Artificial Neural Network Approach to Mobile Robot Localization

David Swords

Master of Science (120 credits)
Space Engineering - Space Master

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
David Swords

Artificial Neural Network Approach to Mobile Robot Localization

School of Electrical Engineering
Department of Automation and Systems Technology

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology

Espoo, October 22, 2012
Instructor: Dr. Tapio Taipalus
Aalto University
School of Electrical Engineering

Supervisors:
Prof. Aarne Halme
Aalto University
School of Electrical Engineering
Prof. Thomas Gustafsson
Luleå University of Technology
School of Computer Science, Electrical & Space Engineering
I would like to thank my instructor Dr. Tapio Taipalus who provided great insight into how to approach academic research and project management. Dr. Taipalus provided a welcoming and motivating work environment and it is very much appreciated. I would also like to thank Prof. Aarne Halme and Prof. Thomas Gustafsson who have given me a lot of constructive feedback throughout the duration of the thesis.

I wish to express my sincere gratitude to Prof. Aarne Halme, Tomi Ylikorpi and Dr. Adrian Stoica for allowing me to visit and work at the Jet Propulsion Laboratory (JPL) in Pasadena, California. Dr. Marco Pavone served as my mentor and I am thankful for his efforts. Dr. Christopher Assad made for a pleasant and motivational atmosphere, I would like to thank him for taking time to discuss any concerns or technical issues that arose during my stay. Without the financial help of the Career Services from Aalto University School of Science and Technology the visit to JPL would not have been possible.

Finally, I owe a great deal of gratitude to both Eva Darulova and Eliott Bartley, without whom this thesis would have ended before it started.

Dublin, October 22, 2012

David Swords
In the robotics community, localization is considered a solved problem, however, the topic is still open to investigation. Mobile robot localization has been focused on developing low-cost approaches and there has been great success using probabilistic methods. Parallel to this, and to a much lesser extent, artificial neural networks (ANNs) have been applied to the problem area with varying success.

A system is proposed in this thesis where the typical probabilistic approach is replaced with one based purely on ANNs. This type of localization attempts to harness the simplicity, scalability and adaptability that ANNs are known for. The ANN approach allows for the encapsulation of a number of steps and elements well known in a probabilistic approach, resulting in the elimination of an internal explicit map, providing pose estimate on network output and network update at runtime.

First, a coordinate-based approach to localization is explored: 1D and 2D trained maps with pose estimates. Second, the coordinate-based approach is eliminated in an effort to replicate a more biologically inspired localization. Finally, a path-finding algorithm applying the new localization approach is presented.

**Keywords:** artificial neural network, mobile robot localization, path-finding
## Contents

1 Introduction .......................... 1
   1.1 Motivation and Objectives .................. 1
   1.2 Artificial Neural Networks .................. 3
      1.2.1 Neuron Models .................. 6
      1.2.2 Network Architectures ............... 10
      1.2.3 Learning Paradigms ............... 12
      1.2.4 Training Methods ............... 15
   1.3 Mobile Robot Localization .............. 16
      1.3.1 Localization Paradigms ............ 16
      1.3.2 Localization Methods ............ 18
   1.4 Laser Scanners .................. 26

2 Related Work .......................... 29
   2.1 Common Motivations .................. 29
   2.2 Supervised Approach ............... 31
   2.3 Unsupervised Approaches ............ 34
   2.4 Simulations .................. 35

3 Implementation & Simulation ........... 40
   3.1 Overview .................. 40
   3.2 ROS Simulator .................. 41
      3.2.1 Robot Operating System .......... 41
      3.2.2 Simulator Features ............ 42
      3.2.3 Simulator Configuration .......... 43
      3.2.4 stageros Execution ............ 44
   3.3 Fast Artificial Neural Network Library ........... 47
      3.3.1 FANN Features ............ 47
      3.3.2 FANNTool .................. 48
      3.3.3 Integration with ROS ............ 48
List of Tables

4.1 ANN Configuration and Training Values for Simulation without Items ........................................ 66
4.2 ANN Configuration and Training Values for Simulation with Items ........................................ 68
4.3 ANN Configuration and Training Values for Simulation of X-axis Position .................................. 73
4.4 The Training Configuration for Landmark Recognition ................................................................. 83
4.5 The Training Configuration for Path-finding ..................................................................................... 91

A.1 The MSE after 2000 Epochs using each Training Method for Rotation without Items ................ II
A.2 The MSE after 2000 Epochs using each Training Method for Rotation with Items ....................... II
A.3 The MSE after 2000 Epochs using each Training Method for X-axis Position ............................. II
A.4 The Index of Activation Functions .................................................................................................. III
A.5 The MSE after 2000 Epochs using each Activation Function Combination for Rotation without Items (Row: Hidden Layer Activation Function, Column: Output Layer Activation Function) .......... IV
A.6 The MSE after 2000 Epochs using each Activation Function Combination for Rotation with Items (Row: Hidden Layer Activation Function, Column: Output Layer Activation Function) .......... V
A.7 The MSE after 2000 Epochs using each Activation Function Combination for X-axis Position (Row: Hidden Layer Activation Function, Column: Output Layer Activation Function) .......... VI
List of Figures

1.1 McCulloch and Pitts neuron model ........................................ 7
1.2 Threshold function plot ....................................................... 8
1.3 Piecewise-linear function plot ............................................ 9
1.4 Sigmoid activation function plot ......................................... 10
1.5 SLP diagram ................................................................. 11
1.6 MLP diagram ............................................................... 12
1.7 RNN diagram ............................................................... 13
1.8 Illustration of symmetric locations ..................................... 18
1.9 Illustration of ML in 1D ...................................................... 23
1.10 Illustration of MCL in 1D .................................................. 24
1.11 Image of LIDAR equipped mobile robot (Courtesy of Wikipedia) 27
1.12 Illustration of LIDAR operation ......................................... 28

2.1 Simulator of [Sethi and Yu, 1990] ........................................ 36
2.2 Room configurations of (Sethi and Yu, 1990) ......................... 36
2.3 Simulator of [Wilkes et al., 1990] .......................................... 37
2.4 Training and validation poses of [Townsend et al., 1995] ........ 38
2.5 Training and validation poses of [Scolari et al., 2003] ............ 38

3.1 Stage simulation of multiple robots (Courtesy of Brian Gerkey) 42
3.2 Bitmap arena used during simulation ................................... 44
3.3 stageros simulation in operation .......................................... 45
3.4 rxgraph overview of running ROS nodes ............................... 46
3.5 Screenshot of FANNTOOL .................................................. 49
3.6 Screenshot of the Stage configuration for Rotation Experiments 52
3.7 Graph of the ROS Nodes running during Rotation Experiments 54
3.8 Screenshot of the Stage configuration for 1-D + Rotation Experiments ............................................. 55
3.9 Graph of the ROS Nodes running during 1-D + Rotation Experiments ............................................. 57
## Symbols and Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>ISS</td>
<td>International Space Station</td>
</tr>
<tr>
<td>LTU</td>
<td>Luleå University of Technology</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration, USA</td>
</tr>
<tr>
<td>Aalto</td>
<td>Aalto University</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>NNN</td>
<td>Natural Neural Network</td>
</tr>
<tr>
<td>FFNN</td>
<td>Feed-forward Neural Network</td>
</tr>
<tr>
<td>RFNN</td>
<td>Region- and Feature-based Neural Network</td>
</tr>
<tr>
<td>RBFN</td>
<td>Radial Basis Function Networks</td>
</tr>
<tr>
<td>SLP</td>
<td>Single-layer Perceptron</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi-layer Perceptron</td>
</tr>
<tr>
<td>AF</td>
<td>Activation Function</td>
</tr>
<tr>
<td>ACO</td>
<td>Ant Colony Optimization</td>
</tr>
<tr>
<td>SA</td>
<td>Simulated Annealing</td>
</tr>
<tr>
<td>BM</td>
<td>Boltzmann Machine</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization</td>
</tr>
<tr>
<td>ML</td>
<td>Markov Localization</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>FANN</td>
<td>Fast Artificial Neural Network</td>
</tr>
<tr>
<td>MCL</td>
<td>Monte Carlo Localization</td>
</tr>
<tr>
<td>SONN</td>
<td>Self-organizing Neural Network</td>
</tr>
<tr>
<td>DR</td>
<td>Dead Reckoning</td>
</tr>
<tr>
<td>EKF</td>
<td>Extended Kalman Filter</td>
</tr>
<tr>
<td>MRL</td>
<td>Mobile Robot Localization</td>
</tr>
<tr>
<td>MRN</td>
<td>Mobile Robot Navigation</td>
</tr>
<tr>
<td>ANNL</td>
<td>Artificial Neural Network Localization</td>
</tr>
<tr>
<td>OCR</td>
<td>Optical Character Recognition</td>
</tr>
<tr>
<td>SLAM</td>
<td>Simultaneous Localization and Mapping</td>
</tr>
<tr>
<td>SOFNN</td>
<td>Self-organizing Fuzzy Neural Network</td>
</tr>
<tr>
<td>SOM</td>
<td>Self-organizing Map</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Networks</td>
</tr>
<tr>
<td>ESN</td>
<td>Echo State Network</td>
</tr>
<tr>
<td>SL</td>
<td>Supervised Learning</td>
</tr>
<tr>
<td>USL</td>
<td>Unsupervised Learning</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>LIDAR</td>
<td>Light Detection and Ranging</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>IMU</td>
<td>Inertial Measurement Unit</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System</td>
</tr>
<tr>
<td>FOV</td>
<td>Field of View</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>BSD</td>
<td>Berkeley Software Distribution</td>
</tr>
<tr>
<td>FLTK</td>
<td>Fast Light Toolkit</td>
</tr>
<tr>
<td>LVQ</td>
<td>Learning Vector Quantization</td>
</tr>
<tr>
<td>LM</td>
<td>Levenberg-Marquardt</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter-process Communication</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Squared Error</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

“Do just once what others say you can’t do, and you will never pay attention to their limitations again.”
- James Cook

1.1 Motivation and Objectives

Mobile Robot Navigation (MRN) can be reduced to 3 problems: finding a current pose estimate relative to a goal, finding a navigable path between poses and the successful execution of a found path. The initial problem is that of Mobile Robot Localization (MRL). MRL itself can be reduced to 3 systems: data collection and processing to build an observation model, a description of the robot kinematics to form a movement model, and to update the pose estimate, a relationship between the first two is found to form an observation-movement model.

Within the robotics community, MRL is considered a solved problem (Durrant-Whyte and Bailey, 2006), however the topic is still open to investigation. The research area has been focused on the development of low-cost approaches, those that do not require heavy computation or sophisticated sensors and there has been great success using probabilistic methods (Thrun et al., 2005). Concurrently and to a much lesser extent, Artificial Neural Networks (ANNs) have been applied to the problem area with varying success.
1.1 Motivation and Objectives

The type of localization focused on in this thesis is that of wheeled mobile robots. Examples of such robots would include the J2B2 (García 2008), Rolloitro (Taipalus 2011) or Turtlebot (Gerkey and Conley 2011). The Dead Reckoning (DR) method forms the foundation for localization with wheeled mobile robots. Unfortunately, with DR, errors are cumulative, mostly due to slippage, drift or an uneven surface. Pose updates need to be made independently of previous movements in order to keep the robot’s current pose estimate accurate, especially in the case of long and winding paths. It is for this reason that effective localization needs one or more types of sensor to compensate for the accumulated error in the others. Routinely, this combined localization estimate is performed with probabilistic methods.

The sensing technology used to augment the DR method typically includes either sonar (Leonard and Durrant-Whyte 1992), laser range-scanner (Bosse et al. 2003) or camera (Davison et al. 2007) systems. As a robot travels through a known environment, it attempts to recognize features with one or more of the previously mentioned sensors and then translate that feature onto a known map. If the feature is successfully found, pose estimation can be calibrated and the cumulative error up to that point can be diminished or even eliminated entirely. Feature matching techniques range from the geometrical (Mouaddib and Marhic 2000) to the Extended Kalman Filter (EKF) (Leonard and Durrant-Whyte 1991). The geometrical applications provide relative displacement and rotation, applying direct transforms to features on a map. The EKF applications keep track of environmental features and then make updates based on that accumulated knowledge. The probabilistic approaches like the EKF are typically chosen due to their robustness, but are more complicated than the geometrical.

Adequate computational power is a necessity in order to maintain an accurate pose estimate in realtime. Typically, mobile robots have onboard computation only. In recent years, offboard architectures have become available with cloud computation (Chen et al. 2010), but used to a much lesser extent than onboard computation. The amount of computational resources, including memory and processing, depends on the size of the environment. The larger the map, the larger the memory to store it. The larger the possible number of poses, the larger the processing power needed to search for one.
1.2 Artificial Neural Networks

An Artificial Neural Network (ANN) is a mathematical model inspired by the functionality of Natural Neural Networks (NNN). The strength of ANNs lie in their atomic simplicity, scalability, adaptability and parallelism. ANNs are most commonly implemented in software, but hardware implementations also exist. For the purposes of this thesis, the implementation is purely software-based.

In this thesis, ANNs are used as the core of the localization process inspired by how the animal brain localizes itself. To this point, the relationship between MRL and ANNs has been mostly restricted to simple control systems (Lin and Goldenberg 2001), instead, the replacement of the previously mentioned probabilistic approach with that of an ANN approach is proposed. The benefits of such an approach mirror those generally known of ANNs, again, their simplicity, scalability, adaptability and parallelism. It is known that the approach is possible as it already exists within animals. Cognitive science is only just starting to understand animal localization (Moser et al. 2008).

The goal of this thesis is to demonstrate the successful application of ANNs to MRL resulting the presentation of an entirely new approach. The effort is made in part to make MRL simpler and faster. Specifically, exploring the mapping of laser range scans to cartesian-based coordinate systems, then exploring non-coordinate-based systems for landmark recognition and path-finding.

The first chapter will introduce the basics behind ANNs, MRL and laser range-scanners. The second chapter will review related work on the application of ANNs to MRL. The third chapter will describe the implementation of the approach and the setup of the simulator. The fourth chapter will describe the experimentation involved in developing the proposed approach. The final and fifth chapter will discuss the results and propose possible future work.

1.2 Artificial Neural Networks

ANNs are a mathematical model that has been designed to resemble the functionality of NNNs. Essentially, it is an attempt to emulate the functionality of the brain to solve certain classes of problem. These ANNs can be implemented in software or hardware.
1.2 Artificial Neural Networks

ANNs were first introduced in the 1940’s ([McCulloch and Pitts, 1943] and [Wiener, 1948]) and followed by intermittent research interest until a resurgence in the 1980’s ([Kohonen, 1982] and [Hopfield, 1982]), which continues to the present day. This was helped mostly by concept innovation and the introduction of cheap personal computers. ANNs are an attempt to replicate the parallel computational power of NNNs and are very successful as non-linear statistical data modeling tools. ANNs have seen great success in the areas of regression analysis, data processing and classification.

ANNs consists of varying sized collections of simple single units called neurons. When this collection configures itself correctly, it can store knowledge. The knowledge is not manually inserted in the network on construction, but a learning phase is used, where the network is taught the knowledge. This knowledge is represented by way of synapses. These synapses have associated weights that are free to vary with the input being learned.

The power of ANNs is derived from two things: generalization and a massively parallel distributed structure. Generalization refers to an ANN's ability to produce reasonable outputs for any number of inputs not encountered during the learning phase. When both the generalization and parallel distribution are combined, it is then possible to solve difficult problems. When applied to a problem, ANNs are typically used as part of a larger system. In addition, certain types of ANNs are more suited to problems than others. The task at hand is divided beforehand into a manageable size and fed accordingly to the ANN. The procedural nature of the teaching of ANNs makes them attractive, because they can capture the mapping present in input-output data with the need of extensive model building.

