2012

A Generalized Neural Network Approach to Mobile Robot Navigation and Obstacle Avoidance

Seyyed Hamid Dezfoulian
University of Windsor

A Generalized Neural Network Approach to Mobile Robot Navigation and Obstacle Avoidance

by

Hamid Dezfoulian

A Thesis
Submitted to the Faculty of Graduate Studies
through Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2011

A Generalized Neural Network Approach to Mobile Robot Navigation and Obstacle Avoidance

by

Hamid Dezfoulian

APPROVED BY:

_____

Dr. Jonathan Wu
Department of Electrical and Computer Engineering

_____

Dr. Alioune Ngom
School of Computer Science

_____

Dr. Imran Ahmad, Co-Advisor
School of Computer Science

_____

Dr. Dan Wu, Advisor
School of Computer Science

_____

Dr. Yung H. Tsin, Chair of Defense
School of Computer Science

September 16, 2011

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

In this thesis, we tackle the problem of extending neural network navigation algorithms for various types of mobile robots and 2-dimensional range sensors. We propose a general method to interpret the data from various types of 2-dimensional range sensors and a neural network algorithm to perform the navigation task. Our approach can yield a global navigation algorithm which can be applied to various types of range sensors and mobile robot platforms. Moreover, this method allows the neural networks to be trained using only one type of 2-dimensional range sensor, which contributes positively to reducing the time required for training the networks. Experimental results carried out in simulation environments demonstrate the effectiveness of our approach in mobile robot navigation for different kinds of robots and sensors. Therefore, the successful implementation of our method provides a solution to apply mobile robot navigation algorithms to various robot platforms.

DEDICATION

This thesis is dedicated to my parents who taught me the value of education and for their endless love, support and encouragement. I am deeply indebted to them for their continued support and unwavering faith in me.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my supervisor, Dr. Dan Wu for giving me an opportunity to pursue a Master's program and for his support, guidance and patience throughout this project whilst allowing me the room to work in my own way. I would also like to express my thanks to Dr. Imran Ahmad for accepting to be my co-supervisor and for his guidance and motivation throughout this research project and lavishly, providing the primers.

Gratitude is also expressed to my committee: Dr. Alioune Ngom for his invaluable support and guidance during my first steps into the field of machine learning and pattern recognition and Dr. Jonathan Wu for his detailed and constructive comments.

Finally, I am eternally grateful to my father for his support, guidance and help in all the time of research and writing of this thesis. With his help in countless ways it was possible for me to complete this research project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

Navigation is one of the most important problems in designing and developing intelligent mobile robots. Staying operational, i.e. avoiding dangerous situations such as collisions and staying within safe operating conditions (temperature, radiation, exposure to weather, etc.) comes first. But if any tasks are to be performed that relate to specific places in the robot environment, navigation is a must.

Robot navigation is defined by the ability of a mobile robot to determine its own position in its frame of reference and then to plan a path towards some goal location [1]. In order to navigate in an environment, the mobile robot requires representation, i.e. a map of the environment, and the ability to interpret that representation. Therefore navigation can be defined as the combination of the three fundamental abilities [1]:

- Self-Localisation
- Path Planning
- Map-Building and Map-Interpretation

In this context, map represents any mapping of the environment onto an internal representation. Moreover, Robot localization indicates the ability of the robot to establish its own position and orientation within the frame of reference.

Path planning is effectively an extension of localization, in that it requires the determination of the robot's current position and a position of a goal location, both within the same frame of reference or coordinates. If the environment is unknown to the robot, then the path planning stage has no sense. In this case, the navigation strategy is purely

reactive. The inputs to the mobile robot navigator are the target position and the sensor system data. If there are no obstacles between the robot and its target, the navigation path is just a straight line between them. If an obstacle is detected, some avoidance strategy is required. Potential function based methods [2,3], neural networks [4-9], and fuzzy logic based controllers [10-13], trained with a heuristic database of rules, are among the possibilities.

Finally, Map-Building can be in the form of a metric map or any notation describing locations in the frame of reference.

In the past few years, neural networks including feedforward neural network, self-organizing neural network, principal component analysis (PCA), dynamic neural network, support vector machines (SVM), neuro-fuzzy approach, etc., [14,15] have been extensively used in mobile robot navigation field [16]. This is due to their assets such as nonlinear mapping, ability to learn from examples, good generalization performance, massively parallel processing, and ability to approximate any function given adequate number of neurons.

Sensors are necessary for a robot to know where it is or how it got there, or to be able to reason about where it has to go. The sensors may be roughly divided into two classes: internal state sensors, such as accelerometers, gyroscopes, and external state sensors, such as laser sensors, infrared sensors, sonar, and visual sensors. The data from internal state sensors are used for estimating the position of the robot in a 2-dimensional space. The data from external state sensors provide information that can be used to recognize obstacles or a situation, or to build a map of the environment. The laser, infrared, and sonar sensors can provide distant and directional information about an

object. Due to the inevitable sensor noise, in most cases, the sensor readings are inaccurate and unreliable. Therefore, it is essential for the navigation algorithm to process the sensor data with noises. Since neural networks have many processing nodes, each with primarily local connections, they may provide some degree of robustness or fault tolerance for interpretation of the sensor data. [16]

However, most of the current research addresses one particular type of sensor or robot platform. The main issue with neural network approaches is the training of the network. Collecting sufficient, yet valuable, samples from the environment to train the network can sometimes be frustrating and very time consuming [17]. In addition, apart from the effort that has to be put to collect valuable samples, the training time of a network can be significantly high [17]. In any neural network navigation algorithm, if the robot platform or the type or number of the sensors are changed or altered, the network architecture requires some modifications to accommodate with the new amount of sensor data. Moreover, new training samples need to be gathered as the previous samples will not be as much useful for the new robot platform. In other words, when a network structure is designed for a specific type of sensor, it cannot be used for other types or different numbers of sensors. By changing the structure of the network, therefore, new training samples are required and the network needs to be trained from the beginning. This presents challenge and opportunity to develop a general method to interpret sensor data from different types of sensors that can yield a global navigation algorithm which can be applied to various types of sensors and robot platforms.

1.2 Contributions

This thesis is concerned with the problem of generalizing the interpretation of sensory data and mobile robot navigation. The contributions of this thesis are as follows:

The primary contribution of this thesis is to develop a general method for interpretation of different types of sensors, such as laser and ultrasonic sensors. Our approach extends the work done by Janglová [18] for determining the free-spaces by applying PCA Neural Network (PCNN). We study the problem of how current neural network navigation approaches are limited to one type of sensor and the kinematic constraints of a mobile robot. Our approach however is extendable to various 2-dimensional sensors and mobile robots. On the other hand, this approach allows the neural networks to be trained using only one type of sensor which contributes positively to reducing the training time. Experimental results, carried out in simulated environments, demonstrate that our approach can be positively affective in mobile robot navigation for different kinds of robots and sensors, when compared to previous works. Therefore, the successful implementation of our method provides a solution to apply navigation algorithms to various robot platforms.

The second important contribution of this thesis is to implement an algorithm to perform the navigation task using our interpretation of sensory data. Our approach is inspired by the works done by Parhi and Singh [19-21] for neural network robot navigation. Parhi and Singh introduced a real-time obstacle avoidance approach, solving each of the target-seeking, obstacle-avoidance, and wall-following tasks with separate neural networks. However, it is our belief that a multilayer neural network is capable of solving both target-seeking and object-avoidance tasks at the same time. Therefore,

instead of using two separate networks, we introduce a structure which uses only one network for this purpose. Yet, the wall-following task will require a more complex structure to accommodate with both directions of rotation. Therefore, our proposed method for mobile robot navigation can yield significant navigation results for various sensors and robots – at less training time and lower sensor costs.

The third contribution of this thesis is that we develop a software application to carry out the proposed approach. The experimental results obtained through this application indicate feasibility of our approach in simulation robots.

1.3 Guide to the Thesis

This thesis is organized as follows.

Chapter 2: Background Knowledge. This chapter provides an introduction to the subjects that the proposed method builds upon. After explaining the concept of artificial neural networks and backpropagation algorithm, some methods of feature extraction and classification will be given. The Principal Component Neural Network (PCNN) method and Support Vector Machine (SVM) algorithms are specifically emphasized, since they constitute the core of the proposed approach. The attention then moves to the discussion of mobile robot navigation and its current applications.

Chapter 3: Design and Methodology. The proposed interpretation of sensory data and mobile robot navigation method based on neural networks is presented in detail in this chapter. First the definition of the problem is described, followed by detailed presentation of the proposed approach.

Chapter 4: Implementation and Experiments. The detailed information of the implementation and the experimental results will be described in this chapter. In the

results section, experiments carried out for training and experiments done for testing on simulation robots are described. Finally, these experimental results are compared with experimental results from previous methods and the evaluations are obtained.

Chapter 5: Conclusion and Future Work. This final chapter brings conclusion of the thesis and presents a sketch of possible future work.

Chapter 2

Background Knowledge

This chapter provides the background knowledge on which the proposed method is based on. After explaining Artificial Neural Networks (ANN), feature extraction methodology, specifically Principal Component Analysis (PCA), is described. Consequently, a method of classification, Support Vector Machines (SVM), is illustrated. Finally, current robot navigation and obstacle avoidance algorithms are reviewed with a view to the applications of artificial neural networks in robot navigation and obstacle avoidance.

2.1 Artificial Neural Networks

Artificial neural networks are information-processing systems which have certain performance characteristics in common with biological neural networks [22]. Artificial neural networks have been evolved as generalizations of mathematical models of human cognition or neural biology, based on the following four assumptions [17]:

1. "Information processing occurs at many simple elements called neurons."
2. "Signals are passed between neurons over connection links."
3. "Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted."
4. "Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal."

A neural network can be characterized, firstly, by its structure of connections between the neurons (known as its architecture), additionally by its method of

determining the weights on the connections (called its training, or learning, algorithm), and finally, its activation function.

Neural networks are structured from a large number of simple processing components called neurons, units, cells, or nodes. Each neuron is connected to other neurons through directed communication links, each with a weight associated to it (as shown in Figure 1). The weights correspond to information being processed by the network to solve a problem. Neural networks can be applied to a wide selection of problems, such as storing and recalling data or patterns, grouping similar patterns, performing general mappings from input patterns to output patterns, classifying patterns, or finding solutions to constrained optimization problems.[17,22]

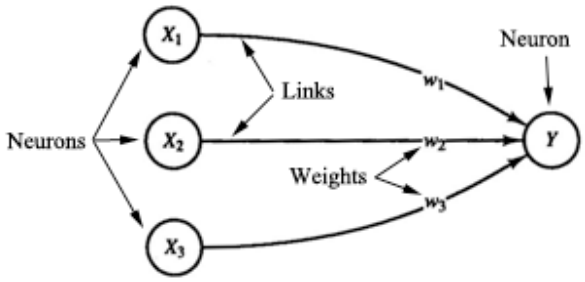


Figure 1: A simple (artificial) neuron [17]

The internal state of a neuron is known as its activation or activity level, which is a function of the inputs it has received. Typically, activation is sent as a signal from one neuron to several other neurons. However, only one signal can be sent from each neuron at the same time, although that signal can be broadcast to several other neurons. For example, consider neuron Y, shown in Figure 1, that receives inputs from neurons X_1, X_2, and X_3. The activations (output signals) of these neurons are x_1, x_2, and x_3, respectively. In addition, the weights on the connections from X_1, X_2, and X_3 to neuron Y are w_1, w_2,

and w_3, respectively. The net input, y_in, to neuron Y is the sum of the weighted signals from neurons X_1, X_2, and X_3, that is:

$$y\_in = w_1x_1 + w_2x_2 + w_3x_3$$

The activation y of neuron Y is given by some function of its net input, $y = f(y\_in)$, for example, the logistic sigmoid function (an S-shaped curve)

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

or any of a number of other activation functions (see Table 1 for a number of common activation functions in use with neural networks).

