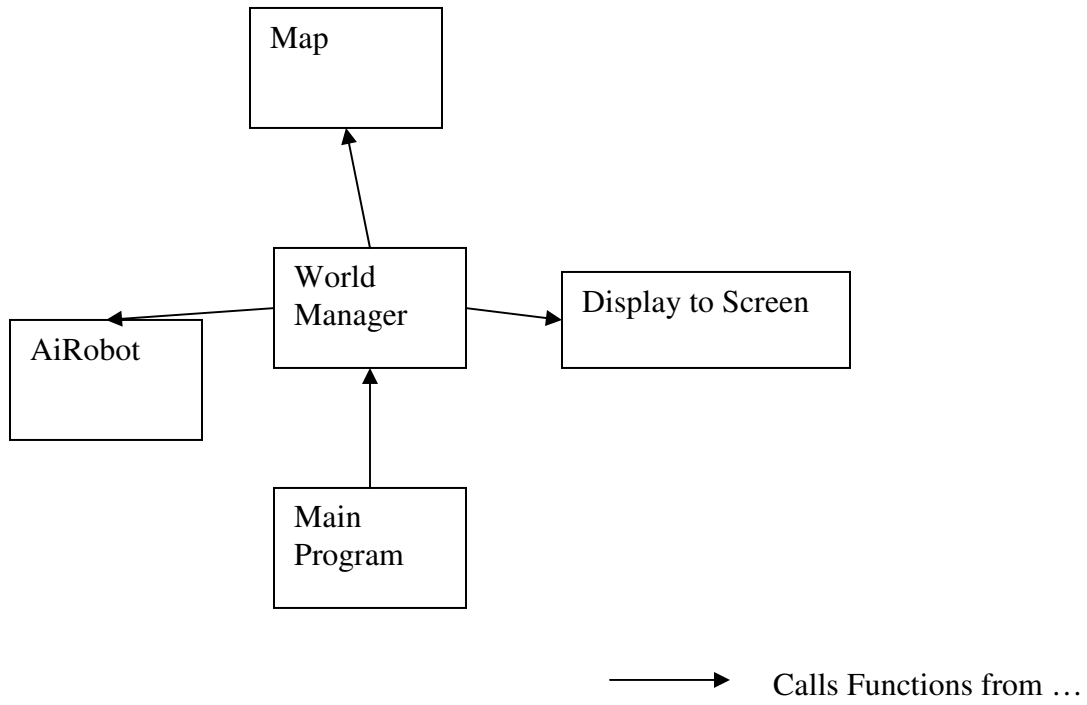


Figure 62 : Program Organization

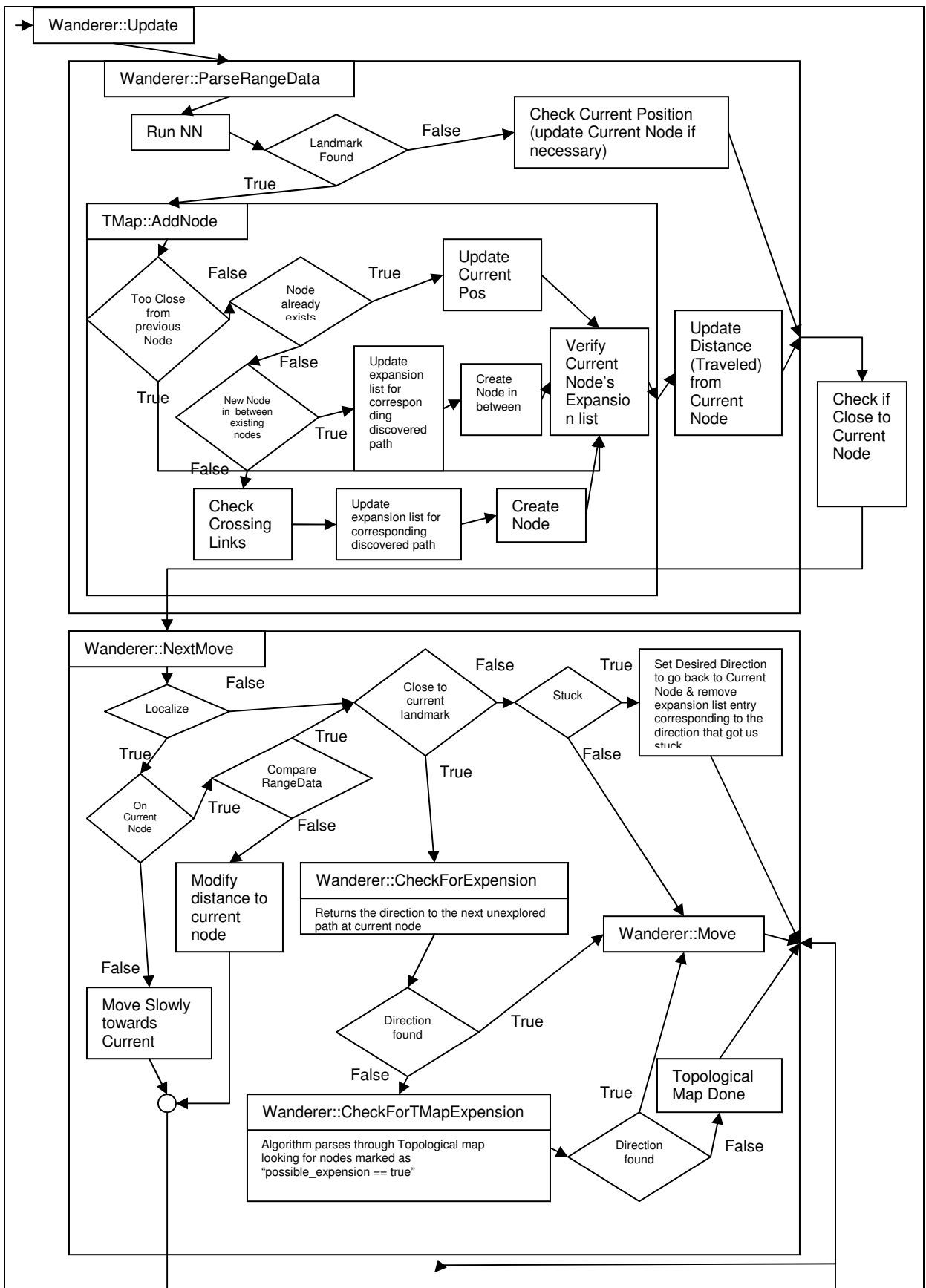
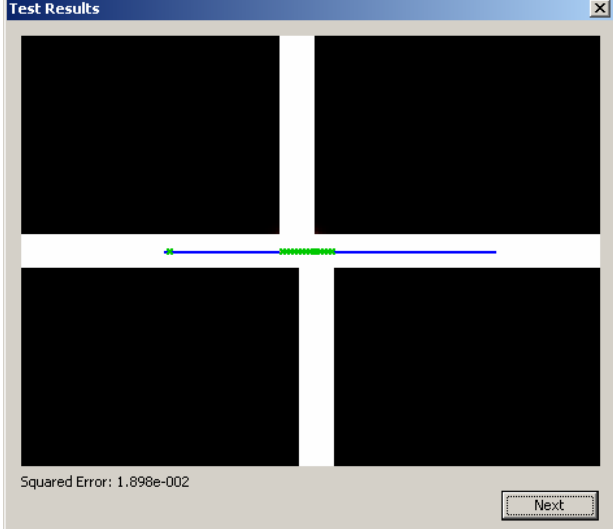
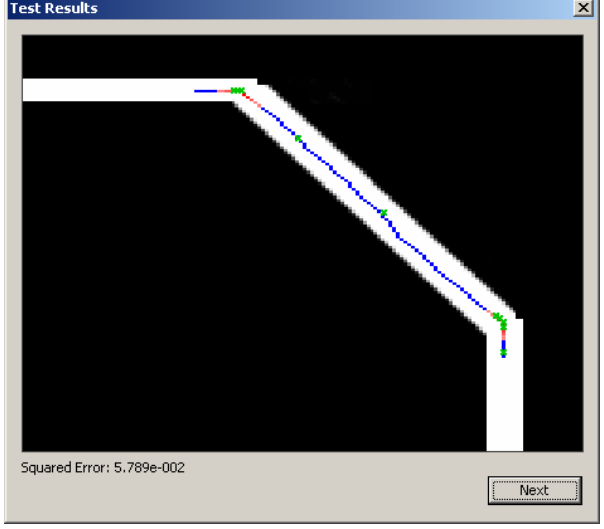
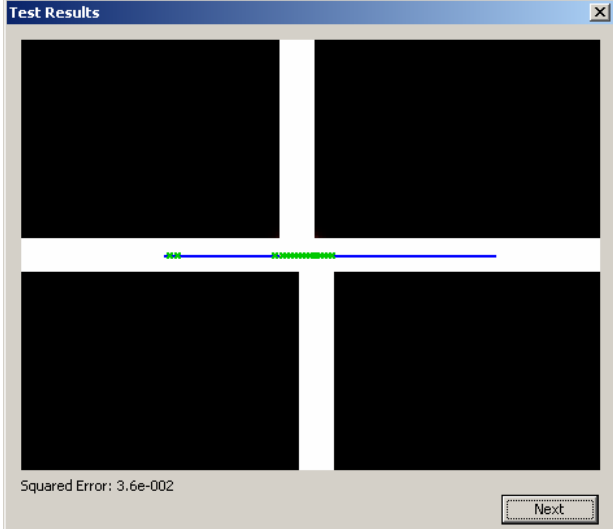
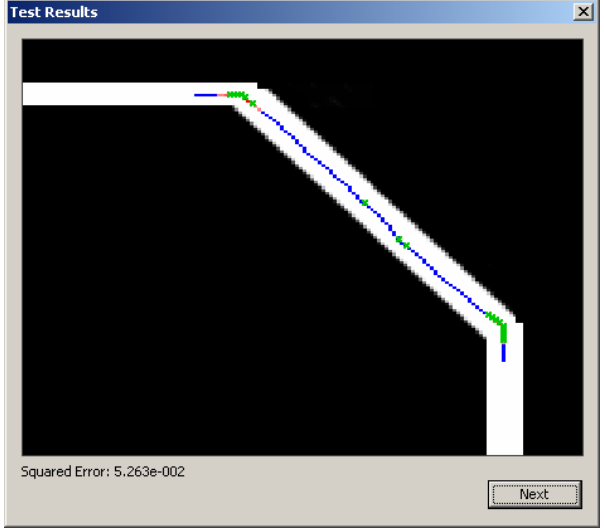
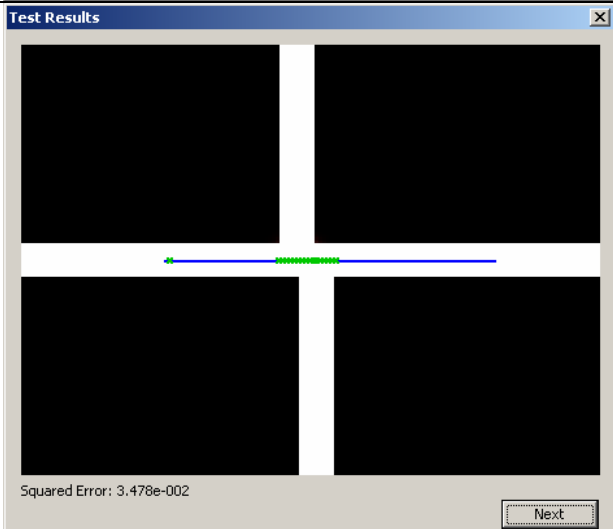
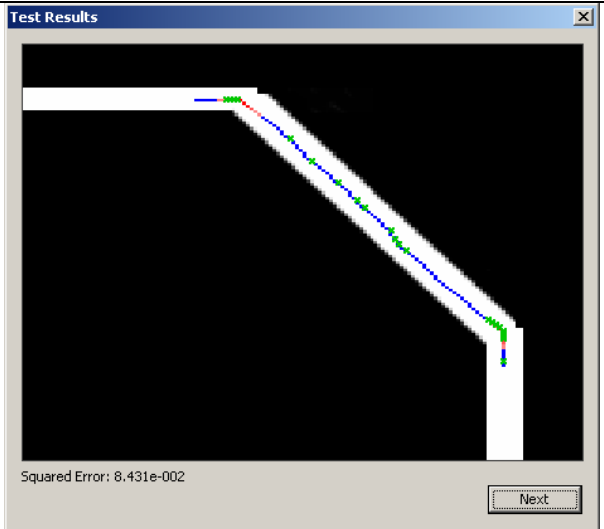