The following is an outline of the beneficial characteristics of ANNs:

1. **Neurobiological Analogy** - A benefit of ANNs over other AI models is that there is an example of how powerful they can be given the necessary degree of complexity, referring to the human brain. The human brain is this fast, powerful massively parallel and fault tolerant system. The ANN model is a natural and accurate one. A model that is routinely used as a medical research tool.

2. **Contextual Information** - The representation of knowledge within the
1.2 Artificial Neural Networks

Artificial Neural Networks (ANN) is dependent on the connectivity and synaptic weights. With ANNs, each neuron can be affected by the states of all the other neurons.

3. Adaptivity - ANNs adapt to their environment by dynamically changing the synaptic weights. When trained on a set of inputs, then applied to the environment, the ANN can easily be retrained to cope with relatively minor changes in that input. Also, an ANN can be implemented such that it has control over the synaptic weights in real time. It is for these reasons that ANNs are commonly used for adaptive control, pattern classification and signal processing. The more adaptable the system the more robust it will be. However, this is not always the case. Making the ANN too adaptable, could lead to a severe degradation in functionality. For example, if an ANN was exposed to a series of extreme isolated inputs, these would lead to a degradation. Since, adaption to the extreme inputs would bias the network to reacting to such inputs. The inputs that typically will fail to produce the desired response. The aforementioned is referred to as the Stability-Plasticity Dilemma (Carpenter and Grossberg, 1988).

4. Nonlinearity - As with the human brain, the input signals to each neuron are inherently non-linear. While each neuron in a network is non-linear it is inferred that the entire network is non-linear.

5. Input-Output Mapping - Supervised learning is essentially learning with a teacher. There is a set of inputs, each item or the entire set itself will be trained to give a known output. The mapping data can be provided by either theoretical or physical example. This process is carried out for an extended period of time, until the synaptic weight adjustment in the network plateaus. An additional trick is to carry out the training again, but randomizing the order, guaranteeing a good mapping. This thesis focuses on this supervised learning.

6. Evidential Response - An important feature to ANNs when evaluating their effectiveness is being able to get a measure of their certainty. This comes into play often, and can be used to avoid ambiguous patterns by ignoring them.

7. Uniformity of Design and Analysis - There is beauty in the simplicity of the ANN. The division of elements is such that, the neurons are
a consistent representation in different areas of research. This makes it possible for algorithms and theories to be easily implemented and shared.

1.2.1 Neuron Models

A neuron is an information processing unit that is fundamental to the operation of an ANN. Figure 1.1 shows the model of a neuron which forms the basis for designing ANNs. Figure 1.1 provides an overview of the relationship between the synapses, the weights, the summation and Activation Function (AF). Each synapse is characterized by a weight, this means that a signal from \( x_n \) of the input of the synapse \( n \) connects to the neuron \( k \), this is then multiplied by the synaptic weight \( w_{kn} \). It should be noted, that unlike in a NNN, the synaptic weights can have a value from negative to positive. The adder of the summing junction seen in Figure 1.1 does no more than add the weight’s synaptic values together, it is a linear combiner. The AF is denoted by \( f() \). This switches the neuron on and off for the appropriate summed weights and it limits the output signal. It may be noticed from Figure 1.1, there is a biased or fixed input \( b \) added. The purpose of this bias is to raise or lower the net input needed in order to activate the neuron. The elements of the neuron are described in a strictly mathematical notation. The neuron \( k \) is then described by Equations 1.1 and 1.2.

\[
\begin{align*}
    u_k &= \sum_{n=1}^{m} w_{kn} x_n \quad (1.1) \\
    y_k &= f(u_k + b) \quad (1.2)
\end{align*}
\]

In Figure 1.1, \( x_1, x_2, \ldots, x_n \) represent the input signals. Then, \( w_{k1}, w_{k2}, \ldots, w_{kn} \) represent the synaptic weights associated with each of those inputs for neuron \( k \). The output from the adder or linear combiner is given by \( u_k \) above in Equation 1.2, \( y_k \) is simply the output of the neuron \( k \). \( w_0 \) then applies an affine transformation to \( u_k \) to produce \( v_k \) as shown below in Equation 1.3.

\[
v_k = u_k + b \quad (1.3)
\]
Activation Function Types

In this thesis the AF is denoted by $f()$. It defines the output of the neuron in regards to the output of the linear combiner. Basically there are three types of AF, they are listed as:

- Threshold Function
- Piecewise-Linear Function
- Sigmoid Function

The threshold function plot can be seen in 1.4. This type of AF is typically referred as a Heaviside function because its value is 0 for negative values and then 1 for positive values.

$$f(v) = \begin{cases} 
1 & \text{if } v \geq 0 \\
0 & \text{if } v < 0
\end{cases} \quad (1.4)$$

The neuron will give 1 if the input is non-negative and 0 for anything else. The threshold function neuron was originally mentioned by McCulloch and Pitts.
1.2 Artificial Neural Networks

This is the classic all-or-none model. In Figure 1.2, the output of the threshold function is depicted.

![Threshold function plot](image)

Figure 1.2: Threshold function plot

The piecewise-linear function, which is defined[1.5], is seen as an approximation of non-linear one. It consists of a series of linear pieces, combined together provide the non-linearity.

\[
f(v) = \begin{cases} 
1, & v \geq \frac{1}{2} \\
v, & \frac{1}{2} > v > -\frac{1}{2} \\
0, & v \leq -\frac{1}{2}
\end{cases} \tag{1.5}
\]

The most common of the AFs is the sigmoid. It can be seen in [1.4] that it has a distinctive s-shape. The advantage of the sigmoid over the other AFs is that it is real-values and differentiable, this leads to a more realistic representation of the neuron firing. It is said to demonstrate a balance between the linear and non-linear. Equation [1.6] provides an example of the sigmoid function.

By varying \( a \), the slope of the function is varied. As the value of \( a \) approaches infinity a simple threshold function appears, where there is a continuous range of numbers between 0 and 1.
Due to its mathematical properties the sigmoid has, the following Equation 1.7 and 1.8 define it.

\[
f(v) = \begin{cases} 
1, & \text{if } v > 0 \\
0, & \text{if } v = 0 \\
-1, & \text{if } v = 0 
\end{cases} \tag{1.7}
\]

Equation 1.7 is commonly referred to as the \textit{signum} function. For the corresponding form of a sigmoid function the hyperbolic tangent function may be used and this defined by Equation 1.8

\[
f(v) = \tanh(v) \tag{1.8}
\]
1.2 Artificial Neural Networks

1.2.2 Network Architectures

The neuron model is reliant on the network architecture. Architecture in this case is how the neurons are interconnected and dictates how the inputs are attached and what form the output will take.

ANN architectures are grouped into three main categories, Single-layered Feedforward (SLP), Multi-layered Feedforward (MLP) and Recurrent (RNN).

Single-Layered Feedforward Networks

When building ANN, the neurons are grouped into layers. For example, input and output neurons are grouped separately. The simplest form of network is with two layers. The first layer being input neurons and the second layer being output neurons, with the latter also functioning as the computational layer. This is a mono-directional system, where the first layer is the Input Layer and the last layer the Output Layer. With these networks, the Input Layer is not taken into account as there are no calculations performed there. An example of a SLP is presented in Figure 1.5.
Multi-Layer Feedforward Networks

[Minsky and Papert, 1969] showed there is a limit to the type of problems that the SLP can solve. To extract higher-order predictions additional layers have to be added between the input and output layers. Each of these internal layers are referred to as a Hidden Layer. The network overall can then be thought of as a black box, as the activity in these hidden layers is not of interest. An example of a MLP is presented in Figure 1.6.

The Input Vector, refers to a set of numerical values that represent the input pattern. In the learning phase, each Input Vector has an associated Output Vector. The Output Vector, refers to a set of numerical values that represent a desired output for a given Input Vector. When the Input Vector is applied to the Input Layer, the overall pattern is then passed on to the hidden computational layer(s) below. As with SLP’s, MLP’s are mono-directional. The mixing of layers is possible, as in the individual outputs of one Hidden Layer, could be connected to the inputs of different Hidden Layers.
1.2 Artificial Neural Networks

Recurrent Networks

SLPs and MLPs are mono-directional, RNNs form a directed cycle. For example, the output of a Hidden Layer could be the input to the same or a preceding Hidden Layer. RNNs have the added functionality of dynamic temporal behavior. Which means, a sequence of Input Vectors can result in a single Output Vector. An example of a RNN is presented in Figure 1.7.

The Echo State Network (ESN) is a type of RNN with a sparsely connected Hidden Layer. With ESNs, the Output Layer is the only layer that can be changed during training.

1.2.3 Learning Paradigms

The three major learning paradigms in ANNs are Supervised Learning (SL), Unsupervised Learning (USL) and Reinforcement Learning (RL).
Supervised Learning

In this paradigm, an *Input Vector* and a corresponding *Output Vector* are provided. The aim is to approximate a function which will map the *Input Vector* to the corresponding *Output Vector*. Typically, a set of varying pairs demonstrating the relationship is provided on training. It is required that a sufficient number of pairs to be provided, a sufficient number of times to the network in order for the function to be approximated, and not too many times so the network cannot generalize.

The training period of the network is performed as long as the weights are still changing. In order to determine in the weights are changing it is common to take the mean-squared error. This will try to take the minimal difference between the output and the target output value.

The SL can be thought of as a teacher-student relationship. The correct answer is always available during the training period, allowing for robustness on the input-output connection. Regression and classification are common tasks where
the SL paradigm is applied.

Unsupervised Learning

With USL, we do not have the pair mapping to learn from. The teacher is no longer available. We have the Input Vector and instead of a corresponding output vector we have a cost function. The goal is to minimize the result of this cost function. We may know a number of properties of the phenomenon and incorporate this as part of the a priori knowledge. The cost function is a reflection of this phenomenon.

Tasks that can be completed with USL are in general estimation problems, including clustering, the estimation of statistical distributions, compression and filtering.

Kohonen (1982) is in the class of USL, which is used in areas like clinical voice analysis, monitoring of industrial plants, statistical data visualization, automatic speech recognition and satellite image classification.

Reinforcement Learning

While SL was seen as learning with a teacher, RL can be thought of as learning with a critic. With SL, a large set of pairs are provided for training, however, in real-life applications a rich dataset is not very likely. RL is similar to SL but that some feedback is given from the environment.

RL is typically used as part of a larger system or algorithm. For example, the moving of a robot arm is based on certain inputs and you may want the arm to move upwards. The connection between the network and the system for controlling the arm is what is meant as the larger system.

There are a great deal of application areas for RL these include, decision-making in autonomous systems, strategy games, low level flight control, collision avoidance and scheduling.
1.2 Artificial Neural Networks

1.2.4 Training Methods

A training stage is required in order to distribute the error function across the hidden layers, corresponding to their effect on the output. This is referred to as backpropagation training, this is the case for the FFNN used in this thesis, the FANN library provides a number of backpropagation methods, incremental, batch, RProp and QuickProp. The following is an outline of how the training algorithm works:

- Repeat the following:
  - Choose a input-output pair and copy it to input layer
  - Pass that input through the ANN
  - Calculate the error between target and actual output
  - Propagate the summed product of the errors and weights in the output layer to calculate the error in the hidden layer
  - Update weights according to the error on that unit

- Until the calculated MSE is below a threshold or the network has settled into a local or global minimum

Incremental

Backpropagation allows for two types of learning, the first, incremental learning, the second, batch learning. With incremental learning, each error propagation is followed immediately by a weight update. With batch learning, many propagations occur before the weights are updated. A drawback of batch learning is that is requires more memory capacity, though alternatively, incremental learning requires more updates.

As the name denotes, the firing errors are propagated backwards, from the output layer to the hidden/inner layers. This means, that the gradient of the error of the network is calculated in regards to the ANN’s modifiable weights. The gradient is then used as a basis for a gradient descent algorithm. The purpose of this algorithm is to minimize the output error. Using backpropagation, convergence on local minima is expected, an optimal configuration is not guaranteed, as in perhaps the case of SVMs.
RProp

Resilient backpropagation or $R_{prop}$ is a batch update algorithm, a supervised learning heuristic for FFNNs. $R_{prop}$ is presented in the works (Riedmiller, 1993) and (Riedmiller and Braun, 1993). $R_{prop}$ only takes into account the sign of the update rule and not its magnitude, essentially eliminating the harmful influence of partial derivative on the weight update step, acting independently on each neuron.

QuickProp

By using information on the curvature of the error surface it is possible to speed up learning. This information can be gained from the second order derivative of the error function. An assumption is made about the error surface being locally quadratic, so that it attempts to jump one step off of the current position into the minimum of the parabola. The theoretical underpinning of Quickprop is outlined in the work of (Fahlman, 1988).

1.3 Mobile Robot Localization

(Thrun et al., 2005) describes MRL as the problem of estimating a robot’s coordinates relative to an external reference frame. In order to understand MRL a collection of the important aspects have been assembled.

1.3.1 Localization Paradigms

Local & Global Localization

Localization problems are difficult to varying levels and are defined by the type of information that is available initially and at runtime. Below is a description of what the difference is between local and global localization.

Local Localization Describes when the initial pose of the robot is known. It is referred to as a local problem because the uncertainty in a robot’s pose is


1.3 Mobile Robot Localization

confined to a region near the true pose. Solving this problem means focusing on the noise produced by a robot’s motion, which can be approximated using a Gaussian distribution.

**Global Localization** Describes when the initial pose of the robot is unknown. This means that a robot could be initially placed anywhere in an environment and not know where it is. It is considered to be a more difficult problem than local localization. The boundedness of the pose error cannot be assumed, it is therefore the case that a pose cannot be effectively approximated using a Gaussian distribution.

A more difficult version of the global localization problem is that of the *kidnapped robot* problem. This describes a robot being carried away from one location and placed in another. The difficulty behind this problem is that a robot might think it knows where it is, but may in fact not. A robot needs to be able to say if it does not know where it is just as importantly as if it does. The *kidnapped robot* problem is one to solve in the name of robustness of global localization, as the situation where a robot is transported from one place to another is unlikely, but to recover from such events is essential.

**Active & Passive Localization**

The motion of a robot can have an effect on a localization algorithm, how much so depends on if it is active or passive.

**Passive Localization**

Provides a localization that simply observes the motion of a robot. The motion of a robot can be controlled by some other way, with no intention of facilitating the localization.

**Active Localization**

Much more commonly there are active localization algorithms that control a robot such as to maximize the accuracy and minimize the cost. The active approaches usually yield better results than the passive ones. In Figure 1.8 the estimated location of a robot being at two locations is illustrated. The reason for the dual location is down to the symmetry of the hallway and a robot traversing
the environment beforehand. If a robot was to move into one of the rooms, only then a correct estimation of location can be made. It is in situations like these that active localization performs better, as instead of waiting for a robot to randomly travel into one of the rooms, an active localization can correct this automatically.

![Illustration of symmetric locations](image)

While the above might be sufficient if the goal of a robot is just to localize, this is not practical. With an active localization, control is required at all times, complicating things if other tasks need to be accomplished. A robot needs to localize itself regardless of what task it is performing. There are a number of ways of approaching this, one would be to combine both a task being performed and the localization, and another is to build an active localization on top of a passive one.