Further, suppose that neuron Y is connected to neurons Z_1, and Z_2, with weights v_1, and v_2, respectively, as depicted in Figure 2. neuron Y sends its signal y to each of these units. However, generally, the values received by neurons Z_1, and Z_2 will be different. Since each signal is scaled by the appropriate weight, v_1 or v_2. As shown in this simple example, in a typical network, the activations z_1 and z_2 of neurons Z_1, and Z_2 would depend on inputs from several neurons and not just one. [17]

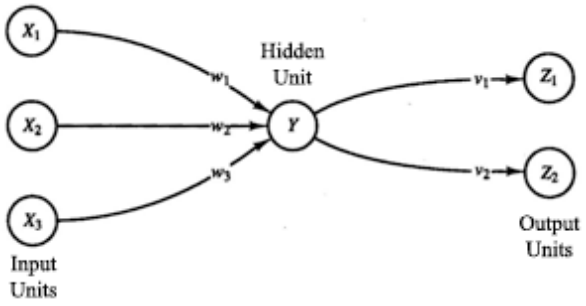


Figure 2: A very simple neural network [17]

Even though the neural network in Figure 2 is very simple, the presence of an intermediate unit Y (also known as the *hidden unit*), together with a nonlinear activation function, gives the network the capability to solve many more problems than can be

solved by a network with only input and output units. However, the difficulty to train (i.e., find optimal values for the weights) a net with hidden units is more than a network with no hidden units.

Table 1: Common activation functions in use with neural networks. [17,22]

	Function	**Definition**	**Range**
a)	Identity	x	$(-\infty, +\infty)$
b)	Binary Sigmoid	$\frac{1}{1+e^{-x}}$	$(0, +1)$
c)	Bipolar Sigmoid	$\frac{1-e^{-x}}{1+e^{-x}}$	$(-1, +1)$
d)	Hyperbolic	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$(-1, +1)$
e)	- Exponential	e^{-x}	$(0, +\infty)$
f)	Softmax	$\frac{e^{x}}{\sum_i e^{x_i}}$	$(0, +1)$
g)	Unit sum	$\frac{x}{\sum_i x_i}$	$(0, +1)$
h)	Square root	$\sqrt{x}$	$(0, +\infty)$
i)	Sine	$\sin(x)$	$[0, +1]$
j)	Ramp	$\begin{cases} -1 & x \le -1 \\ x & -1 < x < +1 \\ +1 & x \ge +1 \end{cases}$	$[-1, +1]$
k)	Step	$\begin{cases} 0 & x < 0 \\ +1 & x \ge 0 \end{cases}$	$[0, +1]$

2.1.1 Typical Architectures

Often, it is more convenient to visualize neurons arranged in layers. Normally, neurons that are in the same layer behave in the same manner. The key factors in determining the behaviour of a neuron are activation function and the pattern of weighted connections. Within each layer, neurons typically have the same activation function and the same pattern of connections with other neurons.

The arrangement of neurons into layers and the connection patterns within and between layers is known as the *network architecture* [17]. Many neural networks have an input layer in which the activation of each unit is equal to an external input signal. The network presented in Figure 2 consists of three input units, two output units, and one hidden unit (a unit that is neither an input unit nor an output unit).

Neural networks are typically classified into two categories; single layer and multilayer. Since no computation is performed by the input units, they are not counted as a layer when determining the number of layers. Similarly, the number of layers in the network can be defined as the number of layers of weighted interconnected links between the layers of neurons. This point of view is motivated by the fact that the weights in a network have extremely important information [17]. The network depicted in Figure 2 has two layers of weights.

Illustrated in Figure 3 are examples of single-layer and multilayer *feedforward* networks—networks in which the signals flow in a forward direction from the input units to the output units.

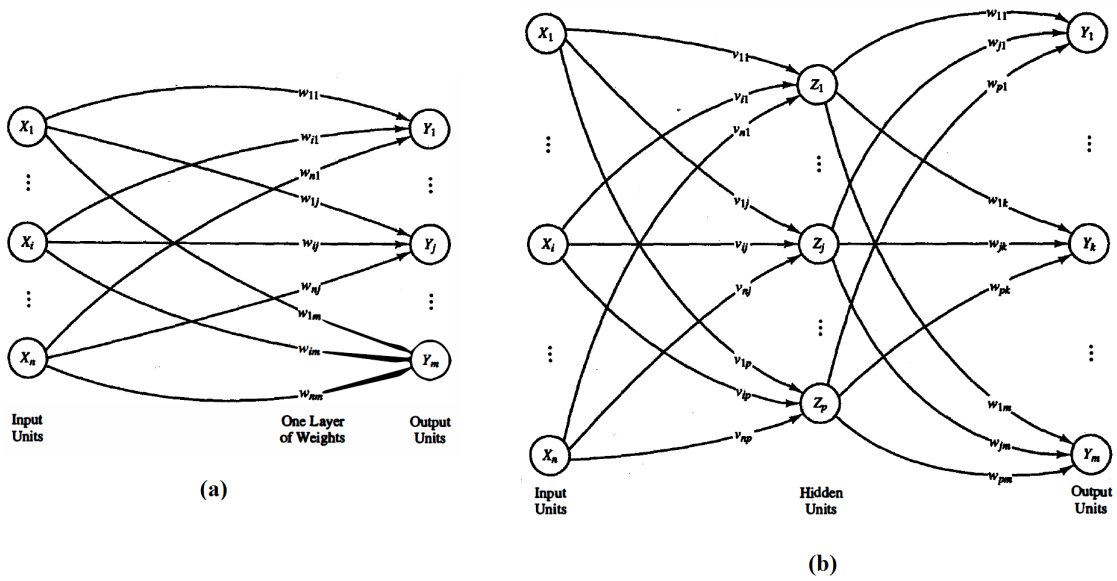


Figure 3: a) A single-layer neural net. b) A multi-layer neural net.[17]

For pattern classification, each output unit corresponds to a particular category to which an input vector may or may not belong. Note that in a single-layer net, the weights of output units will not be influenced by the weights of other output units. For pattern association, the same architecture can be used; however the overall pattern of output signals gives the response pattern associated with the input signal that caused it to be produced. These two examples illustrate that depending on the interpretation of the response of the network, the same type of network can be used for different problems. Alternatively, for more complicated mapping problems a multilayer network maybe required. The problems that require multilayer networks may still represent classification or association of patterns. Although the type of problem affects the choice of architecture, but it does not exclusively determine it.

A multilayer neural network is a network with one or more layers of nodes (hidden units) between the input units and the output units. Usually, there is a layer of

weights between two adjacent layers of units (input, hidden, or output). Multilayer networks can solve more complex problems than single-layer networks can, but training may be more complicated. Nevertheless, in some cases, training may be more successful, since it is possible to solve problems that single-layer networks cannot be trained to perform correctly at all. [22]

In addition to the architecture, the method of setting the values of the weights (training) is an important distinguishing attribute of various neural networks. Typically, neural networks are distinguished by two types of training—supervised and unsupervised; furthermore, there are networks whose weights are fixed without an iterative training process. [17,22]

Various tasks that neural nets can be trained to carry out fall into areas such as mapping, clustering, and constrained optimization. Pattern classification and pattern association may be considered special forms of the more general problem of mapping input vectors or patterns to the specified output vectors or patterns. [22]

Possibly, in the most standard neural network setting, training is achieved by introducing a series of training vectors, or patterns, each with an associated target output vector. Then based on a learning algorithm the weights are adjusted. This process is called *supervised training* [17]. Some of the simplest neural networks are designed to perform pattern classification that is to classify an input vector as either it belongs to or does not belong to a given category. In this type of neural network, the output is a bivalent element, say, either 1 (if the input vector belongs to the category) or -1 (if it does not belong to the category). For more complex classification problems, a multilayer

networ k, such as that trained by back propagation may be better as will be described in the next section.

Pattern association is another special form of a mapping problem, where in which the desired output is not just a "yes" or "no", but rather a pattern. *Associative memory* [17] is a neural network which is trained to associate a group of input vectors with a corresponding group of output vectors. If the desired output vector is the same as the input vector, the network is called an *auto-associative memory* [17]; moreover, if the output target vector is different from the input vector, the network is a *hetero-associative memory* [17]. Following training, an associative memory can recall a stored pattern when it is provided an input vector that is adequately similar to a vector it has learned. Multilayer neural networks can be trained to perform a nonlinear mapping from an n-dimensional space of input vectors (n-tuples) to an m-dimensional output space—i.e., the output vectors are m-tuples.[22]

On the other hand, in unsupervised training, self-organizing neural networks [22] group similar input vectors together without using training data to specify what a typical member of each group looks like or to which group each vector belongs. A series of input vectors is provided, but no target vectors are specified. The network adjusts the weights so that the most similar input vectors are assigned to the same output (or cluster) unit. Hence, the neural network will produce an exemplar (representative) vector for each cluster formed.

2.1.2 Backpropagation Neural Net

In the 1970s, there was a decline of interest in neural networks due to the illustration of the limitations of single-layer neural networks. The discovery and

extensive spreading of an effective general method for training a multilayer neural network [23-26] played a major role in the comeback of neural networks as a tool for solving a wide variety of problems.

The backpropagation network is a multilayer feedforward network trained by backpropagation which can be used to solve problems in many areas. Applications using such networks can be found in almost any area that uses neural networks to solve problems involving mapping a given set of inputs to a particular set of target outputs, i.e. networks that use supervised training. The aim in most neural networks is to train the network to attain a balance between the capability to respond correctly to the input patterns that are used for training (memorization) and the capability to give reasonable (good) responses to input that is similar, but not identical, to that used in training (generalization). [17]

Training a network with backpropagation comprises of three stages: the feedforward of the input training pattern, the calculation and backpropagation of the associated error, and the adjustment of the weights [17]. Subsequent to training, application of the network involves only the computations of the feedforward phase. A trained network can produce its output very fast even if training is slow. While a single-layer network is very limited in the mappings it can learn, a multilayer network (with one or more hidden layers) can learn any continuous mapping to any desired accuracy. For some applications more than one hidden layer may be beneficial, however one hidden layer is usually adequate [22].

A multilayer neural network with one layer of hidden units (the Z units) is shown in Figure 4. The output neurons (Y neurons) and the hidden neurons (Z neurons) may also

have biases. The bias on a standard output unit Y_k is denoted by W_{0k}; the bias on a typical hidden unit Z_y is denoted V_{0j}. These bias terms function like weights on connections from units whose output is always 1 [17]. Only the direction of information flow for the feedforward phase is shown. During the backpropagation phase of learning, signals are sent in the reverse direction. The algorithm in APPENDIX A is presented for one hidden layer, which is adequate for a large number of applications.

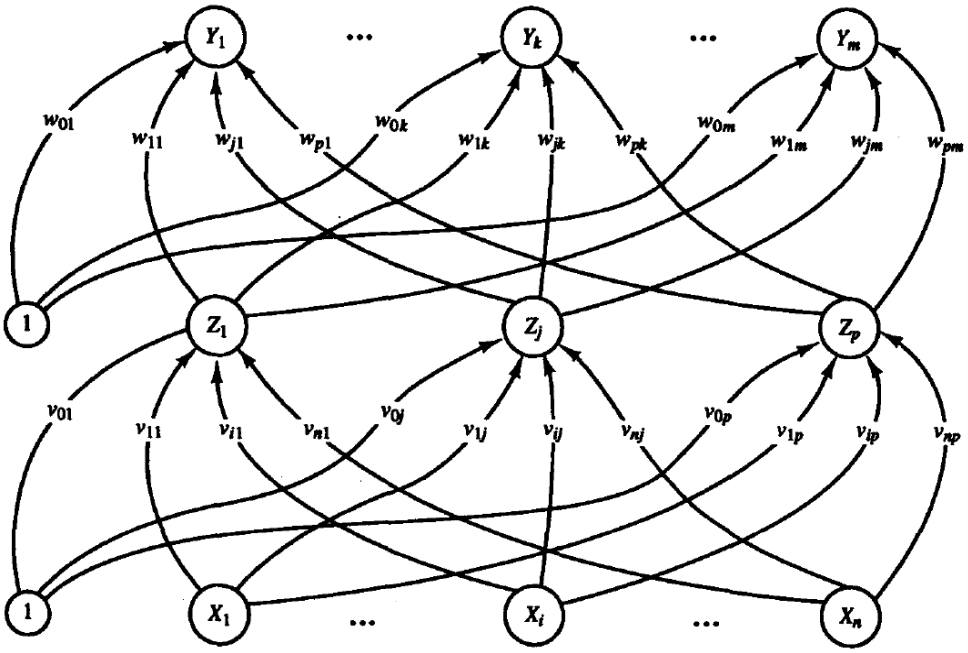


Figure 4: Backpropagation neural network with one hidden layer.[17]

An activation function for a backpropagation network should have several important characteristics: It should be continuous, differentiable, and monotonically non-decreasing. In addition, for computational efficiency, it is beneficial that its derivative be easy to compute. For the most frequently used activation functions, some of which illustrated in Table 1, the value of the derivative, at a particular value of the independent variable, can be denoted in terms of the value of the function (at that value of the independent variable). Typically, the function is expected to saturate, i.e., approach finite

maximum and minimum values asymptotically [17]. The binary sigmoid function, illustrated in Table 1 (b) is one of the most typical activation functions; another common activation function is the bipolar sigmoid function (Table 1 (c)). Note that the bipolar sigmoid function is closely related to the hyperbolic function (Table 1(d)).