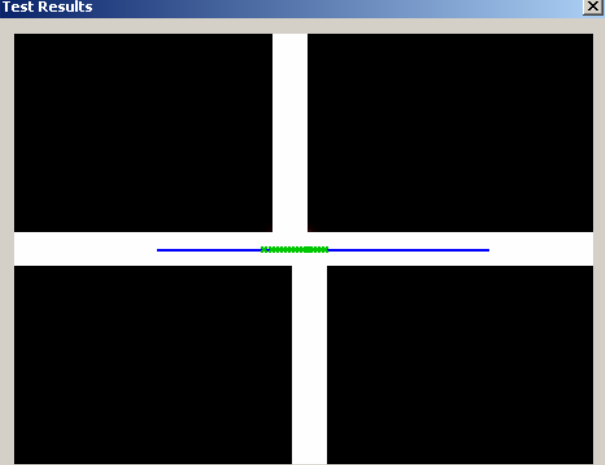
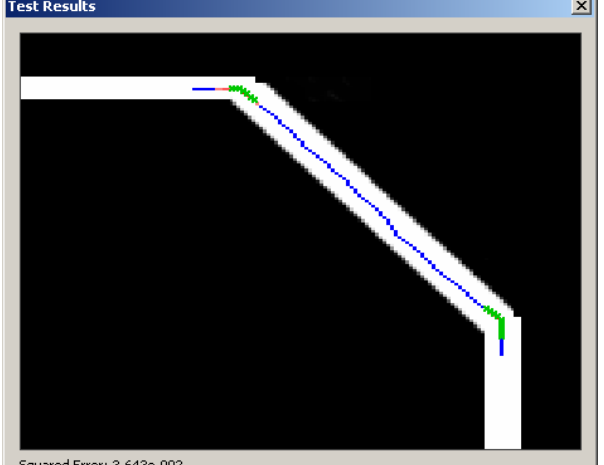
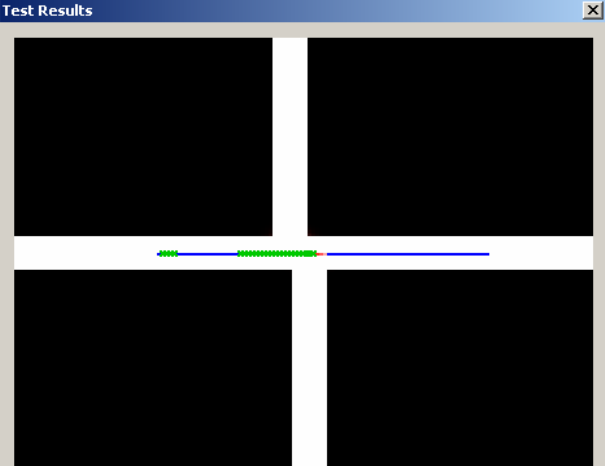
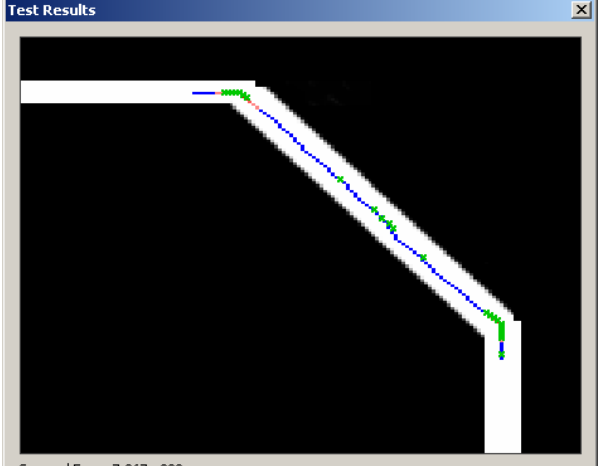
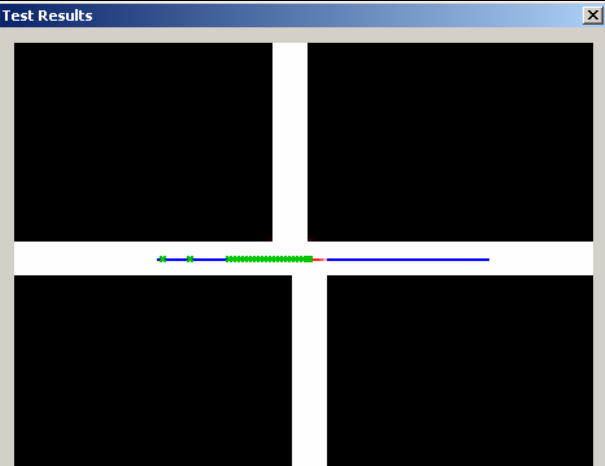
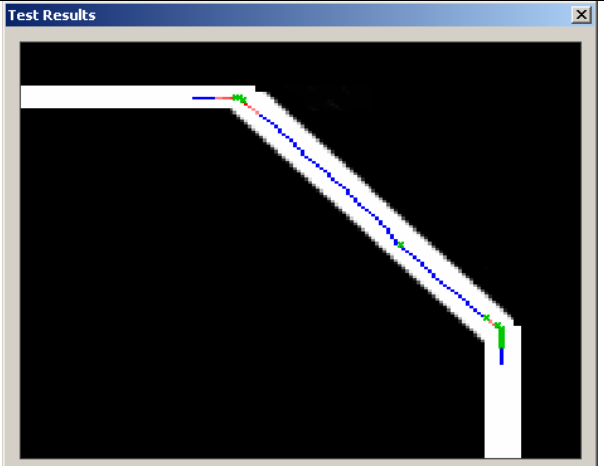


Figure 63 Flowchart of the Robot Update function

	Map 3	Map 17