### 1.3.2 Localization Methods

[MRL](#) is an area that has been well researched, as a result there are a wide range of localization methods. The most widely used methods will now be described.
Dead-reckoning

DR is the most simplistic MRL method in common use. In most cases DR is referred to also as odometry. Odometry is described by (Thrun et al., 2005) as vehicle displacement derived by an onboard odometer along the path of travel. The odometer instrumentation in this case is usually a set of optical encoders on the wheels. Alternatively there are other sensors that rely on periodic magnetic attraction to determine rotation of the wheels, one which is commonly used in the automotive industry.

Within robotics, it is well known that odometry provides high sampling rates, reasonable short-term accuracy and is relatively inexpensive. The main issue with odometry as a means of localization is that it is prone to the accumulation of errors. These errors increase proportionally with the distance travelled. Odometry is an accepted means of short-term localization, and regardless of the accumulating errors is considered an integral part of a robot navigation system.

The following are a list of the most common reasons that odometry is used in a MRN system:

- Combining the odometry with absolute position measurements can provide more reliable position estimates.
- In order to improve matching correctness and achieve short-term processing times, under the assumption that a robot can maintain its position well enough to allow for landmark recognition.
- It may be the case that no other information is available, possibly meaning the lack of any landmarks for deduce an external reference frame.
- Routinely odometry is used to fill in the gaps between position estimates with landmarks. Since, it would be impractical to have a continuously visible landmark at each time step of the localization algorithm.

The heading of the robot can be derived in the following ways:

1. Typically derived from differential odometry
2. A gyro unit or a magnetic compass
3. An onboard steering angle sensor

In the case of straight-line motion, the position is described in increments of X (Equation 1.9) and Y (Equation 1.10) values with the following:

\[
x_{n+1} = x_n + D\sin\theta \\
y_{n+1} = y_n + D\cos\theta
\]

where:

- \( D \) = displacement
- \( \theta \) = heading

While the above considers vehicle displacement, alternatively time elapsed can be considered also.

**Landmark Navigation**

Sensory input can be harnessed for landmark recognition, aiding in the correction of odometry data. Geometrical shapes like lines, rectangles and circles are classically considered landmarks, however this could also include barcodes or some other machine readable pattern. Typically, when landmarks are chosen they are at fixed positions, and the coordinates are recorded to perform later localization in relatively. Landmarks need to be as easily identifiable, and to what degree depends wholly on the sensor being used, allowing for sufficient contrast to the background. If using a laser scanner, landmarks would need to be geometrical, while if a camera is used, a barcode could be used. Regardless of what kind of landmark is used it has to stored in a robot’s memory to be useful. Now, the task becomes to recognize the landmarks reliably and to calculate the robot’s position in relation to them.

In order to limit the search space for possible landmarks, an assumption is made, that the robot can only detect landmarks in a limited area. This simplifies the
problem somewhat, as long as there is an approximate robot pose known. It is with this in mind that the importance of accurate odometry is necessary for landmark recognition.

**Map Matching**

Map matching or "map-based positioning" is a technique where a robot automatically generates a map of its environment using onboard sensors. The generated map in this case is of the local area, this is then matched against a pre-stored global map. It is when a match is found, then the pose of a robot can be calculated. Matching is achieved is first by extracting features from a sensor and second finding a correspondence between what is being seen and the pre-stored map.

The following are advantages to map-based positioning:

- Makes use of the unmodified indoor environment to estimate the pose.
- Routinely maps of an environment need to updated, as indoor environments are typically dynamic. Map-based positioning can be used as a means to update a stored global map. The updated global map can be used to improve tasks like path planning.
- Even if an environment is mapped, new areas can become visible and need mapping, with map-based positioning the generation of a new map is possible.

The following are disadvantages mostly related to the specific requirements that allow satisfactory navigation.

- A satisfactory number of stationary and easily recognizable landmarks to perform a match
- The global map that is used for matching, it has to be accurate to meet the demand of the task
- Quality sensors and sufficient computational power to process sensor data
There are two types of matching algorithms, feature-based and icon-based. Icon-based pose estimation pairs sensor data with features from the global map and tries to minimize the distance error between range values and their corresponding features in the global map. The feature-based method matches every range value to the global map instead of limiting it to a small set of features as is the case for icon-based.

In a range-based system long walls and edges are used most commonly as features and in order to reduce the likelihood of mismatch the more features that are used in a match the better.

**Markov Localization**

Bayes’ theorem forms the basis of probabilistic localization algorithms. As first seen in the work of Fox et al. [1999b], Markov Localization (ML) refers to the direct application of Bayes’ theorem to localization. With ML, it is possible to perform state estimation from sensor data. When ML is described as a probabilistic algorithm this means that alternatively to having a single hypothesis as to the location of a robot in the world, a probability distribution is over all locations is maintained. The advantage of doing this is that it allows a robot to weigh a number of hypothesis in a mathematically sound way.

In order to illustrate the basic concepts behind this algorithm a simple example is included in Figure 1.9. In Figure 1.9, there is an 1D environment in which a robot resides, this means that the robot is on a fixed line and can only move horizontally. A robot then placed somewhere along the line and not told where. In the first step from Figure 1.9 it can be seen that there is a uniform distribution over all locations, as no notable landmarks have appeared. In the second step from Figure 1.9 a robot finds itself next to a door. As a result of being next to a pronounced landmark, the probability of places that are next to doors are increased and everywhere else is lowered. The current probability distribution is not sufficient for global localization because of the multiple possible locations. In the third step from Figure 1.9 a robot is moving away from the clear feature and the probability distribution becomes skewed still resulting in multiple possible locations but with a less clear certainty of being next to a door. The motion of the robot is being incorporated into belief distribution.
The previous observations next to the second door have been multiplied into the current belief, this leaves the probability centered around a single location instead of multiple. Finally, in step four from Figure 1.9 it can be see that the only location to have a door landmark again is presented, this means that the probability raises to one that a robot is in front of one particular door, and the robot is now localized.

Figure 1.9: Illustration of ML in 1D

Monte Carlo Localization

After the second approach is called Monte Carlo Localization (MCL) (Fox et al., 1999a). Considering how recent it is, to date MCL has become the most popular algorithm. Just like MCL applies to both global and local localization problems. It basically works by using particle filters to estimate posteriors over robot poses. MCL is effective of over a wide range of localization problems and is pretty easy to implement.

In order to illustrate the basic concepts behind this algorithm also a simple example is included in Figure 1.10. In Figure 1.10, just as with ML, a 1D
environment in which the robot resides is considered. The robot is on a fixed line and is only allowed to move horizontally. The robot is then placed somewhere along the line and not told where. As seen in step one of Figure 1.10 with the initial deployment, the global uncertainty is achieved with randomly and uniformly distributed pose particles. In step two of Figure 1.10 the robot begins to sense a door, with this MCL assigns a weight signifying the importance of the particle.

![Figure 1.10: Illustration of MCL in 1D](image)

The third step of Figure 1.10 shows that the particle set has been resampled and the robot’s motion incorporated. What is left after this step is a new particle
set which now has uniform weights, but particles have been redistributed to have a higher number near the three places the robot likely is. It is with this third step, the robot has also moved in front of the second door, in step four of [1.10] it can be seen what this new measurement has done to the particle set. New non-uniform importance weights have been added to the particle set. It is at this point that we can see that the bulk of the probability mass is centered at the second door. In step five, there is a repeat of the previous steps and a new particle set is generated based on the motion model. Finally, the particle sets reproduce what would be calculated by an exact Bayes’ filter and correctly approximates a posterior.

Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is a problem area within robotics that is the process in the building a map of an unknown environment, or alternatively updating a previously given map of an environment while at the same time keeping track of the current location of a robot.

In order to determine location, maps are used, and they allow for a depiction of the environment for navigation and path planning. By using a map a robot can generate a perception of the environment, this is then compared to reality. If the quality of the perception of the environment decreases, a location estimate can still be reasonably maintained. A drawback about the map localization is that it can become out of date, typically real-life environments are dynamic and this isn’t depicted in the a priori map.

Due to the complexity of the tasks of both locating and mapping and the errors that arise, a coherent solution to both problems isn’t apparent. SLAM is a concept that allows for the binding of both these tasks in a loop, therefore supporting each and accomplishing the goal. To summarize, the iterative feedback from one task to another enhances both.

Mapping is the problem of integrating the information gathered by a set of sensors into a consistent model and depicting that information as a given representation. It can be described by the first characteristic question, What does the world look like? Central aspects in mapping are the representation of the environment and the interpretation of sensor data.
The minimal requirements that need to be met before SLAM can be performed are the following:

- the kinematics of the robot
- the nature of the sensor information
- the additional source of information from which observations can be made

1.4 Laser Scanners

Light Detection and Ranging (LIDAR) is a technology for measuring the distance to a target by illuminating it with light, and more specifically pulses of light from a laser. LIDAR applications include atmospheric physics, forestry, geomorphology, geology, geography, archaeology, seismology, geomatics and of course robotics.

LIDAR uses visible, ultraviolet or near infrared light to illuminate and image targets. The most effective targets for illumination are non-metallic objects, rain, rocks, clouds and even so small a individual molecules. LIDAR provides very high resolution imagery due to its narrow laser beam.

In Figure 1.12 There are several major components to a LIDAR system:

- Laser - for non-scientific and safety purposes, lasers with a wavelength of 600 - 1000nm are commonly used.
- Scanner and optics - dictates the speed at which images are collected. There are a number of options for collection:
  - polygon mirror
  - dual oscillating plane mirrors
  - dual axis scanner

The angular resolution and the detectable range are affected by the optical choices.
1.4 Laser Scanners

Figure 1.11: Image of LIDAR equipped mobile robot (Courtesy of Wikipedia)

- Photodetector and signal processing - There are two main technologies used in LIDARs:
  - solid stage photodetectors
  - photomultipliers

The overall sensitivity of the detector is another factor that determines the performance of a LIDAR.

- Position and navigation systems - When mounted on mobile applications, LIDARs require additional information about their absolute position and orientation. The nature of these positioning and orientation sensing systems depends on the application. For example, if deployed on a plane additional sensors could be a GPS unit and IMU.

In the scope of these thesis, LIDAR technology is being applied to a mobile robotic platform. In robot applications LIDAR technology is used to perceive the environment as well as classify objects. In Figure 1.11 a LIDAR unit mounted on a mobile outdoor robot can be seen.
Figure 1.12: Illustration of LIDAR operation
Chapter 2

Related Work

“A creative man is motivated by the desire to achieve, not by the desire to beat others.”
- Ayn Rand

This chapter reviews some of the work done in the area of ANNs and MRL. The first section outlines some of the reasons stated by many in the area for the choice of Artificial Neural Network Localization (ANNL). The second section introduces approaches featuring supervised learning. The third section introduces approaches featuring unsupervised learning. The choice of paradigm is one of the key issues in creating a localization based on artificial neural networks. The final section, will deal with how these systems are usually built and simulated.

2.1 Common Motivations

The area of ANNs combined with MRL has been explored extensively during the 1990’s and again in more recent years, each citing various motivations behind their implementations.

Mobile robotics require realtime systems to perform tasks, dynamic motion and deal with hazards effectively. In order to have a realtime system, sufficient memory and computation are required for algorithms to function adequately.
(Won-Seok and Se-Young, 2007), (Hou et al., 2005) and (Dai et al., 2008) were restricted by the available hardware. The reason for this was either that computation at the time of publication wasn’t cheap or the modern systems, such as robot vacuum cleaners, were not equipped with sufficient hardware due to marketability. It is with that in mind that (Won-Seok and Se-Young, 2007), (Hou et al., 2005) and (Dai et al., 2008) take issue with the pressure that classical probabilistic approaches take on their systems and sought to improve that with ANNs.

(Berger and Kleeman, 1992), (Won-Seok and Se-Young, 2007) and (Racz and Dubrawski, 1995), take issue with the complexity of building EKFs for a significant problem like MRL and look to apply more general algorithms like ANNs. EKFs require an estimate of the previous position to be kept before a new estimate is made, an ANN approach may possibly remove the need for that history, an objective of (Townsend et al., 1995).

As stated by (Sethi and Yu, 1990) and (Townsend et al., 1995), an attractive feature of ANNs for MRL is that they avoid the need for extensive map building. ANNs have been shown to allow for lossless data compression (Guowei et al., 2003). In the case of this thesis, and related works, the map is represented by the synaptic weights of the network, resulting in a reduction of the memory and CPU usage. With the removal of the map and update routine, system resources can be freed up (Racz and Dubrawski, 1995). On a related note, it was stated that for the task of scan matching, ANNs are shown to outperform nearest neighbor classifiers at a lower computational cost (Yu et al., 2007).

The main focus of a great deal of the related work has been to make sonar sensors more reliable, meaning ANNs are good at overcoming noisy sensor data. (Sethi and Yu, 1990) states that while ultrasonic sensors are cheap and flexible, they can be somewhat unreliable. The unreliability stems from the errors due to sonar beam shape and specular reflections. The work of (Racz and Dubrawski, 1995), (Oore et al., 1997), (Yun et al., 1998) and (Sang-Hyuk et al., 2007) suggests that ANNs are better suited at overcoming noisy sensor data.

Commonly the drawback of ANNs for MRL referred to by most researchers is that of the training time, as in the case of (Berger and Kleeman, 1992). However, it is shown by (Janet et al., 1996), (Janet et al., 1995b), (Janet et al., 1995a) and (Janet, 1995), that previously trained neural networks can be expanded
2.2 Supervised Approach

The input-output pair features previously trained on a network, form a foundation with which additional pairs can be quickly incorporated. This is opposed to retraining a new network with a newly compiled list of input-output pairs.

2.2 Supervised Approach

When applying ANNs to MRL, the popular architecture is to have a FFNN learn the mapping of a sonar or laser scan to that of a robot pose. A number of researchers have pursued this approach, with varying degrees of success or different aspects of MRL.

The earliest work on this area was performed by (Sethi and Yu, 1990). While noticing that previous work mainly focused on constrained matching, (Sethi and Yu, 1990) chose to take the ANN approach, cast the problem as a regression one and solved it with a multi-layer FFNN. Aside from using back propagation to optimize their networks, (Sethi and Yu, 1990) use an entropy net model, featuring a tree to network the mapping, resulting in a network of the appropriate size. (Sethi and Yu, 1990) found the initial results of their work promising, however did not pursue future work.

Differing from the previous work of (Sethi and Yu, 1990), (Berger and Kleeman, 1992) developed an active beacon MRL system. (Berger and Kleeman, 1992) proposed the CEINT algorithm, which adapts its memory requirements to cope with the complexity of the problem. In this case, the CEINT is taught to track a mobile robot’s position. This is accomplished with six beacons transmitting a pattern of ultrasonic pulses. The mobile robot logs these pulses, and is trained to interpret the pulse arrival times in terms of its own location. (Berger and Kleeman, 1992) concluded that their implementation is successful, but could have worked more effectively with more extensive training data.

(Racz and Dubrawski, 1995) look into the fine positioning of a robot around a local object (an open door) with the goal of more effective door passing algorithms. The (Racz and Dubrawski, 1995) proposed approach looked to learn the direct mapping between ultrasonic range sensor values on the robot and its own pose in relation to an open door. (Racz and Dubrawski, 1995) looked to
accomplish this using the Fuzzy-ARTMAP neural network (Carpenter et al., 1992). The Fuzzy-ARTMAP allows for an incremental and supervised learning to find the relation between ultrasonic sensor outputs and the robot’s pose. (Racz and Dubrawski, 1995) concluded that while the Fuzzy-ARTMAP could learn the needed associations, the error rate was not small enough to allow for the fine positioning.