The mathematical basis for the backpropagation algorithm is the optimization technique called the *gradient descent*. The gradient of a function (in the case of backpropagation, the function is the error and the variables are the weights of the network) gives the direction in which the function increases more rapidly; the negative value of the gradient gives the direction in which the function decreases most rapidly [27]. The derivation clarifies the reason why the weight updates described in APPENDIX A should be done after all of the δ_k and δ_j expressions have been calculated, rather than during backpropagation.

2.2 Feature Extraction

In pattern recognition and image processing, *feature extraction* is a particular type of dimensionality reduction.

When the data is too large to be processed by an algorithm and it is also suspected to be extremely redundant, the input data can be transformed into a reduced representation set of features (also called features vector) by *feature extraction* methods. If the extracted features are carefully chosen, it is expected that the features set will extract the important information from the data in order to perform the desired task with this reduced representation instead of the entire data.

Feature extraction comprises of reducing the amount of resources required to describe a large data set. One of the major problems when performing analysis of

complex data rises from the number of variables involved. Analysis with a large number of variables typically requires large amount of memory and computation power or a classification algorithm which usually over fits the training samples and generalizes poorly to new patterns. Feature extraction is used as a general term for methods for constructing combinations of variables to get around these problems while still describing the data with sufficient accuracy.

Best results are attained when an expert constructs a set of application-dependent features. Nonetheless, if no such expert knowledge is available general dimensionality reduction techniques may be of assistance [28-34].

2.2.1 Principal Component Analysis (PCA)

The most commonly used approach for extracting features from a set of observed variables is perhaps Principal Components Analysis (PCA). PCA is a mathematical procedure where an orthogonal transformation is used to convert a set of observations of possibly correlated variables into a set of values of uncorrelated variables known as *principal components* [34,35]. The number of extracted principal components is less than or equal to the number of original variables. The transformation of the data is defined in such a way that the first principal component has the highest variance possible i.e., constitutes as much of the variability in the data as possible. Moreover, each subsequent component in turn has as high a variance as possible under the constraint that it be orthogonal to (uncorrelated with) the preceding components. If the data set is jointly normally distributed, principal components are guaranteed to be independent. However, PCA is sensitive to the relative scaling of the original variables.[36]

PCA was invented in 1901 by Karl Pearson [37]. PCA has a wide range of applications some of which include data compression, image processing, visualization, exploratory data analysis, pattern recognition, and time series prediction. A complete discussion of PCA can be found in [22,38]. Usually after mean centering the data for each attribute, PCA can be achieved by eigenvalue decomposition of a data covariance matrix or singular value decomposition of a data matrix. Typically, the results of a PCA are discussed in terms of component scores (the transformed variable values corresponding to a particular case in the data) and loadings (the weight by which each standardized original variable should be multiplied to get the component score). [36,39]

The popularity of PCA appears from three important assets. First, it is the optimal (in terms of mean squared error) linear scheme for compressing a set of high dimensional vectors into a set of lower dimensional vectors and then reconstructing the original set. Second, the model parameters can be computed directly from the data - for example by diagonalizing the sample covariance matrix. Third, given the model parameters, compression and decompression are simple operations to perform where they require only matrix multiplication. [36,39,40]

Perhaps, PCA's operation is better thought as exposing the internal structure of the data in a way which best explains the variance in the data. If a multivariate dataset is visualised as a set of coordinates in a high-dimensional data space (1 axis per variable), PCA can supply the user with a lower-dimensional picture [36,39]. This is achieved by using only the first few principal components so that the dimensionality of the transformed data is reduced.

PCA is mathematically defined as an orthogonal linear transformation [40] which transforms the data to a new coordinate system such that the largest variance by any projection of the data comes to lie on the first coordinate (known as the first principal component), the second largest variance on the second coordinate, and so on.

To calculate the principal components we define a data matrix, X^T, with zero empirical mean (the sample mean of the distribution is subtracted from the data set), where each of the n rows stands for a different repetition of the experiment, and each of the m columns provides a particular kind of datum e.g., the results from a particular probe. The singular value decomposition of X is $X = W\Sigma V^T$, where the $m \times m$ matrix W is the matrix of eigenvectors of $X^T X$, the matrix Σ is an $m \times n$ rectangular diagonal matrix with nonnegative real numbers on the diagonal, and the $n \times n$ matrix V is the matrix of eigenvectors of $X^T X$. The PCA transformation that preserves dimensionality i.e., gives the same number of principal components as original variables is then given by:

$$Y^T = X^T W$$

$$= V\Sigma^T$$

In the usual case when $M < n - 1$, V is not uniquely defined. However, Y will usually still be uniquely defined. Since W (by definition of the SVD of a real matrix [41]) is an orthogonal matrix where each row of Y^T is simply a rotation of the corresponding row of X^T. The first column of Y^T is created from the *scores* of the instances with respect to the *principal* component; the next column has the scores with respect to the *second principal* component, and so on.

If a reduced-dimensionality representation is required, we can project X down into the reduced space defined by only the first L singular vectors, W_L:

$$Y = W_L^T X = \Sigma_L V_L^T$$

The matrix W of singular vectors of X is equivalently the matrix W of eigenvectors of the matrix of observed covariance $C = XX^T$,

$$XX^T = W\Sigma\Sigma^T W^T$$

The first principal component corresponds to a line that passes through the multidimensional mean and minimizes the sum of squares of the distances of the points from the line provided a set of points in Euclidean space. The second principal component relates to the same concept after all correlation with the first principal component has been subtracted out from the points. The singular values (in Σ) are the square roots of the eigenvalues of the matrix XX^T. Each eigenvalue is proportional to the portion of the *variance* that is correlated with each eigenvector. More correctly they are proportional to the portion of the sum of the squared distances of the points from their multidimensional mean. The sum of all the eigenvalues is equal to the sum of the squared distances of the points from their multidimensional mean. Basically, PCA rotates the set of points around their mean in order to align with the principal components. This moves as much of the variance as possible, by using an orthogonal transformation, into the first few dimensions. Therefore, the values in the remaining dimensions tend to be small and may be ignored with minimal loss of information. PCA is often used in this manner for dimensionality reduction. Therefore, it has the distinction of being the optimal orthogonal transformation for keeping the subspace that has largest *variance*. [35,36,40]

2.2.2 Principal Component Neural Networks (PCNN)

Since the original work of Oja and his research group, principal component analysis by neural networks and its extensions have become an important research field (a partial list of references is given by [41-47]) both for the interesting implications on unsupervised learning theory and applications to neural information processing [48].

The algorithms considered in this section are based on Oja's principal component neuron described by $z(t) = \boldsymbol{q}^T(t)\boldsymbol{x}(t)$, where $\boldsymbol{x}(t) \in \mathcal{R}^p$ represents the stationary multivariate random process whose first principal component is looked for, $\boldsymbol{q}(t) \in \mathcal{R}^p$ is the neuron's weight vector, and $z(t) \in \mathcal{R}$ is the neuron's output signal. Oja's learning rule [47] is:

$$\boldsymbol{q}(t+1) = \eta \boldsymbol{x}(t)z(t) + \boldsymbol{q}(t)[1 - \eta z^2(t)]$$

where η is a small learning rate and t indicates discrete time. This expression clearly reveals the presence of the Hebbian term $+\boldsymbol{x}(t)z(t)$ [49] and of a stabilizing term, thus it is also referred to as stabilized Hebbian learning equation.

The Generalized Hebbian Algorithm by Sanger [49] is one among the best known learning algorithms that allow a linear neural network to extract a selected number of principal components from a stationary or quasi-stationary multivariate random process. It applies to a single-layered feedforward neural network described by $\boldsymbol{z}(t) = \boldsymbol{Q}^T(t)\boldsymbol{x}(t)$, where $\boldsymbol{x}(t) \in \mathcal{R}^p$, $\boldsymbol{z}(t) \in \mathcal{R}^m$, thus $\boldsymbol{Q}(t) \in \mathcal{R}^{p \times m}$. The GHA rule writes:

$$\boldsymbol{Q}(t+1) = \mu \boldsymbol{x}(t)\boldsymbol{z}^T(t) + \boldsymbol{Q}(t)(\boldsymbol{I}_m - \mu LT[\boldsymbol{z}(t)\boldsymbol{z}^T(t)])$$

where m is a small positive learning rate, the operator $LT[\cdot]$ returns the lower-triangular part of the matrix contained within, and $\boldsymbol{I}_m$ denotes the identity matrix of size m. This

rule is an extension of Oja's rule, where the neurons are forced to encode different features by means of intrinsic Gram-Schmidt orthogonalization [50].

Kung and Diamantaras developed a learning rule (Laterally-Connected Network and Apex Rule) for Rubner-Tavan's principal component neural network [51] described by the following input-output relationships:

$$\boldsymbol{z}(t) = \boldsymbol{Q}^T(t)\boldsymbol{x}(t),$$

$$\boldsymbol{y}(t) = \boldsymbol{z}(t) + \boldsymbol{H}^T(t)\boldsymbol{y}(t).$$

where the input vector $\boldsymbol{x}(t) \in \mathcal{R}^p$, the output vector $\boldsymbol{y}(t) \in \mathcal{R}^m$ (with $m \leq p$, arbitrarily fixed), the direct-connection weight-matrix $\boldsymbol{Q}(t) \in \mathcal{R}^{p\times m}$ and the lateral-connection strictly upper-triangular weight-matrix $\boldsymbol{H}(t) \in \mathcal{R}^{m\times m}$ are evaluated at the same time. The Kung-Diamantaras' APEX learning rule for the weight-matrix $\boldsymbol{Q}$ and the inhibitory weight-matrix $\boldsymbol{H}$ recasts from [48] in matrix notation:

$$\boldsymbol{Q}(t+1) = \mu\boldsymbol{X}(t)\widetilde{\boldsymbol{Y}}(t) + \boldsymbol{Q}(t)\big[\boldsymbol{I}_m - \mu\widetilde{\boldsymbol{Y}}^2(t)\big]$$

$$\boldsymbol{H}(t+1) = -\mu SUT\big[\boldsymbol{Y}(t)\widetilde{\boldsymbol{Y}}(t)\big] + \boldsymbol{H}(t)[\boldsymbol{I}_m - \mu\widetilde{\boldsymbol{Y}}^2(t)]$$

where m is a small positive learning rate, matrices $\boldsymbol{X} \in \mathcal{R}^{p\times m}$, $\boldsymbol{Y} \in \mathcal{R}^{m\times m}$, and $\widetilde{\boldsymbol{Y}} \in \mathcal{R}^{m\times m}$ are defined by:

$$\boldsymbol{X} \triangleq \underbrace{[\mathbf{x}\,\mathbf{x}\cdots\mathbf{x}]}_{m}, \boldsymbol{Y} \triangleq \underbrace{[\mathbf{y}\,\mathbf{y}\cdots\mathbf{y}]}_{m}, \widetilde{\boldsymbol{Y}} \triangleq \mathrm{diag}(\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_\mathrm{m})$$

and operator $SUT[\cdot]$ returns the strictly upper-triangular part of the matrix contained within. Kung-Diamantaras' rule has been heuristically derived by applying Oja's rule to direct-connection weight-vectors, and its anti-Hebbian version to lateral-connection weight-vectors. [48]

2.2.3 Non-negative Matrix Factorization (NMF)

Unsupervised learning algorithms such as principal component analysis can be known as factorizing a data matrix subject to different constraints. Based on the employed constraints, the resulting features can be shown to have very different representational properties. [34,52,53].