NN1	<p>Squared Error: 2.204e-002</p>	<p>Squared Error: 6.339e-002</p>
NN2	<p>Squared Error: 9.121e-002</p>	<p>Squared Error: 5.997e-002</p>
NN3	<p>Squared Error: 4.01e-002</p>	<p>Squared Error: 7.545e-002</p>

<p>NN4</p>	 <p>Squared Error: 1.898e-002</p> <p>Next</p>	 <p>Squared Error: 5.789e-002</p> <p>Next</p>
<p>NN5</p>	 <p>Squared Error: 3.6e-002</p> <p>Next</p>	 <p>Squared Error: 5.263e-002</p> <p>Next</p>
<p>NN6</p>	 <p>Squared Error: 3.478e-002</p> <p>Next</p>	 <p>Squared Error: 8.431e-002</p> <p>Next</p>

<p>NN7</p>	 <p>Squared Error: 2.296e-002</p> <p>Next</p>	 <p>Squared Error: 3.643e-002</p> <p>Next</p>
<p>NN8</p>	 <p>Squared Error: 5.611e-002</p> <p>Next</p>	 <p>Squared Error: 7.267e-002</p> <p>Next</p>
<p>NN9</p>	 <p>Squared Error: 0.141</p> <p>Next</p>	 <p>Squared Error: 5.844e-002</p> <p>Next</p>

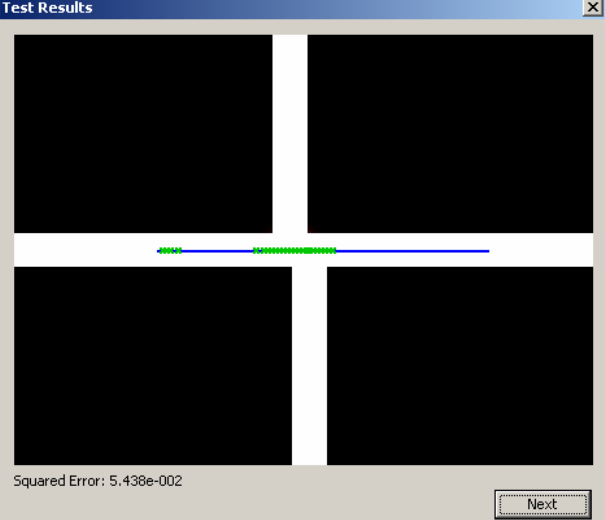
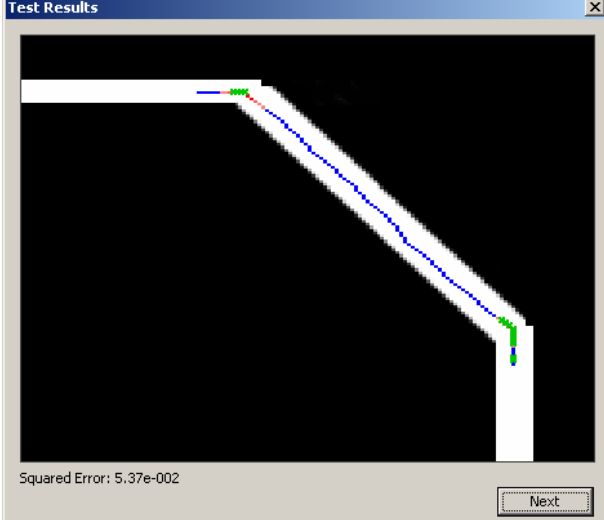
<p>NN10</p>	 <p>Squared Error: 5.438e-002</p>	 <p>Squared Error: 5.37e-002</p>
	<p style="text-align: center;">Next</p>	

Table 4 : Comparing all Neural Networks on Two Different Maps