(Janet et al., 1995b) propose a tool for applying a RFNN to MRL. The RFNN tool allows for a flexible, multi-layered FFNN architecture as well as the ability to optimize the size of the network by adding or pruning nodes. They explore four different architectures offered by this tool, and apply them to the MRL problem, evaluating which is the best suited. (Janet et al., 1995b) discovered that a classification of 98.89% was achieved, while the overall error difference between all four architectures was nominal. (Janet et al., 1995b) concluded that while a mapping can be effectively learned, the addition of new furniture or the rearrangement of a learned room would significantly change how the system interpreted the room. This would then require the room to be relearned. (Janet et al., 1995b) suggest an update learning scheme to detect changes in a previously trained room.

(Townsend et al., 1995) present what is the most rigorous evaluation of the performance of ANNs for the estimation of pose from sensor data for the related works. RBFNs and MLPs are trained to estimate the physical position of a mobile robot using laser range data. (Townsend et al., 1995) perform the first work with laser range scanners and ANNs for the purposes of MRL. They perform an evaluation on RBFNs and MLP implementations, and come to the conclusion that RBFNs perform the mapping task better, while also simplifying training stages. (Townsend et al., 1995) advise that additional work should be performed to find a better input representation for the scans, one which is less effected by input segmentation errors. In addition to this, (Townsend et al., 1995) advise that larger numbers of training patterns be collected in areas were there is a large variation in features.

Like with (Townsend et al., 1995), RBFNs feature heavily in the work of (Scolari et al., 2003). However, their work focuses more on a recommended strategy for using arc:mmnt! to perform MRL. Work is performed in simulation only, position values are estimated from an input-output mapping collected at ran-
dom from a virtual room. RBFNs are able to approximate any multivariate continuous function satisfactorily well as indicated by the work of (Poggio and Girosi, 1990). With RBFNs, (Scolari et al., 2003) state that an acceptable level of error for position approximation was found in indoor applications.

(Sang-Hyuk et al., 2007) present a pose estimate from ultrasonic sensors for autonomous domestic vacuum cleaners using MLPs. The motivation of the ANN approach is driven by their goal of a cheap marketable MRL easily deployable in domestic environments. Relying on cheap and ambiguous sonar sensors has led them to turn to ANNs as a means to find fault tolerance; a quality also emphasized by (Townsend et al., 1995). While previous work looked to represent the map with an ANN, (Sang-Hyuk et al., 2007) makes use of a ANN to estimate the distance and angle to a concave feature like a corner. The corner feature then being compared to a classical map. (Sang-Hyuk et al., 2007) found unreliable results with their implementation and recommended combining it with an EKF. (Won-Seok and Se-Young, 2007) pursue their MRL method featuring a modular MLP. The modularity in this case refers to the use of a single MLP for each of the pose values. (Won-Seok and Se-Young, 2007) state that learning time is lengthy using MLPs, a conclusion also reached by (Townsend et al., 1995). (Won-Seok and Se-Young, 2007) ignore the lengthy training times and suggest that their MRL method is effective, and recommend it for indoor use on computationally constrained units.

(Conforth and Meng, 2008) proposed the SWIRL algorithm, one which performs MRL using FFNNs, with network optimization coming from biologically inspired solutions. ACO (Colorni et al., 1991) is used to choose the ANN topology, and PSO (Clerc and Kennedy, 2002) is used to adjust the weights of the selected topology. ACO’s choice of topology boils down to the simple selection of the number of hidden neurons. (Conforth and Meng, 2008) look to demonstrate that the SWIRL algorithm can replace a classical ML. Experimental results indicate that the SWIRL algorithm can function comparably well to that of a Markov localization. (Conforth and Meng, 2008) claim that this method isn’t limited to MRL and is scalable, robust and can be applied to almost any real-world task.

The present work suggests that the supervised approach allows for sufficiently accurate MRL. With this in mind, the inclusion of the supervised approach has
been featured in the MRL algorithm of this thesis.

2.3 Unsupervised Approaches

While the supervised approach to this area is the most common one, others have investigated unsupervised approaches, such as SONNs, which are typically applied to clustering problems.

Janet et al. [1995a] present the first instance of SONNs applied to MRL. Janet et al. [1995a] use the Kohonen neural network as it can perform data compression, recognition, classification and association in an unsupervised manner. Janet et al. [1995a] approach the problem as an Optical Character Recognition (OCR) one. Their robot features a ring of sonar sensors, each collected distance measurements around the circumference of the robot. If displayed in a 2D manner it would appear as character-like shape. It is the goal of Janet et al. [1995a] to recognize which room a robot is in from a possible 10. A constraint on their approach is that they need to specify beforehand what is the maximum expected number of features, meaning scaling to a larger number of rooms would mean a complete retraining of the network. Janet et al. [1995b] stated that in the case of any addition or rearrangement of furniture, the network for that room would have to be completely retrained. With a Kohonen implementation Janet et al. [1995a] are claiming the same, and reiterate the need for an update learning scheme.

Yu et al. [2007] apply SONNs to process 2D laser scans for feature extraction, in this case a concave corner. Yu et al. [2007] attempted to combine occupancy grid matching with SONNs in order to reduce computational cost. The experimental results indicated that they could indeed reliably extract corner features and improve the overall performance of the localization. However, this approach purely applies the SONN to feature recognition and does not does not map to a location classification.

Dai et al. [2008] attempt to solve the SLAM problem using purely fuzzy logic, clustering algorithms and ANNs. Specifically, they adopt a Self-organizing Fuzzy Neural Network (SOFNN) Leng et al. [2004] for the MRL. Dai et al. [2008] claim that the use of SOFNN allows for the construction of a map on-
line. The SOFNN is used for the local clustering of scans to cells in a grid map allowing for the correction of odometry values. (Dai et al., 2008) describe their work as immature and promising.

The combined work of (Janet et al., 1995a), (Yu et al., 2007) and (Dai et al., 2008) is the sum total of work using SOMs as a means for MRL. With that in mind, this would lead to the conclusion that while the work in this area is promising, it is still in a very early stage. This immaturity results in the unsupervised approach not being considered for the localization algorithm in this thesis.

2.4 Simulations

Simulators are used extensively in this area of research, many being developed in-house. The reasons behind performing experimentation in a simulated environment are featured in the list below.

- To avoid damage or wear to robotic units and the environment
- To allow for rapid and accurate reproducibility
  - A real robot might not always travel the same route
  - Repetitive placement of a robot by an operator might lack precision
  - Remove tedious repetitive placement for an operator
- To allow for fast collection of datasets with varying levels of detail
  - For initial testing, a dataset consisting of 10 scans might only be needed, while extensive testing might need to be run with 10000 scans
- Avoid the possible delay for charging of robotic units
- The ground truth is available for free

The examples of simulated environments from the cited related works is small while the reasoning behind their choice was guided by many of the above reasons.
2.4 Simulations

(Sethi and Yu 1990) use a purpose built simulator to provide ultrasonic sensor data at arbitrary points throughout their map. Figure 2.1 shows the typical display of the (Sethi and Yu 1990) simulator, showing map, robot orientation as well as sonar beam directions.

Figure 2.1: Simulator of (Sethi and Yu 1990)

(Sethi and Yu 1990) collect sonar contours at random locations around the maps. Tests occur with rooms of differing configuration. An example of the room configurations are in Figure 2.2.

Figure 2.2: Room configurations of (Sethi and Yu 1990)

While (Racz and Dubrawski 1995) collect data using a real robot, they also produce data using the Robuter mobile platform simulator. The Robuter simulator was developed at LIFIA in France (Crowley et al. 1991).
2.4 Simulations

Dubrawski (1995) state that performing trials with a simulator make it possible to easily divide the map into subspaces increasing the effectiveness of the algorithm and allowing for easy adjustment of the network parameters. The (Crowley et al., 1991) Robuter simulator is no longer in development.

(Oore et al., 1997) use simulated data under the assumption that the robot’s orientation is known. (Oore et al., 1997) do not develop the simulator but instead use one developed by (Wilkes et al., 1990). The choice is made because the simulator allows for many types of typical non-gaussian noise experienced with sonar sensors. Figure 2.3 from (Wilkes et al., 1990) displays a simulated environment.

In Figure 2.3 the position of the robot is indicated by a circle near the center of the room. The surfaces that are representative of reflective ones are shown as thicker lines. While the thin lines indicate the direction of the sonar sensors. Work on the (Wilkes et al., 1990) simulator has long since been discontinued.

The (Townsend et al., 1995) approach to experimentation with both real and simulated range data, states a need for training data that is evenly distributed throughout the environment and accurate to the position in the environment. (Townsend et al., 1995) cite the problem of getting the ground truth for collected data from a real robot along the trajectory travelled, and state a preference for simulated data because of this. With the simulator of (Townsend et al., 1995), 10,000 training pairs are generated from randomly distributed points in the environment. Figure 2.4 shows the distribution of points. Notice that no points were taken within the robot circumference from the wall.
2.4 Simulations

MATLAB is widely accepted in academic, scientific and industrial settings. (Scolari et al., 2003) recommend a strategy for applying MATLAB to localization. While the algorithm implementation was straightforward, the demonstrated the ability to gather training and validation points from a given 2D map. Figure 2.5 shows their training point distribution.

While there have been a number of simulators applied to this area, many are very much outdated. The work of (Scolari et al., 2003) suggests the use of arc:mnnt!, however an investigation of their environment simulation setup was not satisfactory. The algorithm featured in this thesis will rely on the arc:mnnt! for the training and validation of the ANN component and an al-
ternative will have to be found for the control and visualization of the robot’s pose and trajectory in real-time.
Chapter 3

Implementation & Simulation

“I can, therefore I am.”
- Simone Weil

This chapter summarizes the work that the author has performed initially over a one month period. The work consisted of setting up a mobile robot simulator for simulating the exploration of an environment by a LIDAR equipped robot in general, a lot of effort has been put into providing a realistic simulation. The work also consisted of setting up a satisfactory ANN library for training and realtime execution, then linking that ANN to the ROS simulator. In addition this, an extensive number of datasets for training had to be collected for ANN training, and a number of tools written in C++ were produced. The simulator is based on a ROS variant of the Stage robot simulator, called stageros. Stageros is written in C++ and is entirely open source. The Fast Artificial Neural Network (FANN) is used as the ANN library. All components of ANNL were written in C++ and comply with the ROS framework.

3.1 Overview

A major attempt has been to keep the application of the simulator as simple as possible. It can be configured to reflect any robot, this can be essential when dealing with robots that have different LIDAR sensors equipped. In this thesis work, the simulator is used to simulate a single robot in a closed indoor...
environment. The robot is equipped with a laser scanner, that provides 270 distance measurements over a field of view (FOV) of 270°. Autonomous control over the simulated robot is possible, allowing for repetition. The simulator can be setup to communicate directly with the ANN architecture that is running with FANN where FANN can have access to the current laser data. ROS nodes send motion commands to the robot, get the ground truth position as well as a simulated MCL estimation of position. While there is no GUI for the overall system, both the ROS simulator and FANN library provide easy to use interfaces for control, configuration and data gathering. The ROS simulator allows for real-time simulation, giving the possibility of testing an ANN’s performance in realistic time intervals. Complete recording of the simulation environment for later playback is also possible. Data collection is handled by a number of C++ tools written by the author, this allowed for scripting and expansion of the ROS simulator functionality.

3.2 ROS Simulator

3.2.1 Robot Operating System

The Robot Operating System (ROS) (Quigley et al., 2009) is a software framework to facilitate the creation of robot applications. ROS offers hardware abstraction, libraries, device drivers, message-passing, visualizers, package management and is continually being added to. ROS is released under the open source BSD license by Willow Garage and has become a de facto standard software tool in Robotics research, with applications starting to emerge within industry.

ROS was introduced to address a major issue in robotics, code reuse. This is a non-trivial problem due the varying of hardware from robot to robot. ROS is referred to as an operating system, but is not in the traditional sense. ROS provides a structured communication layer instead of process management and scheduling. ROS sits as a layer above the heterogeneous layer of operating systems and hardware implementations to provide transferable applications. The implementation is based on a graph architecture, within which each node of the graph presents a service. With message passing, the nodes then will be
able to receive, post and multiplex sensor, control, planning, state and actuator messages. One of the many strong points of ROS is its hardware abstraction.

### 3.2.2 Simulator Features

The Stage simulator allows for the simulation of a population of mobile robots, complete with sensors and environmental objects that are represented in a 2-D bitmapped environment. Stage was designed with the intention of supporting research into multi-agent autonomous systems. Stage avoids the large overhead that might be encountered with high fidelity robot models with relatively simple and computationally cheap models, allowing for many robots to be simulated easily. A basic simulation environment can be easily scaled to model one to hundreds of robots at once. Users can define control programs that allow Stage to easily simulate the resultant robot behavior. The Fast Light Toolkit (FLTK) is used as a base for the Stage simulation environment. Typically Stage is interfaced with Player, this allows for access to simulated devices and sensors, however for this thesis this is not the case and an alternative is used.

![Figure 3.1: Stage simulation of multiple robots (Courtesy of Brian Gerkey)](image)

A C++ library has been made available to allow individuals to produce their own programs, this library is called libstage. The value of this is if Player does not provide the functionality needed, or custom simulation models want to be developed with a reliable and well-known simulation engine. Player however was not chosen for this thesis because the author wished to become more familiar with ROS and its related tools.
stageros provides only a subset of Stage’s functionality, however is more than sufficient for the nature of this thesis. stageros provides its functionality via the previously mentioned libstage. In an simple example, if a single robot with differential drive and a laser scanner where described within the simulated world, then both the odometry data of the wheels and the laser data from the scanner would appear as topics available to be published or subscribed to.

3.2.3 Simulator Configuration

As was stated in the previous section stageros is a wrapper for the Stage 2-D simulator, and it does this using libstage. The stageros node only provides a subset of the original Stage’s functionality. In order to simulate a world in stageros a .world file has to be defined. From this .world file stage has a full description of the world, this includes obstacles (bitmap), robots and other objects. With stageros it should find at least one laser and a position model for a robot, if it doesn’t then it just exits.

In this thesis there is only ever one robot being simulated at a time, this means that all topics related to that robot appear at the top of the namespace. The following topics is generated by the simulator:

- cmd_vel (geometry_msgs/Twist)
  - allows for the sending of commands to differentially drive the robot
- odom (nav_msgs/Odometry)
  - odometry data from the robot (usually just the ground truth)
- base_scan (sensor_msgs/LaserScan)
  - scans from the robot’s laser
- base_pose_ground_truth (nav_msgs/Odometry)
  - ground truth pose of the robot

The stageros provides a odom topic, and this topic provides simulated odometry values from the robot. This odom topic can be easily configured in the .world file
so to allow for changes in the origin and also the noise models. The *base_pose_ground_truth* always provides perfect global pose information for the robot. It is purely meant to be used for testing purposes. If *odom*’s configuration is left at default, the output is identical to *base_pose_ground_truth*’s.

As was previously stated the environment itself is described using a bitmap, and in the .world file this is loaded with the following.

```xml
floorplan
(
    name "lab_maze"
    size [4.000 3.000 1.000]
    pose [-2 1.5 0 0.0]
    bitmap "./maps/arena.pgm"
)
```

As it can been seen, a reference is made to "arena.pgm" and this can be seen in Figure 3.2.

![Bitmap arena used during simulation](image)

**Figure 3.2:** Bitmap arena used during simulation

### 3.2.4 *stageros* Execution

On execution of *stageros* the user is presented with a live map of the environment, the pose of the robot and a outline of the laser data. Figure 3.3 shows the simulation in action.
3.2 ROS Simulator

Figure 3.3: *stageros* simulation in operation

*rxgraph* displays a visualization of a ROS Computation Graph, i.e. the ROS nodes that are currently running, as well as the ROS topics that connect them.