NMF is described as to find non-negative matrix factors W and H, given a non-negative matrix V, such that:

$$V \approx WH$$

NMF can be used for the statistical analysis of multivariate data in the following approach: Given a series of multivariate n-dimensional data vectors, the vectors are placed in the columns of an $n \times m$ matrix V where m is the number of samples in the dataset. Matrix V is then approximately factorized into an $n \times r$ matrix W and an $r \times m$ matrix H. Typically, r is chosen to be smaller than n or m, so that W and H are smaller than the original matrix V. This results in a compressed form of the original data matrix [53].

The significance of approximating $V \approx WH$ is that it can be rewritten column by column as $v \approx Wh$, where v and h are the corresponding columns of V and H. More formally, each data vector v is approximated by a linear combination of the columns of W, which is weighted by the components of h. Hence, W can be considered as to contain a basis that is optimized for the linear approximation of the data in V. Since relatively small amount of basis vectors are used to represent many data vectors, high quality approximation can only be attained if the basis vectors discover structure that is latent in the data [34,52,53].

2.3 Support Vector Machine (SVM) Classification

Classification, in machine learning and pattern recognition, refers to an algorithmic procedure for assigning a set of input data to one of a given number of categories [15]. An example would be predicting the species of a flower given petal and sepal measurements [54]. An algorithm that implements classification, particularly in a solid implementation, is called a classifier [15]. The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm, which maps input data to a category.

Typically, classification refers to a supervised procedure, i.e. a procedure that learns to classify new samples based on learning from a training set of instances that have been properly labelled with the correct classes by hand. The corresponding unsupervised method is known as clustering. This procedure involves grouping data into different classes based on some measure of inherent similarity [14] (e.g. the distance between instances, considered as vectors in a multi-dimensional vector space).

The support vector machine (SVM) [55-58] is a training algorithm for learning classification and regression rules from data. For instance SVM can be used to learn polynomial, radial basis function (RBF) and multi-layer perceptron (MLP) classifiers [58]. In the 1960s, Vapnik first suggested SVMs for classification. Recently, support vector machines have become an area of extreme research mainly because of the developments in the techniques and theory joined with extensions to regression and density estimation.

SVMs were born from statistical learning theory [57]; the goal was to solve only the problem of interest without having to solve a more difficult problem as an

intermediate step [58]. SVMs are based on the structural risk minimisation principle which is closely related to regularisation theory. This principle features capacity control to prevent over-fitting and therefore is a partial solution to the bias-variance trade-off dilemma [59].

In the implementation of SVM, there are two main components; techniques of mathematical programming and kernel functions. The parameters are found by solving a quadratic programming problem with linear equality and inequality constraints; rather than by solving a non-convex, unconstrained optimisation problem [56]. SVM is able to search a wide variety of hypothesis due to the flexibility of the kernel functions. The geometrical interpretation of support vector classification (SVC) is that the algorithm searches for the optimal separating surface (hyperplane) that is, in a sense, equidistant from the two classes [55] (see Figure 5). Statistical properties of this optimal separating hyperplane are available at [57].

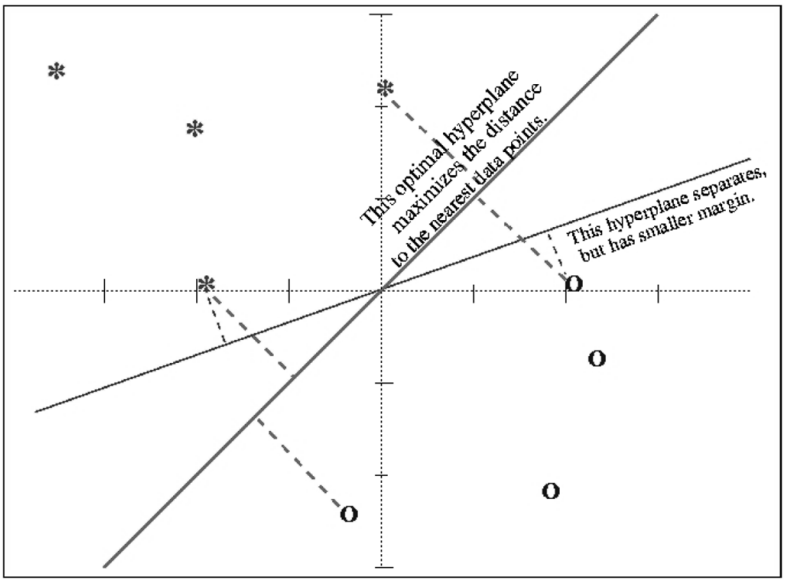


Figure 5: Choosing the hyperplane that maximizes the margin [55]

Without requiring a separate validation set during training, the SVM parameters can be optimized using generalisation theory. As SVM is based on solid statistical and mathematical foundations concerning generalisation and optimisation theory, hence, it has been proven to outperform existing techniques on a wide variety of real world problems [56]. SVMs and related methods are also being increasingly applied to real world data mining. An up-to-date list of such applications can be found at [60].

2.4 Mobile Robot Navigation and Obstacle Avoidance

In the past few years, mobile robots have been widely used in various fields, such as space exploration, industrial and military industries, under water survey, and service and medical applications, hence attracting the attention from researchers. Mobile robots require the capabilities of autonomy and intelligence, therefore, researchers are forced to deal with important issues such as uncertainty (in both sensing and action), reliability, and real-time response [61]. As a result, one of the major challenges in robotics is designing algorithms to allow the robots to function autonomously in unstructured, dynamic, partially observable, and uncertain environments [62]. Figure 6 shows the position of motion control (for obstacle avoidance) and exploration (navigation) compared to other mobile robot research areas.

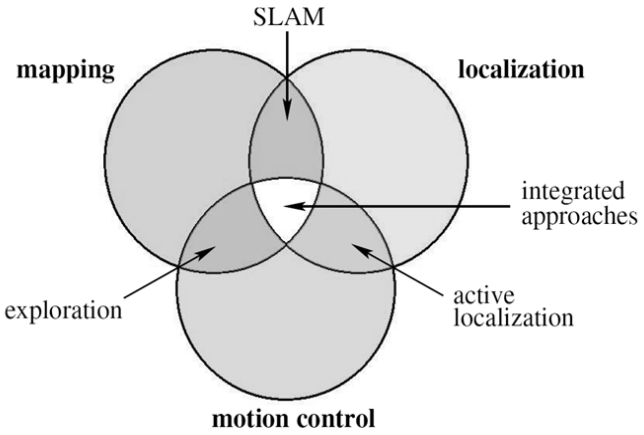


Figure 6: Relationships between mobile robot research areas. [63]

The problem of mobile robot navigation, includes three fundamental matters; map building, localization and path planning. This problem refers to planning a path to a specified target, executing this plan based on sensor readings, and is the key to the robot performing some particular tasks. Artificial Neural networks are increasingly being used in various fields of machine learning, including pattern recognition, speech production and recognition, signal processing, medicine, and business. In the recent years, artificial neural networks, including feedforward neural network, self-organizing neural network, principal component analysis (PCA), dynamic neural network, support vector machines (SVM), neuro-fuzzy approach, etc., have been extensively employed in the field of mobile robot navigation because of their properties such as nonlinear mapping, ability to learn from examples, good generalization performance, massively parallel processing, and ability to approximate an arbitrary function given sufficient number of neurons [16,17].

2.4.1 Neural Networks for Interpretation of the Sensor Data

For a mobile robot to identify where it is or how it got there, or to be able to reason about where it has gone, sensors are necessary. For measuring the distance that wheels have traveled along the ground and for measuring inertial changes and external structure in the environment, the sensors can be flexible and mobile. The sensors can be generally divided into two categories: internal state sensors, and external state sensors. The internal state sensors such as accelerometers and gyroscopes, provide the internal information about the robot's movements. The external state sensors, such as laser, infrared sensors, sonar, and visual sensors, provide the external information about the environment. The data from internal state sensors can be used to estimate the position of the robot in a 2-dimensional space; however, cumulative error is inevitable. The data from external state sensors can be applied for recognizing a place or a situation, or be used to construct a map of the environment. Laser, infrared, and sonar sensors can obtain distant and directional information about an object. Visual sensors can also provide rich information of the environment, but can be very expensive to process. In most cases, because of the available noises, the sensor readings are imprecise and unreliable. Thus, it is inevitable for the mobile robot navigation algorithm to process the sensor data with noises. Given that neural networks have many processing units, each with primarily local connections, they may provide some degree of robustness or fault tolerance for interpretation of the sensor data [16].

Feedforward multi-layer perception neural network, trained by the back-propagation algorithm, has been applied for pattern classification, pattern recognition and function approximation. In [64], Thrun has employed a feedforward neural network to

"translate" the readings of sonar sensors into occupancy values of each grid cell for building metric maps. Meng and Kak proposed a NEURO-NAV system for mobile robot navigation [65]. In the NEURO-NAV, in order to drive the robot to move in the middle of the hallway, a feedforward neural network, which is driven by the cells of the Hough transformation of the corridor guidelines in the camera image, is used to obtain the approximate relative angles between the heading direction of the robot and the orientation of the hallway [65]. self-organizing Kohonen neural networks are well known for their capability to carry out classification, recognition, data compression and association in an unsupervised manner [66]. In [67], self-organizing Kohonen neural networks are applied to recognize landmarks using the measurements from laser sensors in order to provide coordinates of the landmarks for triangulation.

As mentioned before, PCA is a statistical technique, which has been applied to machine learning fields such as data compression and pattern recognition, and is known as one of the effective techniques to extract the principal features from high-dimension data and reduce the dimension of the data [40]. In [68], Vlassis et al. presented an approach for mobile robot localization where PCA was used to reduce the dimensions of sonar sensor data. Crowley et al. proposed an approach to estimate the position of a mobile robot based on the PCA of laser ranger sensor data [69]. In [70,71], PCA has been used to extract features of images for mobile robot localization. PCA Neural Network (PCNN) was applied for navigation to determine the "free space" in front of a mobile robot using ultrasound range finder data in order to construct a collision-free path for the mobile robot in [18].

2.4.2 Neural Networks for Obstacle Avoidance

There are always static, as well as non-static obstacles in the environment. Hence, robots need to autonomously navigate themselves in environments by avoiding obstacles. The neural networks, which have been designed for obstacle avoidance by mobile robots, should take the sensor data from the environment as their inputs, and output the direction for the robot to proceed. In [72], Fujii et al. presented a multilayered neural network model through reinforcement learning for collision avoidance of a mobile robot. Silva et al. proposed the MONODA (MOdular Network for Obstacle Detection and Avoidance) architecture for obstacle avoidance and detection of a mobile robot in unknown environments [8]. This model consists of four three-layered feedforward neural network modules where each module detects the probability of obstacles in one direction of the robot. In [73], Ishii et al. developed an obstacle avoidance method based on self-organizing Kohonen neural networks for underwater vehicles. Gaudiano and Chang proposed an approach for obstacle avoidance by employing a neural network model of classical and operant conditioning based on Grossberg's conditioning circuit [7,74]. Parhi and Singh introduced a real-time obstacle avoidance approach to solve each of the target-seeking, obstacle-avoidance, and wall-following tasks using separate neural networks [19-21]. In their approach, based on certain criteria one of the networks is selected at each time step to control the mobile robot allowing it to move safely in a crowded real-world and unknown environment and to reach a specified target while avoiding static as well as dynamic obstacles.

2.4.3 Neural Networks for Path Planning

The path planning problem may consist of two sub-problems; path generation and path tracking. This problem refers to determining a path between an initial pose of the mobile robot and a final pose such that the robot does not collide with any objects in the environment and that the planned motion is consistent with the kinematic constraints of the vehicle. The existing path planning methods include A* algorithm [75], potential fields [2], and methods using intelligent control technique such as neural networks and neuro-fuzzy. Methods using intelligent control do not plan global paths for mobile robots and can be employed in unknown environments. The input pattern of the neural networks employed for path planning of mobile robots should consider the following data: robot's actual position and velocities; robot's previous positions and velocities; target position and sensor data, and then output commands to drive the robot to follow a path towards the target by avoiding obstacles according to these data [16].