In order to evaluate during runtime the running nodes in the ROS system, a tool called *rxgraph* is used. *rxgraph* displays a graph of ROS computation, this means, the nodes that are currently running and also all the topics that connect them together. Figure 3.4 depicts ROS’s state while the *stageros* is running.
Figure 3.4: rggraph overview of running ROS nodes
3.3 Fast Artificial Neural Network Library

The FANN library (FANN) provides tools implementing, designing, visualizing and simulating ANNs. FANN provides support for FFNNs, RBFNs, SOMs, dynamic networks and a number of other network paradigms.

3.3.1 FANN Features

The following outlines the features of the FANN library:

- Multilayer Artificial Neural Network Library in C
- Backpropagation training
  - RPROP
  - QuickProp
  - Batch
  - Incremental
- Easy to use
  - creation, training and execution ANNs with just three function calls
- Fast
  - up to 150 times faster execution than other libraries
- Versatile
  - possible to adjust many parameters and features on-the-fly
- Well documented
- Several different activation functions implemented
- Easy to save and load entire ANNs
- Graphical Interfaces
FANN provides many commandline functions and visualization tools for creating, training and simulating ANNs. With the availability of visualization tools, like FANNTool, make it much easier to develop ANNs and applying them to common tasks like curve fitting, pattern recognition, and clustering. After using the provided tools to create an ANN, FANN allows for easy integration with any ROS code.

### 3.3.2 FANNTool

FANNTool is a GUI to the FANN library which allows ease of use without the need to program the function calls necessary to perform simple tasks. FANNTool is an open source project, supported by a community of FANN users. A screenshot of the FANNTool GUI can be seen in Figure 3.5.

To give an overview of the key features of FANNTool:

- prepare the data in FANN library standard
- the design of an ANN
- the training of the designed ANN
- the testing of the trained ANN
- and then the running the trained ANN

Figure 3.5 displays the options available to the user of FANNTool, allowing for the modification of hidden layer size, training methods, activation function on each layer, the number of training epochs and the stopping function. At the bottom the log of the training activity generated can be seen.

### 3.3.3 Integration with ROS

The development revealed the need for a bridge between ANNs and the robotic systems, either simulated or physically. While a number of options are available for ANNs libraries, each offered its own complications in making the the...
available in realtime to the sensors on the robot, and sending control signals back. The first version of the localization used the MATLAB neural network toolbox, but it did not have a direct interface with ROS, it relied heavily on a interprocess communication library that is for the most part deprecated and provided unnecessary functionality. Ideally, ANNL is light, versatile and easy to use. The inclusion of MATLAB Neural Network Toolbox would not allow for this. FANN provided a very simple library to access the functionality, the ROS nodes of this thesis are written in C++, while FANN is written in C, making its inclusion in the software very straightforward.

3.4 Cartesian Coordinates

The experimentation with cartesian coordinates requires the ability to gather the ground truth of the robot in a space. As previously stated, using a simulator to gather scans and location data seems more practical in the rapid collection and replication of experiments. With a simulator, the ground truth comes for
3.4 Cartesian Coordinates

ROS publishes a number topics that provide a 270 point scan, x, y and rotation values, concurrently at a rate of once per second. Topic publication rates are adjusted accordingly for the problem, refresh rate of higher than 1Hz seems excessive. All topics are collected using rosbag. rosbag is a set of tools for the recording of and playing back of ROS topics. After recording, topics are played back out of a rosbag.

To facilitate in the training of the ANN during experimentation, a framework is set-up. This framework consists of 5 stages: generating coordinate-based training data; generating scan-based training data; coalescing data-sets; training the ANN against the coalesced data-set; and testing the effectiveness of the ANN.

As several of the experiments require some complex pre-processing, coordinate-based training data is often generated from a C++ program. Depending on the experiment, a Java program is written to generate mathematically ideal outputs for every given input. The results of these are written to a space-separated flat-file.

Scan-based training data is generated by running ROS against the output of the coordinate-based training data, creating an associated scan for every coordinate and orientation in that data. To do this, it is necessary to be able to autonomously put ROS into specific configuration—that is, place the robot at an exact coordinate and orientation—and then take a scan, before moving on to the next configuration. It is noted that to move the robot in ROS, three options are available:

1. the robot can be "picked up" and "dropped" at another location by the user dragging the robot with a pointing-device

2. the robot’s position and orientation can be initialised during the ROS start-up in a configuration file

3. the robot can be "walked" to a position by several relative movement commands, including: move forward; move backward; turn left; and turn right. This can be controlled manually or automatically. Unlike the other two movement options, this does not allow the robot to pass through a wall.
In choosing an option to generate the scan-based training data, the first option is dropped as the number of times the robot would have to be moved is prohibitively large for a manual process. The third option is also dropped as, although it can be automated, it has two major drawbacks, these are:

1. the process that walks the robot must be able to do pathfinding to navigate around the various locations, requiring an entire robot navigation system to be set-up

2. due to the step size of the relative movements, it becomes difficult to place the robot on the ideal positions generated for the coordinate-based training data, adding a certain amount of error to the results

This just leaves the second option—starting ROS with the exact coordinates and orientation set in a configuration file. To do this, a script is created. This script takes an input file, this being the output of the coordinate-based training data, and for every line in this file, performing the following steps:

- loads a template configuration file with placeholders for the x and y coordinates, and orientation
- creates a copy of the configuration file replacing the placeholders with the coordinates and orientation of the current line from the coordinate-based training data
- runs ROS (and related processes) redirecting its output to a unique file (incrementing a file-number for each line)
- waits several seconds to allow ROS to initialise and start scanning
- force-terminates ROS (and related processes)

This results in one file being created for every line in the coordinate-based training data. As ROS is running for several seconds to allow scanning, each file generated from it has multiple scans.

All the files are then coalesced. To do this, another Java program is created. This Java program goes through all the lines of the coordinate-based training
data, reads the fourth line (skipping over any lines that may have been affected by start-up errors) of the related scan-based training data, and writes them to a new file where each line contains the related coordinate- and scan-based training data.

To validate the performance of the ANN after training, the final testing stage is done. To do this, the previous 4 steps are repeated with slight modifications. The original C++ program for generating coordinate-based training data is modified to generate results for random locations across the map, creating a collection of ideal results for locations off of the training data. Scan-based training data is generated from this and the results are coalesced in exactly the same way as above. The results are then fed into MATLAB and run as testing data through the saved ANN. The results of this are then analysed to see a high-level view of how well the ANN performed and get an idea of what parts of the map require more training. Real-time testing is also performed using the FANN-ROS interface to confirm the results as often the emergent behaviour of the robot using the ANN is markedly greater than the instantaneous view of single points.

Figure 3.6: Screenshot of the Stage configuration for Rotation Experiments
3.4 Cartesian Coordinates

3.4.1 Rotation

For the rotation experiments, the arena is setup to place the robot at the very center of the arena. This positioning allows for good arena visibility as the robot rotates about the one point. In order to do this a .world file describing all this is written and included with the run. In Figure 3.6 the placement as well as the scan FOV can be seen. While Figure 3.6 depicts a single robot for testing the ANN in realtime, a different .world is generated for the training data collection. For reasons that are stated in Section 4.1.1 scans are also collected from 20 or so directions at once, in this case 20 robots are included in the .world file each having different directions. A custom ROS simply collects the scans from each robot afterwards.

Figure 3.7 provides an overview of the live system for rotation experiments outlined in Section 4.1.1. The stageros node, is at the root of the experiment execution, displaying the position data of the robot, as well as the FOV of the laser scanner and displaying the map. The neural_odom node is where the execution of the ANN occurs, and it can be seen that it accepts a laser scan of the name of base_scan. The move_base node is responsible for handling autonomous navigation of the robot, however its functionality is not taken advantage of during the rotation experiments. The turtlebot_teleop_keyboard node allows for the setting of angular and linear velocities and the manual/keyboard control of the robot in the simulation. The odometry_publisher node handles coordinate system transformations. The amcl node provides MCL but as move_base is not taken advantage of here. The rosout node is a channel for warning and error messages, all nodes publish there.
Figure 3.7: Graph of the ROS Nodes running during Rotation Experiments
3.4 Cartesian Coordinates

3.4.2 1-D

For the 1-D experiments, the arena is setup to place the robot at the far left center of the arena. This positioning allows for the robot to travel the length of the arena without obstacles. In order to do this a .world file describing all this is written and included with the run. In Figure 3.8 the placement as well as the scan FOV can be seen. While Figure 3.8 depicts a single robot for testing the ANN in realtime, a different .world is generated for the training data collection. For reasons that are stated in Section 4.1.2 scans are also collected from 20 angles at 5 positions, in this case 100 robots are included in the .world file each having different directions and positions. A custom ROS simply collects the scans from each robot afterwards.

![Stage Configuration](image)

Figure 3.8: Screenshot of the Stage configuration for 1-D + Rotation Experiments

Figure 3.9 provides an overview of the live system for 1-D experiments outlined in Section 4.1.2. The stageros node, is at the root of the experiment execution, displaying the position data of the robot, as well as the FOV of the laser scanner and displaying the map. The neural_odom node is where the execution of the x-axis ANN occurs, and it can be seen that it accepts a laser scan of the name of base_scan, estimating the position. By extension of neural_odom, there are a number of nodes with the prefix of neural_rotat, each of which also
accepts *base_scan* and has an associated ANN. Each of the *neural_rotat* nodes performs the rotation estimation, for their corresponding positions. All *neural_rotat* nodes pass their results to the *neural_odom* node, which in turn will combine the rotation estimation with the position estimation. The *move_base* node is responsible for handling autonomous navigation of the robot, however its functionality is not taken advantage of during the rotation experiments. The *turtlebot_teleop_keyboard* node allows for the setting of angular and linear velocities and the manual/keyboard control of the robot in the simulation. The *odometry_publisher* node handles coordinate system transformations. The *amcl* node provides MCL, but as *move_base* is not taken advantage of here. The *rosout* node is a channel for warning and error messages, all nodes publish there.
Figure 3.9: Graph of the ROS Nodes running during 1-D + Rotation Experiments
3.5 Landmark Recognition

3.4.3 2-D

For the 2-D experiments, the arena is setup to place the robot at the far left center of the arena. This positioning allows for the robot to travel the length of the arena without obstacles, identical to the 1-D experiments outlined in Section 3.4.2. In order to do this a .world file describing all this is written and included with the run. In Figure 3.8 the placement as well as the scan FOV can be seen, it is an identical configuration as the 1-D experiments. While Figure 3.8 depicts a single robot for testing the ANN in realtime, a different .world is generated for the training data collection. For reasons that are stated in Section ?? scans are also collected from 20 angles at 25 positions, in this case 500 robots are included in the .world file each having different directions and positions. A custom ROS simply collects the scans from each robot afterwards.

The inclusion of an rxgraph screenshot is not made, as the configuration of the software was updated to allow for better management of switching between the 7 axis and 25 rotation networks. During runtime, the ROS environment would look the same as Figure 3.7.

3.5 Landmark Recognition

The next stage of experimentation required a move away from the use of cartesian coordinates, despite this, the underlying implementation stays rather the same. As with the cartesian coordinate experiments, the FANN library provides the backbone of the landmark recognition algorithm. The Stage simulator is used to gather scans for training ANNs, and later the simulator is used to collect location data, testing the performance of ANNL and easy replication of experiments. ROS publishes a number topics that provide a 270 point scan, x, y and rotation values, concurrently at a rate of once per second. All topics are collected using rosbag. After recording, topics are played back out of a rosbag into corresponding training sets or for performance comparison.
Figure 3.10: Graph of the ROS Nodes running during Landmark Recognition and Path-finding Experiments
3.5 Landmark Recognition

3.5.1 Spiral Path-finding

Figure 4.20 depicts the autonomous path-finding of a mobile robot using purely landmark recognition. There is no coordinate system used. The robot follows a simple spiral pattern and wall following algorithms until a recognized landmark is found. As it can be seen, there are only 4 trained landmarks throughout the map.

3.5.2 Final Path-finding

The path-finding experiments outlined in Section 4.3 required little modification from the implementation outlined in Section 3.5 for the landmark recognition experiments in Section 4.2. Figure 3.10 provides an overview of the live system for Path-finding experiments outlined in Section 4.3. The `stagers` node, is at the root of the experiment execution, displaying the position data of the robot, as well as the FOV of the laser scanner and displaying the map. In previous implementations, like in Figure 3.9, there would be a `neural_odom` node. Within this node the execution of the axes and rotation ANNs occurs,
however in order to streamline the implementation, that functionality, as well as the driving algorithms are incorporated into a single node, now called `path_finding`, and it can be seen that it accepts a laser scan of the name of `base_scan`, estimating the position. `path_finding` publishes to the `cmd_vel` topic allowing for the modification of the robot’s angular and linear velocities. The `rosout` node is a channel for warning and error messages, all nodes publish there.
Chapter 4

Experimentation

“The harder the conflict, the more glorious the triumph.”
- Thomas Paine

4.1 Cartesian Coordinates

To gauge the effectiveness of ANNs on localization, several experiments are carried out, each increasing the complexity of and reflecting different aspects of the problem. Some of these experiments duplicate to a certain extent work done by (Sethi and Yu, 1990) and (Racz and Dubrawski, 1995), but are implemented differently and being performed to get a deeper understanding of how ANNs can be used to solve localization problems, and to guide further experiments.

To form the building blocks of the system, naturally the most basic parts are investigated first. This means experimenting with rotation estimation with and without items present, expanding upon this with 1-D localization, finding a position along the x-axis for example. 1-D localization is expanded upon by adding rotation estimation along the x-axis of the arena. 1-D localization is then expanded upon again by added the y-axis, providing x and y position estimations. Finally, 2-D localization is expanded upon with rotation estimation.
4.1 Cartesian Coordinates

4.1.1 Rotation

The starting point for experimentation is the rotation angle estimation. Initially, it is not known what are the ideal number of scans to be taken for each direction, how far those scans need to be spaced apart and the effect of symmetry in the environment has on the network estimation. Those aspects are revealed with the following experiments. The first, explores rotation without any items in the room, the second, explores rotation with two items in the room. In each case the following are outlined, arena features, scan collection methods, AF and training method selection criteria, network training and a live demonstration.

Without Items

A rectangular arena void of all items is chosen to start rotation experimentation. It is expected that the shape of the arena along with the lack of items within, will present a symmetric environment, one which will likely make direction estimation difficult. The experiment is conducted with a single robot, placed at the centre of the arena. Scans are then collected at rotated intervals at the centre point, then training is performed. The following outlines the details of said experiment.

Figure 4.1 presents the arena. A rectangular arena, the length and width of which was chosen arbitrarily. The robot itself has a FOV of $270^\circ$ with a range of 20 meters. Due to the area of the arena and the range of the laser, a complete contour is available to the robot wherever it travels, possibly providing a unique contour for each position.

The implementation of this approach relies on the collection of a number of scans from each direction, as the collection of a single scan is not adequate for the ANN to generalize about that direction. The training data will consist of input and output vectors and this will be referred to as an input-output pair. Through trial and error, it was deemed that 5 scans each offset from the other by $1^\circ$ provided sufficient samples for ANN generalization, for one particular direction. For this implementation, four main directions are selected to simplify the problem, each consisting of 5 scans, resulting in a total of 20 scans. In Figure
4.1 Cartesian Coordinates

Figure 4.1: Simulated Arena Vacant of Items for Rotation Experiments

4.2 The distribution of the directions to which each scan is taken can be seen, with the main direction centered for each sample set.

The search for the correct AF for both the hidden and output layers, requires the review of 13 different AFs. Each AF is evaluated against all others, training for 2000 epochs. When the MSE drops below a predefined threshold, training stops. The combination with the fewest number of epochs during training is then selected for the rotation network.

Table A.4 lists the available AFs. The Linear in Figure 1.3 and the Symmetric-sigmoidal in Figure 1.4 have performed the most efficiently out of all of the combinations. The Linear AF being applied to the hidden layer, while the Symmetric-sigmoidal is applied to the output layer. An overview of the resulting MSEs for each combination of AF can be seen in Table A.7.

Using the AFs selected in Section 4.1.1, the search for the correct training method is performed, requiring the review of 4 different methods. Each training method is evaluated against all the others, for 2000 epochs. When the MSE drops below a predefined threshold, training stops. The training method with the fewest number of epochs during training is then selected for the rotation network. The available training methods are described in Section 1.2.4 and listed in Table A.1, they are Incremental, batch, Rprop and Quickprop. Table
4.1 Cartesian Coordinates

Figure 4.2: Scan Collection Directions for Rotation Experiments

Table A.1 gives an cross-comparison of the resulting MSEs, which clearly indicates the Rprop as the best training method.

After reviewing the training methods in Section 4.1.1, the AFs in Section 4.1.1 and processing the simulated data, training of the network can begin. Table ?? lists the configuration details.

Figure 4.3 depicts the training progress. What should be expected is a drop to zero MSE, however the network never settles into a minimum such that a direct mapping between the scans and directions is obvious.

It is clear that the reason for the inability of the network to distinguish between directions is due to each direction looking very similar. The robot could be facing the opposite direction and confuse the position for the mirror one on the other side of the arena, as in Figure 1.8. It is for these reasons, the inclusion of items is experimented in the next section.

With Items

In the previous it is concluded that the presence of a symmetric environment makes it impractical to find a mapping between scans and direction. It is with this in mind that the arena is altered, to include irregularities, two items. The
4.1 Cartesian Coordinates

Training Method: Rprop
Hidden Activation Function: Elliot
Output Activation Function: Elliot-Symmetric
Number of Inputs: 270
Number of Outputs: 4
Number of Hidden: 3
Desired MSE: 0.00001
Learning Rate: 0.7
Number of Epochs: 50000

Table 4.1: ANN Configuration and Training Values for Simulation without Items

Figure 4.3: MSE During Training of Rotation Network without Items Present
resulting configuration is to provide a unique scan contour for each direction. The experiment is conducted with a single robot, placed at the centre of the arena. This experiment essentially recreates the previous described in without items experiment. The following outlines the details of said experiment.

Figure 4.4 presents the arena. The circle in the top left appears as a cylinder, and the rectangle in the lower left appers as a elongated cube to the robot. The robot itself has a $\text{FOV}$ of $270^\circ$ with a range of 20 meters, identical to the previous section. The arena configuration is meant to serve as a feature varied environment.

Figure 4.4: Simulated Arena Complete with Items for Rotation Experiments

The training data for this experiment is collected and processed identically to the previous experiment.

The results of the AF selection for both hidden and output layers are found to be identical to those of the previous experiment. They are merely reused for this training session, which means a linear activation on the hidden layer, and a Symmetric-Sigmoidal on the output layer.

The results of the training method selection indicate a change in training method to the previous experiment. The final MSE of each training method is show in Table A.2. The MSE for incremental drops to 0.0 during testing, give the clear indication that it should be chosen for this experiment.
An identical training configuration to that featured in Table ?? is used for this training session.

Figure 4.5 draws comparison between the training of scans with and without items present. It can be clearly seen that the MSE drops to 0.0 before 100 epochs, while without items, the training continues without finding a definite mapping. The results strongly indicate that the network with items present will accurately estimate the rotation of the robot during live testing in the next section.

In order to demonstrate the estimation functioning in a real-time system, the ANN is included ROS node. Output from the ANN and the simulator ground truth are collected simultaneously for comparison.

Figure 4.6 describes the motion of the robot. It starts by facing the right-hand direction, or 180°. It then rotates clockwise, increasing yaw from 180° to 360°. Yaw then drops to 0° and the rotation finishes back at the 180° position. Rotation is performed at an angular velocity of 0.2 m/s.

Figure 4.7 depicts the actual rotation values and the corresponding output from the neural network given the same scan. It can be clearly seen that the rotation follows closely to that of the actual. Smooth overlapping does not occur due to the intervals the rotation has been separated to allow for a simpler configuration while experimenting with ANNs for this purpose.

<table>
<thead>
<tr>
<th>Training Method</th>
<th>Incremental</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden Activation Function</td>
<td>Linear</td>
</tr>
<tr>
<td>Output Activation Function</td>
<td>Sigmoid-Symmetric</td>
</tr>
<tr>
<td>Number of Inputs</td>
<td>270</td>
</tr>
<tr>
<td>Number of Outputs</td>
<td>4</td>
</tr>
<tr>
<td>Number of Hidden</td>
<td>3</td>
</tr>
<tr>
<td>Desired MSE</td>
<td>0.00001</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.7</td>
</tr>
<tr>
<td>Number of Epochs</td>
<td>50000</td>
</tr>
</tbody>
</table>

Table 4.2: ANN Configuration and Training Values for Simulation with Items
Figure 4.5: Comparison of MSE During Training of Rotation Network with and without Items Present

Figure 4.6: Illustrating the Direction of Rotation of the Simulated Robot
4.1.2 1-D

Section 4.1.1 outlined the simplest case of localization, that of rotation about a single point. It follows that rotation along a number of points would be the next step in this thesis. However, before that is possible, localization at a number of distinct positions and then the rotation at each of those positions are seen to be different steps. It is in that case that position estimation at a number of points is investigated first, then using the work from Section 4.1.1 rotation estimation at those positions are handled. The following outlines the arena features, scan collection methods, AF and training method selection criteria, network training and a live demonstration of localization along a single axis.

Figure 4.4 presents the arena. The same arena is used for all experiments into cartesian localization. The robot itself has a FOV of 270° with a range of 20 meters.

A number of training sets have to be collected for the x-axis and rotation networks along that axis. For the x-axis, a mapping between a scan and a binary sequence is needed. The binary output of the network is then transformed into the cartesian coordinates. In the case of this experiment, the collection distribu-
tion is done along the axis of travel, as depicted in Figure 4.9. Scans then need to be collected for each of the rotation networks, four in total, their distribution is depicted in Figure 4.2.

![Figure 4.8: Scan Collection Directions for 1-D Experiments](image)

The selection of the AF is done so in an identical manner to that of the rotation experiments, outlined in Section 4.1.1. The final MSE of each combination can be seen in Table A.7. From the results outlined in Table A.7 it is determined that a symmetric-sigmoidal AF be used on the output layer, and a linear AF on the hidden layer. Identical AFs are used for the rotation networks, the testing for which was done previously in Section 4.1.1 and the results visible in Table A.7.

Using the AFs selected in previous section, the search for the correct training method is performed, requiring the review of 4 different methods. Each training method is evaluated in an identical manner as outlined in Section 4.1.1, the results of which can be seen in Table A.3.

In order to decide the number of hidden neurons, there can be two rules of thumb (Liu et al., 2007). The first way, is to simply take the number of inputs and divide that number by the number of outputs (Equation 4.1). In the case of this experiment that would be 270 hidden neurons. The second way, is to take the number of input-output pairs and divide that by the number of inputs to the network (Equation 4.2). In the case of this experiment that would mean 2 hidden neurons, a computationally less expensive number.

\[
\text{#ofHiddenNeurons} = \frac{\text{#ofInputNeurons}}{\text{#ofOutputNeurons}} \quad (4.1)
\]

\[
\text{#ofHiddenNeurons} = \frac{\text{#ofInputOutputPairs}}{\text{#ofInputNeurons}} \quad (4.2)
\]

From the two options of hidden neuron number selection, the author has decided to select Equation 4.2. This will be the case for all experimentation during this
thesis. As was previously stated, this would result in there being 2 hidden neurons used. This method scales according to the number of input-output pairs and allows for greater flexibility while minimizing the size of the network.

It can be the case with networks of this type that there is more than one hidden layers, however having multiple layers is something that is reserved for special problems, and the standard single layer is usually sufficient for the vast majority of problems.

To represent the error between outputs and targets, the Mean Squared Error (MSE) is used as described by DeGroot and Schervish, 2010. MSE is the average squared difference between the outputs and the targets (Equation 4.3). The lower the MSE value the better the ANN is at reproducing a target given the corresponding input.

\[ MSE(\hat{\theta}) = E[(\hat{\theta} - \theta)^2] \] (4.3)

Table A.2 provides an overview of the configuration of the x-axis network when trained. An increased number of hidden neurons over previous experiments, to deal with the increase in the number of input-output pairs.

![MSE During Training of X-axis Network with Items Present](image)
Table 4.3: ANN Configuration and Training Values for Simulation of X-axis Position

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of input-output pairs</td>
<td>1600</td>
</tr>
<tr>
<td>Training Method</td>
<td>Incremental</td>
</tr>
<tr>
<td>Hidden AF</td>
<td>Linear</td>
</tr>
<tr>
<td>Output AF</td>
<td>Sigmoid-Symmetric</td>
</tr>
<tr>
<td>Number of Inputs</td>
<td>270</td>
</tr>
<tr>
<td>Number of Outputs</td>
<td>4</td>
</tr>
<tr>
<td>Number of Hidden</td>
<td>11</td>
</tr>
<tr>
<td>Desired MSE</td>
<td>0.00001</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.7</td>
</tr>
<tr>
<td>Number of Epochs</td>
<td>50000</td>
</tr>
</tbody>
</table>

Figure 4.9 depicts the MSE during training. It can be clearly seen that a MSE close to 0.0 is obtained, indicating that a mapping was successfully found. It is with this in mind that the network is applied to a live test in the next section.

In order to demonstrate the estimation functioning in a real-time system, the ANN is included ROS node, the implementation of which is outlined in Section 3.4.2. Output from the ANN and the simulator ground truth are collected simultaneously for comparison.

Figure 4.10 depicts the route that the robot made through the arena during the live test. Intersecting each of the training positions for the rotation networks, then performing a rotation clockwise before continuing on to the next. When reaching the end of the arena, the robot turns to face in the direction it came from and returns to the position it started.

Figure 4.11 depicts a comparison of the x-axis motion as being estimated by the simulated odometry and the neural network. As scans are collected around four points, the ones which are trained for the rotation networks, a realization occurs after a lapse in time then a sudden clipping of the position occurs as it updates its location.

Figure 4.12 depicts a comparison of the x-axis rotation motion as it is estimated by the odometry and the ANN. A clockwise rotation occurs at each of the trained positions, when reaching the end of the arena, a clockwise 360° rotation
occurs, in addition to make the robot face the direction it came, a rotation of 180° is also made. After describing that behavior in the figure 4.12 it can be seen that there is a regular oscillation occurring for half of the run, with the ANN accurately estimating, after reaching the end, a number of turns are made on the spot leading to high switching. The returning vector to start points results in a 0° values with some misfiring during transitioning between rotation points. Finally the robot makes a 180° to reset to the original starting pose.

Both angular and linear velocities had a maximum of 0.2m/s.

Figure 4.10: Illustrating the Direction of Rotation along the X-axis of the Simulation

With the results of Figures 4.11 and 4.12 the approach is considered to be accurate enough to merit application to a 2-D scenario.

4.1.3 2-D

Section 4.1.2 deals with the training of an ANN for a second x-axis dimension, in order to present a complete localization a 2-D implementation is experimented with in this section. In order to adequately test the same approach in 2-D, a number of changes have to be made the experiments featured in Section 4.1.2. It is the case that a number of points expanded from the original 1-D line
Figure 4.11: Comparison of the ANN Output to the Odometry during Simulation of 1-D Localization

Figure 4.12: Comparison of the ANN Output to the Odometry during Simulation of Rotation during 1-D Localization
across the arena are set, then the rotation estimation for each point is handled. The following outlines the arena features, scan collection methods, AF and training method selection criteria, network training and a live demonstration of localization around the arena.

A number of training sets have to be collected for the x-axis, y-axis and rotation networks along each of those axes. For the x and y axes, a mapping between a scan and a binary sequence is needed. The binary output of the network is then transformed into the cartesian coordinates. In the case of this experiment, the collection distribution is done along the axis of travel, as depicted in Figure 4.9 for the x-axis, as depicted in Figure ?? for the y-axis, and as depicted in Figure 4.2 for the rotation.

Figure 4.13 depicts the distribution of scan collection. The distance between each collection is based on trial and error in development. Essentially, there is a distance of 1 meter between each x and y axis collection, but only a 0.5 meter distance between scans in the same collection. This is done such that a solid representation of each axis is trained to their corresponding networks, while there is enough distance between collections that confusion over which collection does not occur.

![Figure 4.13: Illustration of 2-D Training Points and Responsible Location ANNs](image)

Table A.2 provides an overview of the configuration of the x and y axes network when trained. Inclusion of the training graphs for all 7 axes in the experiment and rotation networks, would be repetitive, so they are excluded. Training for
all axes performed at the same level as in the previous experiments leading to practical inclusion in a live test.

In order to demonstrate the estimation functioning in a real-time system, the ANN is included ROS node, the implementation of which is outlined in Section 3.4.3. Output from the ANN and the simulator ground truth are collected simultaneously for later comparison.

Figure 4.14 depicts the route that the robot made through the arena during the live test. Intersecting a subset of trained positions. The course makes the robot travel in diagonal patterns across the arena, testing its ability to switch between axis networks.

Figure 4.15 depicts a comparison of the xy-axis motion as being estimated by the simulated odometry and the neural network. Due to the granularity of the scan collection, position estimates during diagonal movement clip to their closest neighbour reliably, however the distance between actual and neural is sizeable at these dimensions.

Figure 4.16 depicts a comparison of the xy-axis rotation motion as it is estimated by the odometry and the ANN. Performance is comparable to the x-axis rotation in the previous section, lending to an accurate algorithm.

Both angular and linear velocities had a maximum of 0.2m/s.

With the results of Figures 4.15 and 4.16 the approach is considered to be promising for further inspection.

### 4.2 Landmark Recognition

With the ANN showing promising results, the next step in ANN localization is considered. For more than three decades, the role of place cells in the human spatial navigation have been researched. Place cells are neurons in the hippocampus that present a high rate of firing when a person or animal are in a certain location (Moser et al., 2008). Another key element of the human navigation system are grid cells. Grid cells provide a sort of internal map, consisting of a dot for each known position, the better know the position becomes,
Figure 4.14: Illustrating the Direction of Rotation along the X and Y Axes of the Simulation

Figure 4.15: Comparison of the ANN Output to the Odometry during Simulation of 2-D Localization
the more the place cells build up and map to specific grid cells, this then results in clusters of vertices to form a grid of equilateral triangles [Moser et al., 2008].

While the work on understanding human spatial navigation is not complete, the picture that is emerging from the research that the neural network representation does not rely on a cartesian coordinate system as such. ANNs are a mathematical model of NNNs, it is then possible that the application of ANNs to MRL can learn from biological representations.

It is with the above in mind that an alternative approach to MRL is proposed. If it is the case that humans and animals don’t have an internal coordinate system, it is rather intuitive that a MRL based on ANNs in robots would follow the same architecture.

It is reasoned that a robot does not need to know its exact coordinate position to be able to localize itself, or for it to move from one location to another. It is instead only necessary for a robot to know its general location and the general locations it must pass through to get to another location. Once a robot knows its general location it can use alternative means to go to exact positions in that location, for example, using a camera.