Kozakiewicz and Ejiri have used a human expert to train a feedforward neural network that reads inputs from a camera and outputs the appropriate commands to the actuators [76]. In [77], Sfeir et al. presented a path generation technique for mobile robot using memory neuron network proposed by Sastry et al. [78]. The memory neuron network is a feedforward neural network that uses memory neurons. A memory neuron is a combination of a classic perception and unit delays, which gives the network memory abilities. If a mobile robot is totally insensitive to context, it will often get trapped in oscillations in front of wide objects. Pal and Kar employed a notion of memory into the network to overcome the oscillation problem [79]. In [6], Glasius, Komoda, and Gielen presented a Hopfield-type neural network for dynamic trajectory formation without

learning. Fierro and Lewis proposed a control structure which integrates a kinematic controller with a feedforward neural network computed-torque controller for non-holonomic mobile robots, where the neural network weights are adjusted on-line, with no "off-line learning phase" needed [80-82]. Yang and Meng studied a biologically inspired neural network approach for motion planning of mobile robots [83-85]. This model is inspired by Hodgkin and Huxley's membrane model [86] for a biological neural system and Grossberg's shutting model [87]. The proposed model by Yang and Meng plans motions for mobile robots without any prior knowledge of the environment, without explicitly searching over the free workspace or the collision path, and without any learning procedure.

Chapter 3

Design and Methodology

As reviewed in chapter2, Janglová [18] introduced an intelligent controller for solving the motion-planning problem in mobile robots using two neural networks. The first neural network is a modified principal component analysis network (Figure 7(a)) used to extract the features (V_i segments) of the workspace determining the *free space* using the data from ultrasound range finders (d_i) as shown in Figure 7(b). These segments (V_i) are used as inputs to the second neural network along with direction of the goal (S_i). The second network is a multilayer perceptron (Figure 8), which successfully finds a safe direction (O_i), from the segments extracted from the first network, for the robot's next step to navigate towards the target in a collision-free environment while avoiding obstacles.

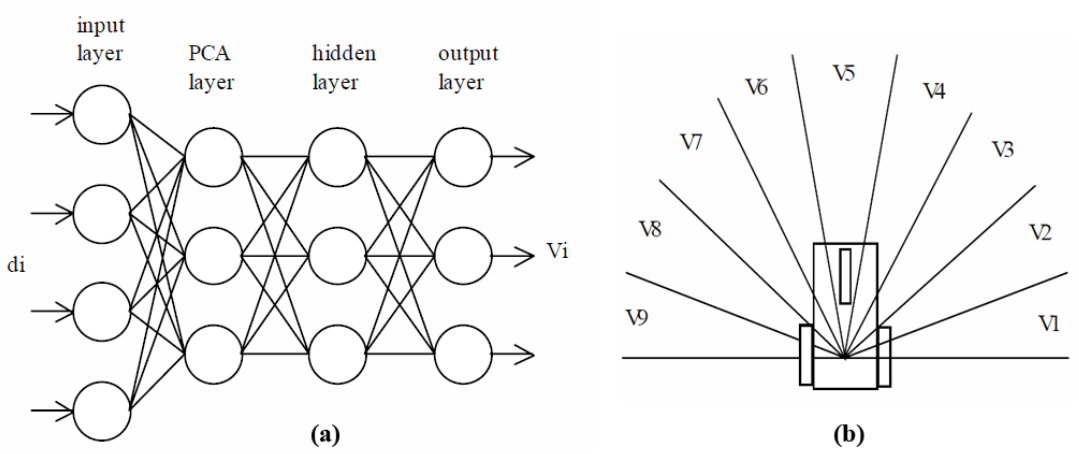


Figure 7: a) Modified PCA Neural Network Topology. b) Workspace Segments. [18]

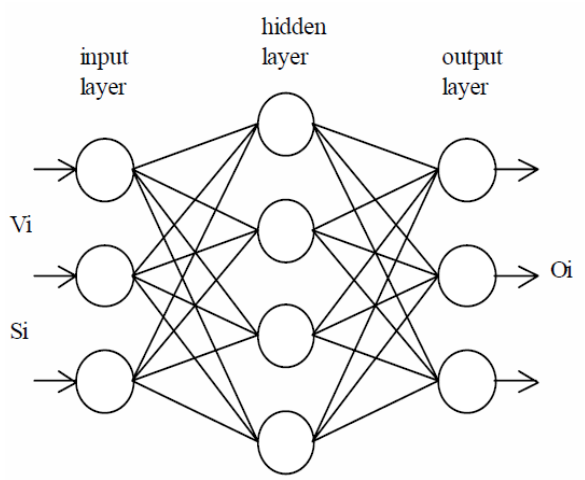


Figure 8: Topology of Multi-Layer Perceptron. [18]

Parhi and Singh [20,21] proposed a real-time obstacle avoidance method to solve the main problems of navigation using three neural networks. This approach was later improved by them to optimize the path of the mobile robot [19]. In their approach, three identical four-layer feedforward neural networks have been used (see Figure 9). Each network is trained separately with different training samples so that each network can solve one of the problems of navigation; target-seeking, object-avoidance, or wall-following. The inputs to their proposed neural controller consist of the signals from the sensors (in this case, the distance from the left, right and front obstacle with respect to the robot's position) and the direction of the target (goal). The output of the networks is the steering angle which provides real-time collision-free motion planning for mobile robots in a real world dynamic environment. The neural networks are trained by presenting them with 200 patterns representing typical scenarios.

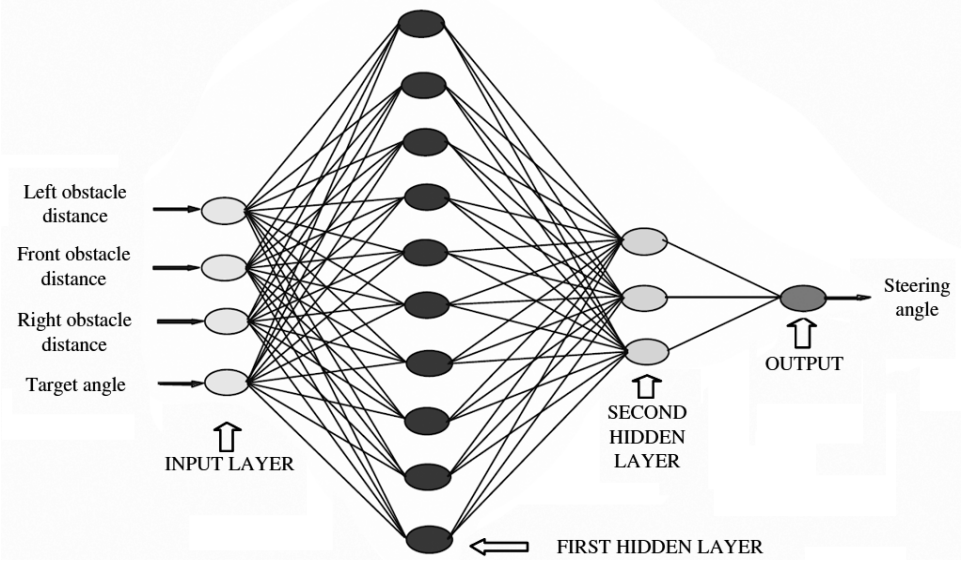


Figure 9: Four-layer neural network for robot navigation. [20,21]

In [8] Silva et al. presented the MONODA (modular network for obstacle detection and avoidance) architecture for obstacle detection and avoidance for controlling the NOMAD autonomous mobile robot in an unknown environment (see Figure 10). As depicted in Figure 11(c), this model consists of four three-layered feedforward neural network modules (each detects the probability of obstacle in one direction of the robot). The convention in neural networks is to use architectures as small as possible to obtain better generalisation.

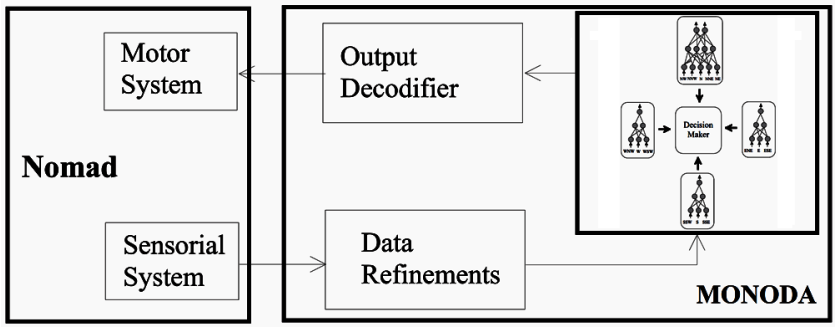


Figure 10: MONODA modular system. [8]

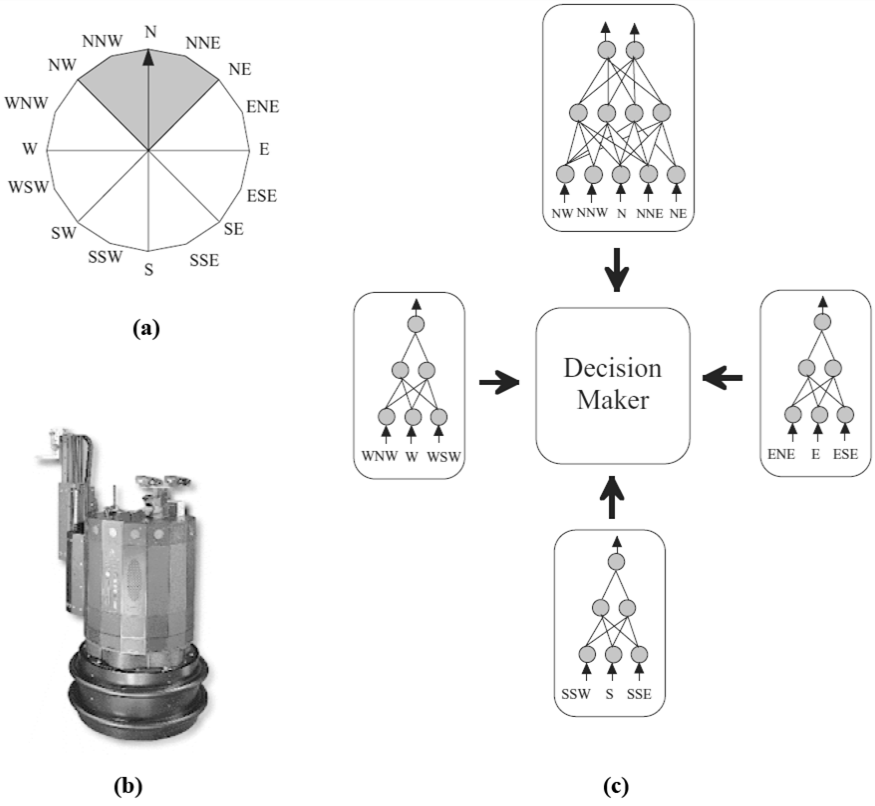


Figure 11: a) Localisation of Nomad 200™ sensors. b) Nomad mobile robot. c) Modular architecture. [8]

With modular networks generalisation is improved, because each one of the network modules is easier to train well. Due to the modular architecture, usually, the number of weights is less than in a fully connected MLP. Therefore, the overall training time of the networks is also significantly reduced.

3.1 Problem Statement

Although all the described methods have been successful to some extent in their specific applications, however, most of the current approaches address one particular type of sensor or robot platform. The main problem with neural network approaches is the training of the network. In supervised training collecting sufficient, yet valuable, samples from the environment to train the network can sometimes be frustrating and very time

consuming [17]. In addition, apart from the effort to collect valuable samples, the training time of a network can be significantly high [17].

Moreover, in any neural network navigation algorithm, if the robot platform or the type or number of the sensors are changed or altered, the network architecture requires some modifications to accommodate with the new amount of sensor data. In other words, same network architecture cannot be used for different robot platforms with dissimilar types or different numbers of range sensors. Therefore, these network structures will only be effective for the mobile robots that they have been designed for and are not extendable to other sensor configurations. For example, if a neural network is designed to have eight inputs from eight ultrasonic sensors, then this network is not operational for a robot which has only four ultrasonic sensors. Same situation occurs with different types of sensors. For instance, a network which is designed to function with one type of sensor (e.g. a laser scanner) cannot be applied to robots with other types of sensors (e.g. ultrasonic sensors). As a result, the structure of the network needs to be changed and new samples need to be gathered in order to accommodate with the new configurations.