Figure 4.16: Comparison of the ANN Output to the Odometry during Simulation of Rotation during 2-D Localization
In this way, the problem of MRL is broken into two parts, one that deals with distance localization (how a robot localizes across a whole map) and near localization (how a robot localizes in a known area of the map). This can be views as synonymous to how a person localizes in that, given a street address, a person will not et coordinates of that address and attempt to reduce the difference between their own coordinates and the destination, but will instead recognize several areas and streets they must pass through (distant localization), and on reaching the street, will walk along the street looking for the door number (near localization). To this end, experiments are done whereby an ANN is traine to recognize locations across a map, such that if a robot were anywhere inside these locations, the ANN would respond that it recognised it.

To add a certain degree of distinction to the arena, and to add rooms which mirror a more likely indoor environment, the arena has been changed from the previous experiments (Figure 4.1), this can be seen in Figure 4.17. The layout of the arena was chosen arbitrarily. There is a lack of items in the rooms, as the presence of doorways and corridors provide enough distinction for reliable recognition. The robot has the same configuration, with a FOV of 270° with a range of 20 meters. The size of the arena is left modest, while additional investigation could be provided if a larger map with open spaces larger than 20 meters are available.

Following from the collection details in Section 4.1.3, the implementation of this approach also relies on the collection of a number of scans from each direction. In Figure 4.2 illustrates the scan distributed collection. To train the ANN to recognize locations, several key points are chosen across the map. These points are chosen manually as an automated system to handle optimal cell coverage was considered outside the scope of this thesis. Training is then done by performing a 270° FOV scan 20 times, each at the same location, but at a different orientation, the distribution of which can be seen in Figure 4.2. While it was the focus in previous experiments to have a collection position evenly distributed across the map, it was not the case here, where a single room had to be recognized and the not the position within that room. Figure 4.18 depicts the distribution of the robots that are used for scan collection.

The AF selection for the networks at each position are deemed to be identical to those featured in Section 4.1.1. This is referring to the search for the correct
Figure 4.17: Simulated Arena with Distinct Rooms for Landmark Recognition Experiments

Figure 4.18: Illustration of Scan Collection Points in the Landmark Recognition Experiments
neuron type in the hidden and output layers. Each AF is again evaluated against all 13 other AFs with training for 2000 epochs, the best of which is determined with the lowest resulting MSE. Table A.4 lists the available AFs, of which the Symmetric-sigmoidal is chosen for the output layer, and the Linear for the hidden.

The training method used for these experiments is identical to that mentioned in Section 4.1.2 or the 1-D localization experiments. Incremental training is performed with the robot networks used. While configuration would appear sparser than in the 1-D experiments (Figure 4.18), the application is essentially the same, as the resulting network outputs a number corresponding to the position of each rotation network. However, the rotation not being utilized to recognize the room.

After reviewing the training methods in Section 4.2, the AFs in Section 4.2 and processing the simulated data, training of the network began. Table 4.4 lists the configuration details.

![MSE of ANN Training](image)

Figure 4.19: MSE During Training of Landmark Recognition Network
Table 4.4: The Training Configuration for Landmark Recognition.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Training Pairs</td>
<td>1600</td>
</tr>
<tr>
<td>Training Method</td>
<td>Incremental</td>
</tr>
<tr>
<td>AF on Hidden Layer</td>
<td>Linear</td>
</tr>
<tr>
<td>AF on Output Layer</td>
<td>Sigmoid-Symmetric</td>
</tr>
<tr>
<td>Number of Hidden Neurons</td>
<td>6</td>
</tr>
<tr>
<td>Number of Input Neurons</td>
<td>270</td>
</tr>
<tr>
<td>Number of Output Neurons</td>
<td>4</td>
</tr>
<tr>
<td>Desired Error (MSE)</td>
<td>0.00001</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.7</td>
</tr>
<tr>
<td>Max Number of Epochs</td>
<td>50000</td>
</tr>
<tr>
<td>Stopping Function</td>
<td>MSE</td>
</tr>
</tbody>
</table>

Figure 4.19 illustrates the progress of the training with the configuration in Table 4.4. Starting from a random network, a mapping is clearly found after 1300 epochs, based on this, it is assumed to be satisfactory to apply this to a live test.

The ANN is then tested to ensure that, given an arbitrary location and orientation within the scanned area, the ANN returns the correct unique identifier for that location.

Figure 4.20 depicts the autonomous path-finding of a mobile robot using purely landmark recognition. There is no coordinate system used. The robot follows a simple spiral pattern and wall following algorithms until a recognized landmark is found. As it can be seen in Figure 4.18, there are only 4 trained landmarks through the arena.

With a reactive path-finding algorithm in place, the responsible ROS node searches for the next recognizable landmark, using a vector field obstacle avoidance algorithm to stay away from walls, the robot can be clearly seen in Figure 4.20 to make slow but direct progress through the arena. Each time looking for a numerical landmark higher than the last, and away from lower numbers, finishing in Room 2 (Figure 4.20) or position 4 (Figure 4.21). It is thus concluded that the proposed landmark recognition approach is effective in this environment.
4.2 Landmark Recognition

Figure 4.20: Spiral Landmark Recognition Path-finding

Figure 4.21: Status of the Landmark Recognition ANN during Live Testing
4.3 Path Finding

Considering all stages of robot navigation (localization, path-planning, and path-execution) it is necessary to validate that any localization solution gives sufficient input to path-planning and path-execution. For the purpose of performing this validation, path-planning and path-execution are reduced to a simple abstraction-of-progress which asks, given a destination and a movement, is progress towards the destination made by that movement. This abstraction is a black-box around the localization system’s post-condition, and therefore should internally have a notion of origin. It should be the case that if a robot were to perform every movement to which this "progress black-box" responded in the affirmative, that the robot would eventually reach the destination, and conversely, if a robot were to perform every movement which received a negative response, that it would never reach the destination. The "progress black-box" should also respond in the neutral if the movement is orthogonal to progress, to say that progress was neither facilitated nor impeded.

An attempt is made to create a "progress black-box" for the solution featured in Section 4.2. The post-condition of Section 4.2 is as follows: a set of locations across the map (as (x, y) pairs); and a single location, from this set, as the one nearest the robot (referred to below as the robot’s location). It should be noted that the robot’s relative position to this nearest location and the robot’s orientation are not known. The "progress black-box" is created as such:

- keep in memory an ordered list of locations that follow the shortest path from the robot’s location to the destination location across the set of locations in the map
- if not previously populated, populate the shortest-path list from the robot’s current location
- if the movement results in the robot’s location remaining in the first location in the shortest-path list, respond in the neutral
- if the movement results in the robot’s location changing to a different location in the shortest-path list, reconstruct the shortest-path list and respond in the positive
• if the movement results in the robot’s location changing to a location not in the shortest-path list, reconstruct the shortest-path list and respond in the negative

A robot is then programmed to perform the following:

• if the robot’s location and destination location are the same, stop

• if the robot’s location differs from the destination location and a forward movement has a positive or neutral response, continue moving forward

• if the robot’s location differs from the destination location and a forward movement has a negative response, stop, turn right approximately 157°, and start moving forward again

• if the robot hits a wall, perform a wall hugging algorithm

This has the result of causing the robot to "bounce" off location cells that are away from the destination, and continue through location cells that are along the way to the destination. The turn of 157° causes the robot to almost follow a 5-point-star pattern that, if repeated, would cover all 360° in 1° increments. The idea of this is that, without the robot knowing with any particular accuracy where it is or, what direction it’s facing, it will tend to favor movements that are towards the destination.

As the important traits of the previous algorithm is that the robot continue moving straight when progress is positive or unknown, and change direction when progress is negative, then alternative robot algorithms are to either turn randomly for negative progress (requiring no notion of the degree of angle turned), or for the robot to follow a spiral pattern, turning more sharply when it receives a negative response, and resetting the spiral and switching direction when receiving a positive response.

Although this shows that coordinate- and orientation-free localization via ANN is possible, and that its output can be fed into the path-planning stage that follows, it does not lend itself to path-execution, resulting in poor performance at this stage of robot navigation. To address this, an alternative method of
4.3 Path Finding

ANN localization is explored; one where path-execution attributes are encoded straight into the localization training.

In the previous solution, the ANN is taught to recognize location by taking a position on the map and learning to look at its surroundings in all directions, such that, if a robot were to look at that surrounding again from roughly the same vantage as the trained location, it would recall which location was taught that. This resulted in an ANN that had an input of 270 LASER scans, and an output of 1 location identifier. This gives the path-planner enough information to create a path, but not enough information for the path-execution to accurately follow the path. If a solution is to be found that will allow the localization stage to give information to the path-execution, the path-planning stage must be assimilated by the localization stage’s ANN. Path-planning requires an origin and a destination as input, and gives as output a path to follow. For the localization ANN (which already embodies origin) to include path-planning, it must now also be given a destination as input, and instead give information about a path to follow as output. To do this, path-planning must happen as a per-processing step during training. This results in a localization ANN that takes, as input 270 LASER scans and 1 destination location identifier, and outputs a value suitable for the path-execution step to follow.

Training of this localization ANN is done in a manner similar to the previous solution except that for every location trained, instead of training each location in isolation, it must be trained against every other location; that is, it must be trained for every input destination, and it must do this for every viewed angle of 270 LAZER scan inputs. When training against all other locations, pre-processing must be done to calculate the shortest-path from the trained location to the destination location. The immediate direction to follow along this path is recorded as the ideal angle. The rest of the path is ignored. For every angle the location is trained on against this destination, the difference between the training angle the ideal angle is calculate using the following calculation:

\[
w = 1^\gamma(At + 1)/2 - Ai \cdot At < 0: -1; -Ai \cdot At \geq 0: 1 \tag{4.4}\]

This results in a value that is 0 if the training angle is equal to the ideal angle and extends to +- 1 in proportion to the training angle being off from the ideal
angle to $\pm 180^\circ$.

The robot navigation now has enough information to navigate through its environment to any known destination. This is achieved by the following steps:

- the robot takes a scan at its location
- this scan is passed to the localization [ANN] along with a destination
- the location where the scan is trained, along with the destination, will result in a response value from -1 to +1
- if the returned value is in the negative, the robot knows it must turn right, if it’s a positive value, the robot turns left
- the robot continues turning and passing scans to the localization [ANN]
  - as a corollary, the values returned by the localization [ANN] should tend towards 0
- once the robot is getting values below $\pm 0.5$ it can begin moving forward
- the robot will eventually move into a new location with different trained data and start receiving new values on how to reach the destination—different angles having been trained for the destination point in this new location
- if the robot hits a wall, perform a wall hugging algorithm
- the robot continues following the above steps until the destination is reached

In practice, the values returned from the localization [ANN] are not exact enough to allow the path-execution to be reactive. It is necessary for the path-execution to average the values over several requests, accepting that as the robot moves away from learned locations, the confidence it has in the returned values will decrease. Additionally, because the path-execution is given information in terms of the learned location, which may be away from the robot’s actual location, it may be given values that lead it into a wall. These problems can be reduced by adding more locations during the training of the localization [ANN].
4.3 Path Finding

Figure 4.22: Directed localization and path-finding with ANN

Figure 4.22 depicts the resulting path from the proposed algorithm. If a comparison is made with the reactive path in the previous Section 4.2, there is a clearly a more direct path emerging from this approach. Testing on this approach is not expanded upon in this thesis, but is mentioned as the topic of future work in Section 5.2.

This experiment simply places a more intelligent path-finding algorithm over the landmark recognition system from the previous Section 4.2. With this in mind, there is not any major requirement to make alterations to the arena. The arena is identical as depicted in the previous experiment, Figure 4.17.

The scan data collected for the landmark recognition experiment of Section 4.2 are identical for this experiment. The difference in this case is the inclusion of neighbouring position identifier. Since the number of position options is only four, the neighbouring positions are added manually, and no autogeneration code was developed. In Figure 4.18 the original collection points can be found.

The AF selection for the networks at each position are deemed to be identical to those featured in Section 4.1.1 and 4.2. Table A.4 lists the available AFs, of which the Symmetric-sigmoidal is chosen for the output layer, and the Linear for the hidden. Inclusion of an additional input to the network proved not to
require a change in AF selection, likely due to the normalization to the same range of -1 to +1 as the distance measurements.

The training method used for these experiments is identical to that mentioned in Section 4.1.2 and 4.2. Incremental training is performed with the robot networks used. As previously mentioned in Section 4.3, inclusion of an additional input to the network did not require a change in AF or training method selection.

After reviewing the training methods in Section 4.3, the AFs in Section 4.3 and processing the simulated data, training of the network began. Table 4.5 lists the configuration details.

![Figure 4.23: MSE During Training of Path-finding Network](image)

Figure 4.23 illustrates the progress of the training with the configuration in Table 4.5. Starting from a random network, a mapping is clearly found after 1350 epochs, based on this, it is assumed to be satisfactory to apply this to a live test.
### 4.3 Path Finding

Table 4.5: The Training Configuration for Path-finding.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Training Pairs</td>
<td>1680</td>
</tr>
<tr>
<td>Training Method</td>
<td>Incremental</td>
</tr>
<tr>
<td>AF on Hidden Layer</td>
<td>Linear</td>
</tr>
<tr>
<td>AF on Output Layer</td>
<td>Sigmoid-Symmetric</td>
</tr>
<tr>
<td>Number of Hidden Neurons</td>
<td>8</td>
</tr>
<tr>
<td>Number of Input Neurons</td>
<td>271</td>
</tr>
<tr>
<td>Number of Output Neurons</td>
<td>4</td>
</tr>
<tr>
<td>Desired Error (MSE)</td>
<td>0.00001</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.7</td>
</tr>
<tr>
<td>Max Number of Epochs</td>
<td>50000</td>
</tr>
<tr>
<td>Stopping Function</td>
<td>MSE</td>
</tr>
</tbody>
</table>
Chapter 5

Discussion

“Things do not happen. Things are made to happen.”
- John F. Kennedy

5.1 Summary

This thesis examines the existing localization methods used to date, from <x to y>, including coordinate-based ANN. It then goes on to test the limits of coordinate-based ANN, before introducing a novel method of localization using ANN that being one in which MRL is handled entirely by the ANN. This novel method is shown to take raw sensor scans as input into the ANN and output a generalized location. This method is also shown to avoid the need to recognise a robot’s coordinates or orientation, avoiding the need to worry about mechanical slippage and rounding errors. This thesis then looks at how this method fits in to the overall robot navigation process, bringing attention to a missing piece of information required for path-execution. This paper then ends by showing how bringing more of the robot navigation process into the ANN presenting a path-execution algorithm.
5.2 Future Work

The inclusion of an automated tool for the collection of scans to make training 2-D maps faster.

It is recognised that one draw-back to this approach is that it relies heavily on a priori knowledge. This is due to the fact that path-finding information is encoded into the ANN, requiring pathfinding to be a pre-processing step. A solution to this is left as a suggestion for future work.

Another area for future work is the idea of learning as the robot moves about. This means creating new scans and all the troubles that are associated with that, backtracking and doing the path-finding parts.

Moving to a specific point in the room is another aspect of this approach that has to be researched. For example, currently, the algorithm allows for a robot to travel to a particular room, however a specific point in the room cannot yet be reached. The current algorithms could be expanded upon to include an image processing element to recognise specific places within a room.
References


Clerc, M. and Kennedy, J. (2002). The particle swarm - explosion, stability, and convergence in a multidimensional complex space. Evolution-
REFERENCES


REFERENCES


REFERENCES


REFERENCES


Appendix A

Training Method & Activation Function Selection
<table>
<thead>
<tr>
<th>Training Method</th>
<th>Mean Squared Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental</td>
<td>0.127913</td>
</tr>
<tr>
<td>Batch</td>
<td>0.188661</td>
</tr>
<tr>
<td>RPROP</td>
<td>0.106841</td>
</tr>
<tr>
<td>QuickPROP</td>
<td>0.188909</td>
</tr>
</tbody>
</table>

Table A.1: The MSE after 2000 Epochs using each Training Method for Rotation without Items.