In other words, in neural network navigation systems, the data from the sensors usually form the inputs of the network. Hence, if there are any changes to the sensors, the architecture of that network requires to be altered and the entire training process (collecting samples and training the network) has to be carried out all over again. Therefore, the problem with current neural network navigation approaches is that they cannot be extended to various robot platforms with different sensors.

To overcome this problem, so that a neural network algorithm can be employed for various robots and sensors, we propose a new method of sensor data interpretation.

We interpret data from different types of sensors in a general form and introduce a global neural network algorithm to perform the navigation task by using the interpreted data.

3.2 The Proposed Method

In this section, we propose an approach to address the problem of extending neural network navigation algorithms for various robots and sensors. In order to overcome this problem, we have introduced a new structure, illustrated in Figure 12, to interpret different types of 2-dimensional sensor data and a global navigation algorithm that can be applied to various types of sensors and robot platforms.

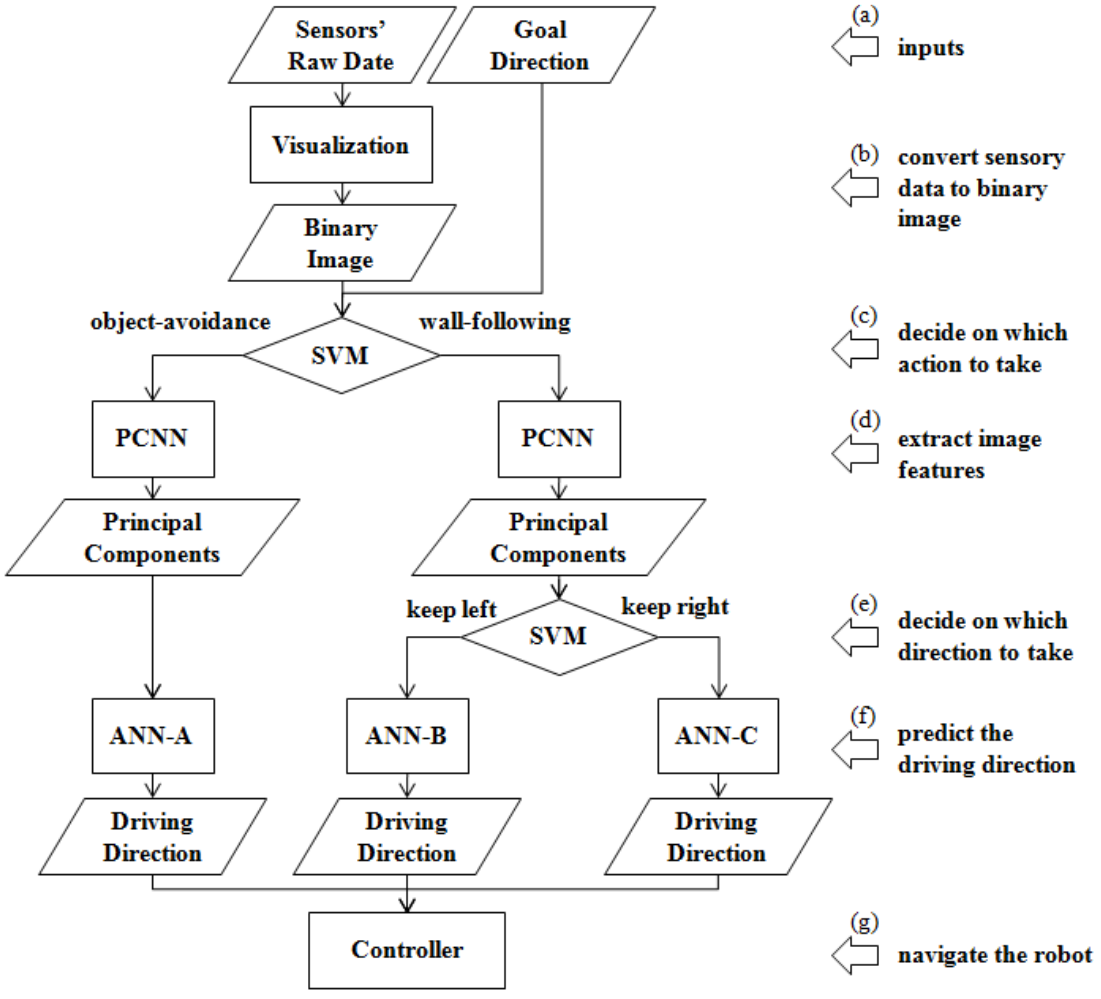


Figure 12: Flowchart of proposed method

To have a global algorithm that can be applied to all kinds of 2-dimensional range sensors, the sensory data needs to be interpreted in such way that same number of input units for the network can be extracted for different types or numbers of sensors. As mentioned earlier in this chapter, Janglová [18] introduced a feature extraction method employing PCNN to find the subspaces using sonar sensors. In addition, the MONODA architecture, proposed by Silva et al. [8], proved to reduce the overall training time and also improve generalization. However, if the number of sensors changes, then the structure of the networks introduced in these methods needs to be altered and training needs to be performed from the beginning. In order to solve this issue, before presenting the sensory data to the networks, we developed a general form of representation of the sensory data so that different types or number of 2-dimensional range sensors will have the same number of features. Therefore, we can design a network with constant number of inputs units for any type or number of sensors.

Parhi and Singh [19-21] successfully solved the navigation problem, by assigning different multilayer neural networks to solve each of the navigation tasks—target-seeking, object-avoidance, and wall-following. Figure 13 illustrates a simple example of these three scenarios. However, it is our belief that a multilayer neural network is capable of solving both target-seeking and object-avoidance tasks at the same time. Therefore, instead of using two separate networks, which will only lead to consuming more time and resources on training and gathering training samples, we introduce a structure which uses only one network for this purpose. Yet, the wall-following task will require a more complex structure to accommodate with both directions of rotation.

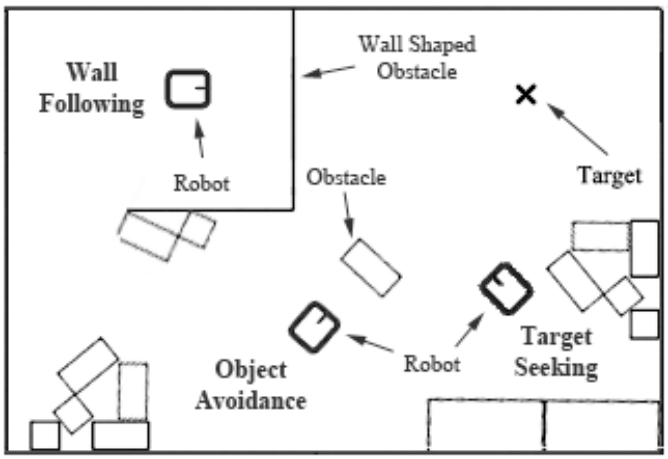


Figure 13: Example of scenarios of wall-following, object-avoidance and target-seeking tasks

For example, as illustrated in Figure 14, when the mobile robot encounters a wide object, or a wall, while moving towards the target, it needs to decide which direction to take (left or right) to get passed the object. In addition, if the object in front of the robot is a U-shaped object, then it will require keeping to only one direction, at all times, in order to safely navigate out of the U-shaped object. For Example, Figure 15 shows a robot's path which has performed a wall-following task after encountering a wall shaped obstacle. In this image we can see that the robot has kept the wall to its right at all times.

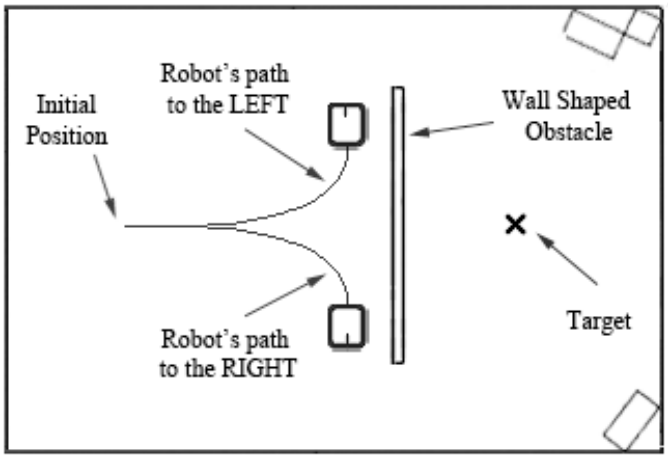


Figure 14: Paths showing two different directions chosen when encountering a wall.

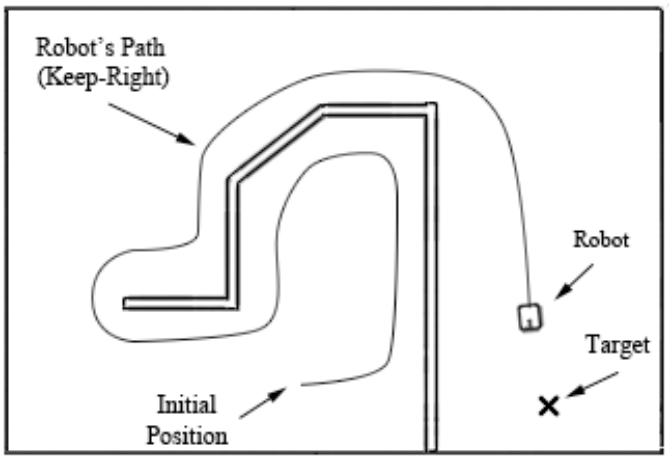


Figure 15: Path of a wall-following task performed by a mobile robot only in keep-left direction.

Therefore, we introduced two networks; one network for navigating while the wall is on the right side of the robot, and another for keeping the wall on the left side of the mobile robot.

We can divide our proposed method, illustrated in Figure 12, into five sections:

- Visualization (Figure 12(b))
- Dimensionality Reduction or Feature Extraction (Figure 12(d))
- Classification (Figure 12(c, e))
- Prediction (Figure 12(f))
- Controller (Figure 12(g)).

We introduce a method to generate a general representation of the raw sensory data for all types of 2-dimensional range sensors so that different types, or numbers, of sensors can be processed in the same way. The visualization algorithm is implemented to produce binary images, visualizing the distance of the obstacles in front of the mobile robot, with same dimensions for any type of 2-dimensional sensor. Therefore, at each time step, by employing the visualization algorithm, the readings of the sensors are converted into binary images. Subsequently, using the goal direction, an SVM classifier,

classifies the binary images to either *object-avoidance* or *wall-following*. In case of *object-avoidance* three scenarios can be portrayed. First, the robot is moving towards the target with no obstacles in its path (Figure 16 (a)). Second, the robot encounters a not very large obstacle while navigating towards the target (Figure 16 (b)). Finally, the target is in the range of the robot and there are no obstacles in the path of the robot to the target, so the robot can change its heading towards the target (Figure 16 (c)). The last scenario usually happens after the robot has navigated around an obstacle or a wall, now it needs to change its path back towards the target. An image is classified as *wall-following* only when the robot encounters a wall shaped, or a very large or wide, obstacle while navigating towards the target (Figure 16 (d)). However, in this scenario the robot needs to decide on which direction (left or right) to follow the wall. Hence, we use another SVM algorithm to classify the image patterns to *keep-left* or *keep-right* directions. If the former is selected, the robot moves alongside the wall while maintaining a safe distance from it and keeping the wall to the left side, and vice versa for *keep-right*.

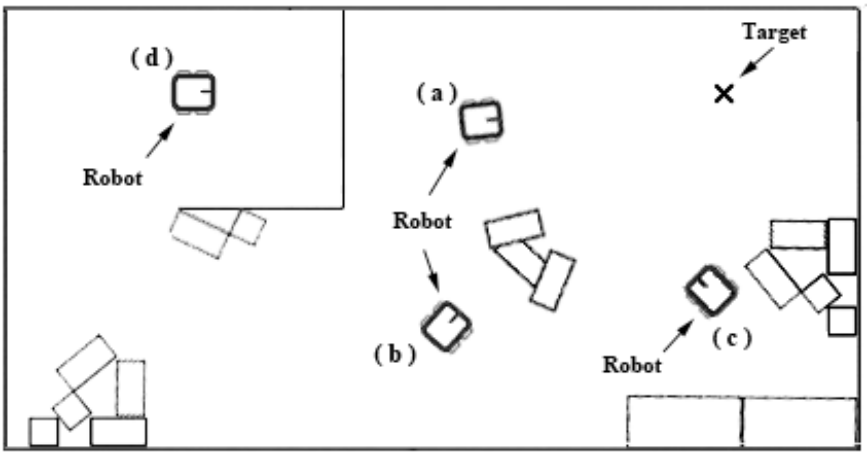


Figure 16: Examples of three different situations for object-avoidance (a, b and c) and a situation for wall-following (d).

Given that, multilayer neural networks can be very slow when introduced with too many input units [22], it is necessary to reduce the dimensions of the inputs. Therefore, before any other process can be done, the dimensions of the images need to be reduced. By using a feature extraction method like PCNN, we can extract as many features (principal components) required from the images. To keep the general representation for the patterns, we extract the same number of features from all of the images. Additionally, three similar multilayer neural networks are designed and separately trained using the supervised learning. The neural networks will provide driving directions for the robot to perform the aforementioned tasks. The driving direction is then passed to the controller to navigate the robot.

3.3 Explanations of Proposed Method

In section 3.2, we divided our proposed method to smaller subsections and explained each part very briefly. In this section a more detailed review of the proposed algorithm is provided.

The *visualization* section (Figure 12(b)) is the first and most essential section in our proposed method. It is very important to have a general representation of the raw sensory data for all types of 2-dimensional range sensors so that different types, or numbers, of sensors can be processed in the same manner. Therefore, if the networks are trained to perform the navigation task using one type of sensor, e.g. laser sensor, then they will also be able to extend to other types of sensors, such as sonar sensors, to navigate in the environment without requiring any further training. In this part, the data retrieved from the sensors are converted into binary images, visualizing the distance of the obstacles in front of the mobile robot. These generated images will have the same

dimensions regardless of the number of sensor inputs. For example, an image depicted from a laser range finder (SICK) (Figure 17 (a)) has the same dimensions (number of pixels) as an image created from 8 sonar sensors mounted in front of a robot (Figure 17 (b)). Therefore, this will allow us to have the same number of features for the inputs of the neural networks.

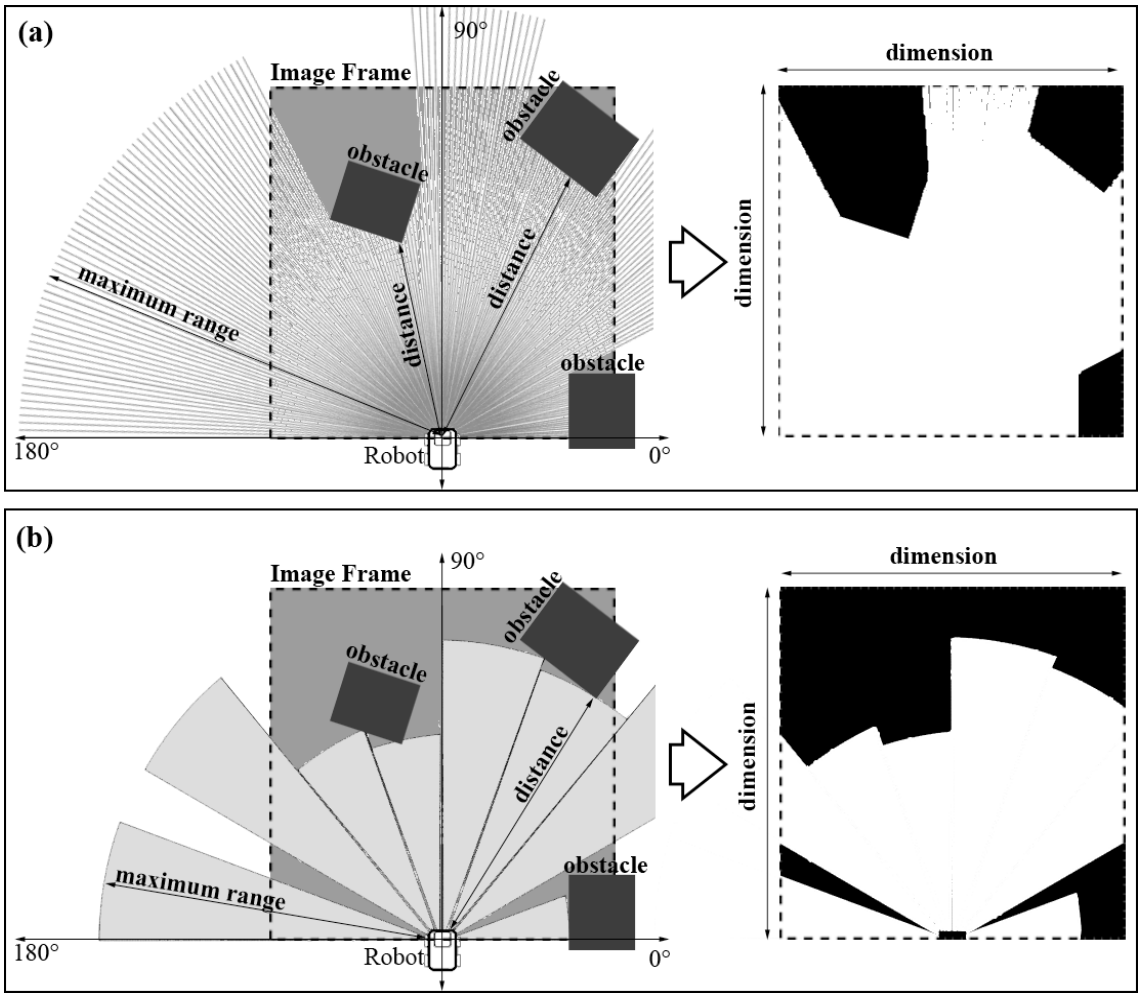


Figure 17: a) Example of visualizing the data from a laser range finder into a binary image. b) Example of visualizing the data from 8 sonar sensors into a binary image.

However, if we apply the plain images as the inputs by means of using each pixel as an input unit to the networks, the training process could be very slow due to the large

number of pixels. In a very simple scenario where we have images as small as 50×50 pixels (each pixel will be either black or white), the neural networks will have to process 2500 input units of zeros and ones which represent black and white pixels respectively. Since, multilayer neural networks can be very slow when introduced with too many input data [22], it is compulsory to reduce the dimensions of the inputs. This can be done by extracting the most relevant and important features from the images using feature extraction methods such as GHA, APEX or NMF. Therefore, the second part of the algorithm (Figure 12(d)) consists of a *dimensionality reduction* algorithm to reduce the dimensions of the images. One of the best known learning algorithms is the Generalized Hebbian Algorithm (GHA) [49], which can extract a selected number of principal components from a stationary or quasi-stationary multivariate random process. It applies to a single-layered feed-forward neural network described by $\boldsymbol{z}(t) = \boldsymbol{Q}^T(t)\boldsymbol{x}(t)$, where $\boldsymbol{x}(t) \in \mathcal{R}^p$, $\boldsymbol{z}(t) \in \mathcal{R}^m$, thus $\boldsymbol{Q}(t) \in \mathcal{R}^{p\times m}$. The GHA rule writes:

$$\boldsymbol{Q}(t+1) = \mu\boldsymbol{x}(t)\boldsymbol{z}^T(t) + \boldsymbol{Q}(t)(\boldsymbol{I}_m - \mu LT[\boldsymbol{z}(t)\boldsymbol{z}^T(t)])$$

where m is a small positive learning stepsize, the operator $LT[\cdot]$ returns the lower-triangular part of the matrix contained within, and $\boldsymbol{I}_m$ denotes the identity matrix of size m. In addition, to maintain the general representation for the patterns, we extract the same number of features from all of the images. However, there is always a trade off between speed and performance when deciding on the number of extracted features.

The *classification* (Figure 12(c,e)) and *prediction* (Figure 12(f)) sections compose our proposed navigation algorithm. Using the *classification* segments we decide on which action to take at each step. The first SVM classifier (Figure 12(c)) classifies the images as object-avoidance or wall-following action, given an image and the direction of the target.

If the wall-following action is selected, the other SVM classifier (Figure 12(e)) uses only the extracted features of the images to choose a direction to follow the wall.

However, the *prediction* section is the core of our proposed navigation algorithm. In which, three multilayer neural networks (Artificial Neural Networks A, B and C) are trained by using supervised learning with backpropagation algorithm to provide driving directions. The chosen number of layers and units in each layer were found empirically to facilitate training. These networks are trained only by samples generated using the laser range scanner and sonar sensors are used for validation and testing purposes only.

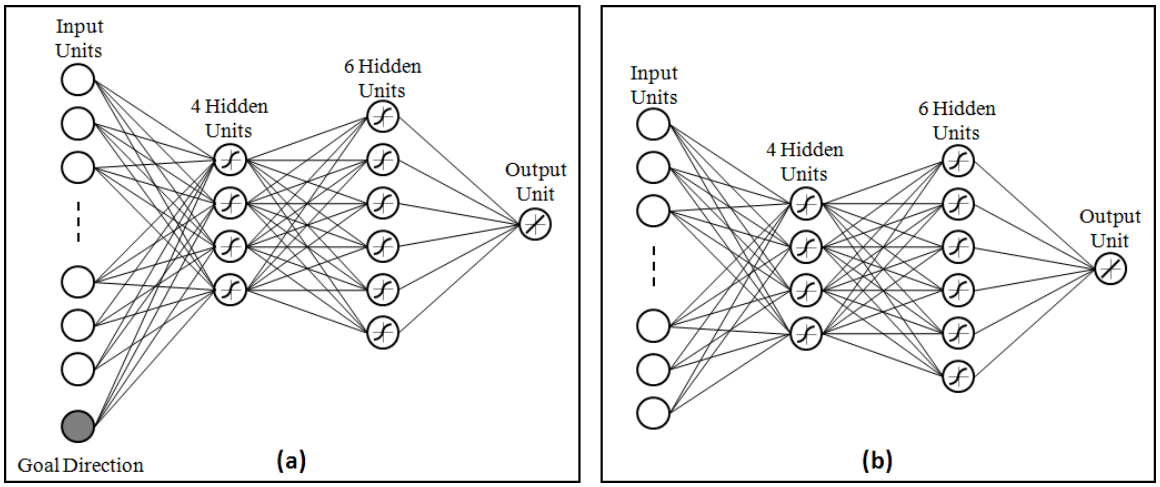


Figure 18: Proposed neural network architectures for a) ANN-A, b) ANN-B and ANN-C

In the object-avoidance module, an artificial neural network (Figure 18 (a) ANN-A) is trained to navigate the mobile robot towards the target while avoiding static as well as dynamic obstacles. This part of the algorithm will make sure that the mobile robot will safely reach its target with no collisions. Since, the network will provide driving directions at each sensor reading, this will provide a safe path for the robot to avoid both static and dynamic obstacles while maintaining the bearing of the target. During training

and normal operation, the input patterns provided to neural network A comprise the following components:

$y_n^{\{1\}}$ = Extracted features from the binary image.