<table>
<thead>
<tr>
<th>Training Method</th>
<th>Mean Squared Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental</td>
<td>0.000000</td>
</tr>
<tr>
<td>Batch</td>
<td>0.188728</td>
</tr>
<tr>
<td>RPROP</td>
<td>0.064792</td>
</tr>
<tr>
<td>QuickPROP</td>
<td>0.188973</td>
</tr>
</tbody>
</table>

Table A.2: The MSE after 2000 Epochs using each Training Method for Rotation with Items.

<table>
<thead>
<tr>
<th>Training Method</th>
<th>Mean Squared Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental</td>
<td>0.055685</td>
</tr>
<tr>
<td>Batch</td>
<td>0.181106</td>
</tr>
<tr>
<td>RPROP</td>
<td>0.076525</td>
</tr>
<tr>
<td>QuickPROP</td>
<td>0.189025</td>
</tr>
</tbody>
</table>

Table A.3: The MSE after 2000 Epochs using each Training Method for X-axis Position.
<table>
<thead>
<tr>
<th>Index</th>
<th>Activation Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Linear</td>
</tr>
<tr>
<td>2</td>
<td>Sigmoid</td>
</tr>
<tr>
<td>3</td>
<td>Sigmoid-Stepwise</td>
</tr>
<tr>
<td>4</td>
<td>Sigmoid-Symmetric</td>
</tr>
<tr>
<td>5</td>
<td>Sigmoid-Symmetric-Stepwise</td>
</tr>
<tr>
<td>6</td>
<td>Gaussian</td>
</tr>
<tr>
<td>7</td>
<td>Gaussian-Symmetric</td>
</tr>
<tr>
<td>8</td>
<td>Elliot</td>
</tr>
<tr>
<td>9</td>
<td>Elliot-Symmetric</td>
</tr>
<tr>
<td>10</td>
<td>Linear-Piece</td>
</tr>
<tr>
<td>11</td>
<td>Linear-Piece-Symmetric</td>
</tr>
<tr>
<td>12</td>
<td>Sin-Symmetric</td>
</tr>
<tr>
<td>13</td>
<td>Cos-Symmetric</td>
</tr>
</tbody>
</table>

Table A.4: The Index of Activation Functions.
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.901577</td>
<td>1.0</td>
<td>1.0</td>
<td>0.192180</td>
<td>0.124629</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.120502</td>
<td>1.0</td>
<td>0.335111</td>
<td>0.156765</td>
<td>0.374993</td>
</tr>
<tr>
<td>2</td>
<td>0.940281</td>
<td>1.0</td>
<td>1.0</td>
<td>0.188976</td>
<td>0.135165</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.188974</td>
<td>1.0</td>
<td>0.188976</td>
<td>0.749817</td>
<td>0.615023</td>
</tr>
<tr>
<td>3</td>
<td>0.940145</td>
<td>1.0</td>
<td>1.0</td>
<td>0.188976</td>
<td>0.126087</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.188974</td>
<td>1.0</td>
<td>0.188976</td>
<td>0.749817</td>
<td>0.570900</td>
</tr>
<tr>
<td>4</td>
<td>0.938972</td>
<td>1.0</td>
<td>1.0</td>
<td>0.185336</td>
<td>0.126223</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.108012</td>
<td>1.0</td>
<td>0.189069</td>
<td>0.187509</td>
<td>0.254265</td>
</tr>
<tr>
<td>5</td>
<td>0.538269</td>
<td>1.0</td>
<td>1.0</td>
<td>0.189093</td>
<td>0.125003</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.123152</td>
<td>1.0</td>
<td>0.194401</td>
<td>0.201690</td>
<td>0.620586</td>
</tr>
<tr>
<td>6</td>
<td>0.961667</td>
<td>1.0</td>
<td>1.0</td>
<td>0.125003</td>
<td>0.127512</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.188974</td>
<td>1.0</td>
<td>0.188976</td>
<td>0.188976</td>
<td>0.188974</td>
</tr>
<tr>
<td>7</td>
<td>0.993008</td>
<td>1.0</td>
<td>1.0</td>
<td>0.188974</td>
<td>0.188970</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.188971</td>
<td>1.0</td>
<td>0.188968</td>
<td>0.188974</td>
<td>0.188977</td>
</tr>
<tr>
<td>8</td>
<td>0.394203</td>
<td>1.0</td>
<td>1.0</td>
<td>0.085253</td>
<td>0.126091</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.062524</td>
<td>1.0</td>
<td>0.069448</td>
<td>0.122808</td>
<td>0.124999</td>
</tr>
<tr>
<td>9</td>
<td>0.637258</td>
<td>1.0</td>
<td>1.0</td>
<td>0.108429</td>
<td>0.141973</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.091123</td>
<td>1.0</td>
<td>0.188947</td>
<td>0.202338</td>
<td>0.624928</td>
</tr>
<tr>
<td>10</td>
<td>0.993183</td>
<td>1.0</td>
<td>1.0</td>
<td>0.188975</td>
<td>0.188975</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.188973</td>
<td>1.0</td>
<td>0.188973</td>
<td>0.188975</td>
<td>0.495378</td>
</tr>
<tr>
<td>11</td>
<td>0.899851</td>
<td>1.0</td>
<td>1.0</td>
<td>0.188972</td>
<td>0.126471</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.124904</td>
<td>1.0</td>
<td>0.188975</td>
<td>0.212926</td>
<td>0.489446</td>
</tr>
<tr>
<td>12</td>
<td>0.761837</td>
<td>1.0</td>
<td>1.246459</td>
<td>0.367311</td>
<td>0.126678</td>
<td>1.0</td>
<td>0.25</td>
<td>2.984976</td>
<td>0.188974</td>
<td>1.0</td>
<td>0.195024</td>
<td>0.178235</td>
<td>0.279830</td>
</tr>
<tr>
<td>13</td>
<td>0.811237</td>
<td>1.0</td>
<td>1.0</td>
<td>0.132384</td>
<td>0.130857</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.135034</td>
<td>1.0</td>
<td>0.125011</td>
<td>0.170706</td>
<td>0.248362</td>
</tr>
</tbody>
</table>

Table A.5: The MSE after 2000 Epochs using each Activation Function Combination for Rotation without Items (Row: Hidden Layer Activation Function, Column: Output Layer Activation Function)
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>90000.66</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001653</td>
<td>0.374733</td>
<td>1.0</td>
<td>0.0</td>
<td>0.499878</td>
<td>0.244924</td>
</tr>
<tr>
<td>2</td>
<td>0.516843</td>
<td>1.0</td>
<td>1.0</td>
<td>0.191485</td>
<td>0.122122</td>
<td>1.0</td>
<td>0.186176</td>
<td>1.002382</td>
<td>0.191328</td>
<td>1.0</td>
<td>0.067299</td>
<td>0.177167</td>
<td>0.184076</td>
</tr>
<tr>
<td>3</td>
<td>0.516843</td>
<td>1.0</td>
<td>1.0</td>
<td>0.191485</td>
<td>0.193193</td>
<td>1.0</td>
<td>0.185638</td>
<td>1.002382</td>
<td>0.191328</td>
<td>1.0</td>
<td>0.164992</td>
<td>0.177167</td>
<td>0.184076</td>
</tr>
<tr>
<td>4</td>
<td>48.182339</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.002382</td>
<td>0.000453</td>
<td>1.0</td>
<td>0.0</td>
<td>0.25</td>
<td>0.594692</td>
</tr>
<tr>
<td>5</td>
<td>42.074204</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.002382</td>
<td>0.000469</td>
<td>1.0</td>
<td>0.0</td>
<td>0.25</td>
<td>0.75</td>
</tr>
<tr>
<td>6</td>
<td>0.516843</td>
<td>1.0</td>
<td>1.0</td>
<td>0.191485</td>
<td>0.193193</td>
<td>1.0</td>
<td>0.186176</td>
<td>1.002997</td>
<td>0.191328</td>
<td>1.0</td>
<td>0.164992</td>
<td>0.177167</td>
<td>0.184076</td>
</tr>
<tr>
<td>7</td>
<td>35.666785</td>
<td>1.0</td>
<td>1.0</td>
<td>0.157243</td>
<td>0.161890</td>
<td>1.0</td>
<td>0.138038</td>
<td>1.002382</td>
<td>0.191456</td>
<td>1.0</td>
<td>0.1</td>
<td>0.25</td>
<td>0.124699</td>
</tr>
<tr>
<td>8</td>
<td>1.046225</td>
<td>1.0</td>
<td>1.0</td>
<td>0.159738</td>
<td>0.086145</td>
<td>1.0</td>
<td>0.223746</td>
<td>1.002381</td>
<td>0.002556</td>
<td>1.0</td>
<td>0.148744</td>
<td>0.130234</td>
<td>0.299032</td>
</tr>
<tr>
<td>9</td>
<td>35.129622</td>
<td>1.0</td>
<td>1.0</td>
<td>0.156152</td>
<td>0.156975</td>
<td>1.0</td>
<td>0.125</td>
<td>1.002381</td>
<td>0.000501</td>
<td>1.0</td>
<td>0.1</td>
<td>0.25</td>
<td>0.749982</td>
</tr>
<tr>
<td>10</td>
<td>0.516843</td>
<td>1.0</td>
<td>1.0</td>
<td>0.191485</td>
<td>0.161105</td>
<td>1.0</td>
<td>0.158212</td>
<td>1.002993</td>
<td>0.191328</td>
<td>1.0</td>
<td>0.105347</td>
<td>0.177167</td>
<td>0.184076</td>
</tr>
<tr>
<td>11</td>
<td>39.775974</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.062502</td>
<td>1.002382</td>
<td>0.000558</td>
<td>1.0</td>
<td>0.0</td>
<td>0.25</td>
<td>0.591799</td>
</tr>
<tr>
<td>12</td>
<td>29.851565</td>
<td>1.0</td>
<td>1.0</td>
<td>0.157252</td>
<td>0.161902</td>
<td>1.0</td>
<td>0.138049</td>
<td>1.003761</td>
<td>0.191457</td>
<td>1.0</td>
<td>0.1</td>
<td>0.25</td>
<td>0.278599</td>
</tr>
<tr>
<td>13</td>
<td>0.514791</td>
<td>1.0</td>
<td>1.0</td>
<td>0.191468</td>
<td>0.193184</td>
<td>1.0</td>
<td>0.186143</td>
<td>1.002728</td>
<td>0.191332</td>
<td>1.0</td>
<td>0.132442</td>
<td>0.177100</td>
<td>0.184027</td>
</tr>
</tbody>
</table>

Table A.6: The MSE after 2000 Epochs using each Activation Function Combination for Rotation with Items (Row: Hidden Layer Activation Function, Column: Output Layer Activation Function)
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>90003.36</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001691</td>
<td>0.302408</td>
<td>1.0</td>
<td>0.0</td>
<td>0.624847</td>
<td>0.244588</td>
</tr>
<tr>
<td>2</td>
<td>0.125681</td>
<td>1.0</td>
<td>1.0</td>
<td>0.112089</td>
<td>0.116714</td>
<td>1.0</td>
<td>0.130240</td>
<td>1.001691</td>
<td>0.180674</td>
<td>1.0</td>
<td>0.025</td>
<td>0.069586</td>
<td>0.75</td>
</tr>
<tr>
<td>3</td>
<td>0.125679</td>
<td>1.0</td>
<td>1.0</td>
<td>0.112089</td>
<td>0.116714</td>
<td>1.0</td>
<td>0.090320</td>
<td>1.001691</td>
<td>0.180674</td>
<td>1.0</td>
<td>0.025</td>
<td>0.069586</td>
<td>0.749992</td>
</tr>
<tr>
<td>4</td>
<td>406.658867</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.000001</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001691</td>
<td>0.000452</td>
<td>1.0</td>
<td>0.0</td>
<td>0.25</td>
<td>0.290450</td>
</tr>
<tr>
<td>5</td>
<td>401.930625</td>
<td>1.0</td>
<td>1.0</td>
<td>0.061079</td>
<td>0.054666</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001691</td>
<td>0.016311</td>
<td>1.0</td>
<td>0.0</td>
<td>0.249985</td>
<td>0.302864</td>
</tr>
<tr>
<td>6</td>
<td>0.125672</td>
<td>1.0</td>
<td>1.0</td>
<td>0.112089</td>
<td>0.116714</td>
<td>1.0</td>
<td>0.090320</td>
<td>1.001691</td>
<td>0.180674</td>
<td>1.0</td>
<td>0.025</td>
<td>0.069586</td>
<td>0.85599</td>
</tr>
<tr>
<td>7</td>
<td>392.648555</td>
<td>1.0</td>
<td>1.0</td>
<td>0.066747</td>
<td>0.063289</td>
<td>1.0</td>
<td>0.243181</td>
<td>1.001691</td>
<td>0.120008</td>
<td>1.0</td>
<td>0.025</td>
<td>0.249798</td>
<td>0.170222</td>
</tr>
<tr>
<td>8</td>
<td>0.375647</td>
<td>1.0</td>
<td>1.0</td>
<td>0.058496</td>
<td>0.067497</td>
<td>1.0</td>
<td>0.110628</td>
<td>1.001691</td>
<td>0.111795</td>
<td>1.0</td>
<td>0.025</td>
<td>0.088107</td>
<td>0.749997</td>
</tr>
<tr>
<td>9</td>
<td>413.416367</td>
<td>1.0</td>
<td>1.0</td>
<td>0.063543</td>
<td>0.061078</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001691</td>
<td>0.113655</td>
<td>1.0</td>
<td>0.025</td>
<td>0.249988</td>
<td>0.203425</td>
</tr>
<tr>
<td>10</td>
<td>0.125672</td>
<td>1.0</td>
<td>1.0</td>
<td>0.112089</td>
<td>0.116714</td>
<td>1.0</td>
<td>0.092405</td>
<td>1.001691</td>
<td>0.180674</td>
<td>1.0</td>
<td>0.025</td>
<td>0.069586</td>
<td>0.583900</td>
</tr>
<tr>
<td>11</td>
<td>398.54875</td>
<td>1.0</td>
<td>1.0</td>
<td>0.000001</td>
<td>0.0</td>
<td>1.0</td>
<td>0.25</td>
<td>1.001691</td>
<td>0.055021</td>
<td>1.0</td>
<td>0.0</td>
<td>0.25</td>
<td>0.289869</td>
</tr>
<tr>
<td>12</td>
<td>467.362383</td>
<td>1.0</td>
<td>1.0</td>
<td>0.066546</td>
<td>0.063293</td>
<td>1.0</td>
<td>0.25</td>
<td>1.002361</td>
<td>0.120016</td>
<td>1.0</td>
<td>0.025</td>
<td>0.25</td>
<td>0.307331</td>
</tr>
<tr>
<td>13</td>
<td>0.124626</td>
<td>1.0</td>
<td>1.0</td>
<td>0.111842</td>
<td>0.116451</td>
<td>1.0</td>
<td>0.169968</td>
<td>1.002383</td>
<td>0.180562</td>
<td>1.0</td>
<td>0.025</td>
<td>0.069255</td>
<td>0.086281</td>
</tr>
</tbody>
</table>

Table A.7: The MSE after 2000 Epochs using each Activation Function Combination for X-axis Position

(Row: Hidden Layer Activation Function, Column: Output Layer Activation Function)