$y_{n+1}^{\{1\}}$ = Target bearing from the robot's location.

where n is the number of extracted features. These input values are distributed to the hidden units which generate outputs by

$$y^{\{l\}} = f\left(V_j^{\{l\}}\right)$$

where

$$V_j^{\{l\}} = \sum_{\mathrm{i}} \mathrm{W}_{\mathrm{ji}}^{\{\mathrm{l}\}} . y_i^{\{l-1\}}$$

where l = layer number (2 or 3); j = label for jth unit in hidden layer $\{l\}$; i = label for ith unit in hidden layer $\{l-1\}$; $\mathrm{W}_{\mathrm{ji}}^{\{\mathrm{l}\}}$ = weight of the connection from unit i in layer $\{l-1\}$ to unit j in layer $\{l\}$; $f(\cdot)$ = activation function, chosen in this work as hyperbolic tangent sigmoid:

$$n = \frac{2}{1+e^{-2\times n}} - 1$$

During training, the network output θ_{actual} may differ from the desired output $\theta_{desired}$ as specified in the training pattern presented to the network. A measure of the performance of the network is the mean squared difference between $\theta_{desired}$ and θ_{actual} for the set of presented training patterns.

$$Err = \frac{1}{n}\sum_{i=1}^{n}(\theta_{desired} - \theta_{actual})^2$$

where n is the number of training patterns. The error is then back propagated to the previous layers using the backpropagation method to train the network. In order to determine the appropriate weight adjustments to reduce error, this method requires the computation of local error gradients. For the output layer, the error gradient $\delta^{\{4\}}$ is

$$\delta^{\{4\}} = f'\left(V_1^{\{4\}}\right)(\theta_{desired} - \theta_{actual})$$

The local gradient for units in the hidden layer $\{l\}$ is provided by

$$\delta_j^{\{l\}} = f'\left(V_j^{\{l\}}\right)\sum_k \delta_k^{\{l+1\}} W_{kj}^{\{l+1\}}$$

The synaptic weights are updated according to the following expressions

$$W_{ij}(t+1) = W_{ij}(t) + \Delta W_{ij}(t+1)$$

and

$$\Delta W_{ij}(t+1) = \alpha \Delta W_{ij}(t) + \Delta\, \eta \delta_j^{\{l\}} y_i^{\{l-1\}}$$

where $\alpha =$ momentum coefficient; $\eta =$ learning rate and $t =$ iteration number. Every iteration is composed of presentation of a training pattern and correction of the weights. The final output from the neural network is

$$\theta_{actual} = f(V_1^4)$$

where

$$V_1^4 = \sum_i W_{1i}^{\{4\}} y_i^{\{3\}}$$

However, this will only ensure a collision free path to the target, which will not provide a solution for navigating around deadlocks and U-shaped objects or for instance going from one room to another. Therefore, a wall-following algorithm is a necessity.

In the wall-following section of the algorithm, first the features of the sensory images are extracted. Then using a pre-trained SVM classifier (Figure 12(e)) a wall-following direction (keep-left or keep-right) is selected. In this section, two identical artificial neural networks (Figure 18 (b)) are separately trained. These will provide driving directions for keeping left or keeping right respectively at all times. The architecture of these two networks (ANN-B and ANN-C) differ from the *object-avoidance* network (ANN-A) in the input layer. During training and normal operation, the input patterns provided to the *wall-following* neural networks consist of only the extracted features from the binary image ($y_n^{\{1\}}$, n being the number of extracted features). In other words, these networks provide an output regardless of the direction of the goal. Therefore, allowing the robot to safely move out of U-shaped objects and other situations such as deadlocks without getting stuck. Besides the difference in the number of units in the input layer, the process for backpropagating the error and calculating the output also applies to these networks.

The output, which we will note as driving direction or the steering angle, from the neural networks is then passed to the *controller* (see Figure 12(g)). The *robot controller* calculates the velocity of the left wheel (V_{left}) and right wheel (V_{right}), and then sends commands to drive the robot. A positive output is translated as rotation to the right, and vice versa. Based on this assumption the velocities are calculated as

$$\frac{V_{right}}{V_{left}} = \frac{r_1}{r}$$

where V_{right} is the velocity of the right wheels, V_{left} is the velocity of the left wheels, r is the radius of the curvature of the outside wheels (in this case left wheels) of

the robot and $r_1 = r - r_2$ is the inside wheels curvature radius (see Figure 19). The radius of the curvature is determined from

$$r = \frac{r_2 \times x}{|\theta|}$$

where x is an arbitrary value, r_2 is the width of the robot, θ is the steering angle (driving direction). The width of the robot is known, therefore x is used to determine the total curvature desired for rotating the robot.

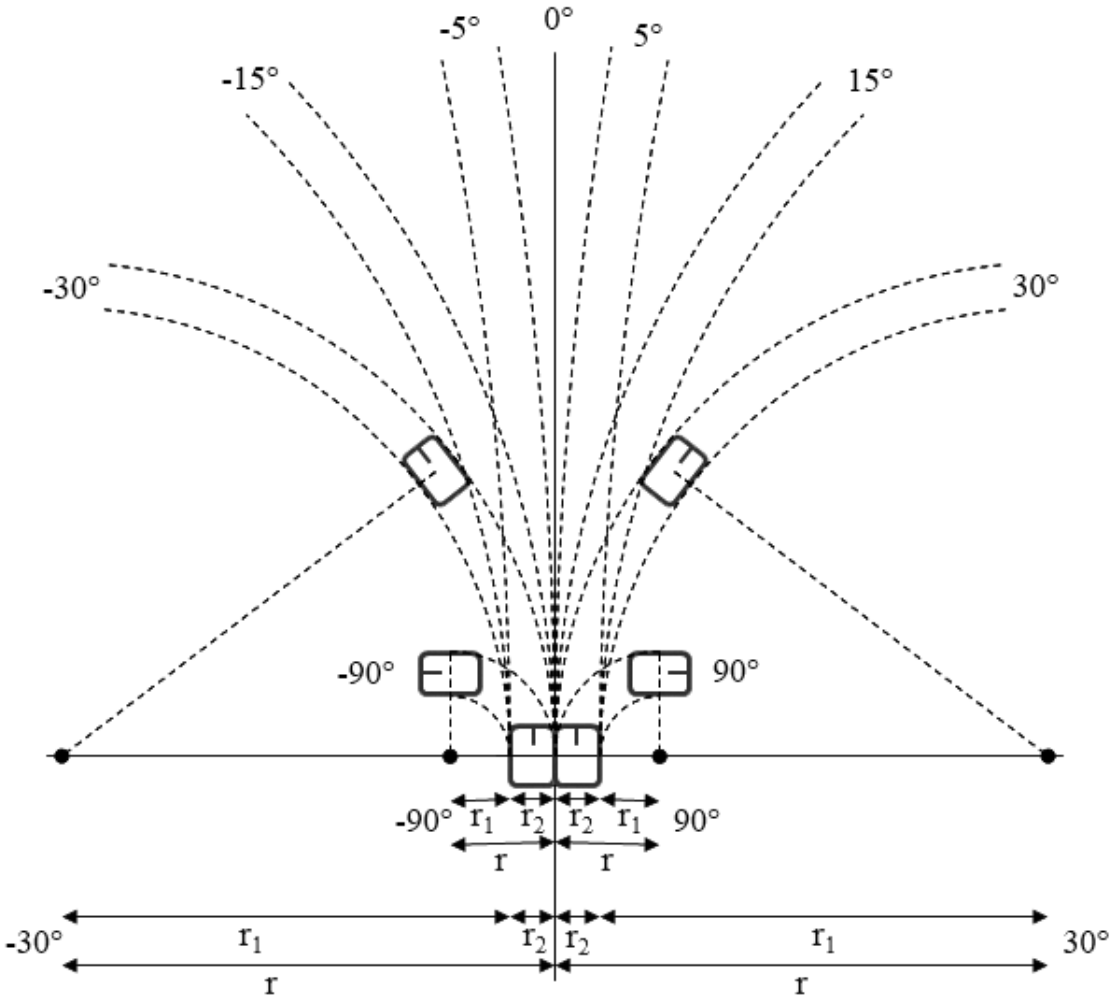


Figure 19: Concept of the motion controller with respect to steering angles.

The velocity of the left wheel (V_{left}) is set to an arbitrary value which determines the movement speed of the mobile robot. The same equations apply when steering to the left (steering angle is less than zero). But in this case the velocities are inverted

$$\frac{V_{left}}{V_{right}} = \frac{r_1}{r}$$

This approach is proved in experiments to positively affect the robot's movement resulting in smooth and continuous movements while navigating towards the target.

3.4 Summary

In this chapter, we proposed a general method for interpretation of different types of sensors, such as laser range scanner and ultrasonic sensors. We also proposed a new algorithm to perform the navigation task using our interpretation of sensory data. According to the analysis, it is expected that our approach can provide a generalized representation of the sensor's data for different types and numbers of sensors. On the other hand, it is also expected that this approach provides driving directions for mobile robots to safely plan a path to their destination while avoiding obstacles for different types and numbers of 2-dimensional sensors while trained with only one kind of 2-dimensional range sensor. In the next chapter, we will demonstrate the above analysis. The detailed implementations of the above proposed approach and the experimental results will also be presented in the following chapter.

Chapter 4

Implementation and Analysis of Results

In this chapter, we present the implementation details of our experiments in section 4.1, which includes simulation environment and its details, programming environment, and implementation of our proposed algorithm. The experimental results are given in section 4.2.

4.1 Implementation Details

4.1.1 Simulation Environment

USARSim (Unified System for Automation and Robot Simulation) [88,89] has been designed as a highly reliable simulation of urban search and rescue (USAR) robots and environments which is intended as a research tool for the study of human-robot interaction (HRI) and multirobot coordination. Since its initial release, it has been expanded to support many diverse environments including highway robots, the DARPA urban challenge, robotic soccer, submarines, humanoids, and helicopters. USARSim is designed as a simulation companion to the National Institute of Standards' (NIST) Reference Test Facility for Autonomous Mobile Robots for Urban Search and Rescue [90]. The NIST USAR Test Facility is a standardized disaster environment consisting of three scenarios: Yellow, Orange, and Red physical arenas of progressing difficulty. The USAR task focuses on robot behaviours, and physical interaction with standardized but disorderly unstructured environments. USARSim supports HRI by accurately rendering user interface elements (particularly camera video), accurately representing robot

automation and behaviour, and accurately representing the remote environment that links the operator's awareness with the robot's behaviours. [88,89]

Full effort of USARSim is dedicated to the robotics-specific tasks of modeling platforms, control systems, sensors, interface tools and environments. This is done by offloading the most difficult portions of simulation to a high volume commercial platform, which provides superior visual rendering and physical modeling. High reliability, at low cost, is made possible by constructing the simulation on top of a game engine (Unreal Tournament 2004). The robotics-specific tasks are in turn, accelerated by the advanced editing and development tools integrated with the game engine leading to a virtuous spiral in which a widening range of platforms can be modeled with greater reliability in less time. [88,89]

The current release of the simulation consists of: various environmental models (levels), models of commercial and experimental robots, and sensor models. For full documentation, please see [89].

The protocol used for communication by Unreal engine is proprietary. This makes it difficult for other applications to access Unreal Tournament. Hence, researchers have built a modification to Unreal Tournament (Gamebots) to connect Unreal engine with outside applications. It opens a TCP/IP connection in Unreal engine to exchange information with the outside. Some changes are applied to Gamebots to support USARSim control commands and messages which enables Gamebots to communicate with the controllers. [89]

4.1.1.1 Sensor Simulation

In robot control, sensors are very important. Three kinds of sensors are simulated in USARSim through checking the state of the object or some calculations in the Unreal engine: [89]

- "Proprioceptive sensors" (These include battery state and headlight state)
- "Position estimation sensors" (These include location, rotation, and velocity sensors.)
- "Perception sensors" (These include sonar, laser, pan-tilt-zoom (ptz) camera, touch sensor, and RFID tag reader.)

All of the sensors in USARSim are configurable and a sensor can be easily mounted on a robot by adding a line into the robot's configuration file. When mounting a sensor to a robot, the sensor's name, type, position (where it is mounted), and the direction it will face can be specified. For every type of sensor, certain properties can be configured. Examples of these properties include the maximum range of the sonar, the resolution of the laser and FOV (field of view) of the camera. [89]

Range Sensor

The range sensors are used to detect distances from objects and walls in the environment. There are two types of range sensor in USARSim; sonar and Infra Red (IR). Essentially, the range sensor is simulated by emitting a line from the position of the sensor along its direction in the Unreal world. The first point reached by the line is the hit point. Therefore, the distance between the hit point and the sensor is returned as the range value for that sensor. If the range is beyond the range which the sensor can detect, the maximum detection range will be returned. To simulate random noise, a random number

is added to the range value before the data is sent back. In addition, to simulate the real range sensor, a distortion curve is employed to interpolate the range data. [89]

For sonar sensors, it tries to emit several lines from the sensor within its beam cone instead of emitting just one line (see Figure 20). Therefore, the range value in sonar sensors is the shortest distance detected by the lines. For Infra Red (IR) sensor, only one line is used. However the line can cross through transparent materials (glass).

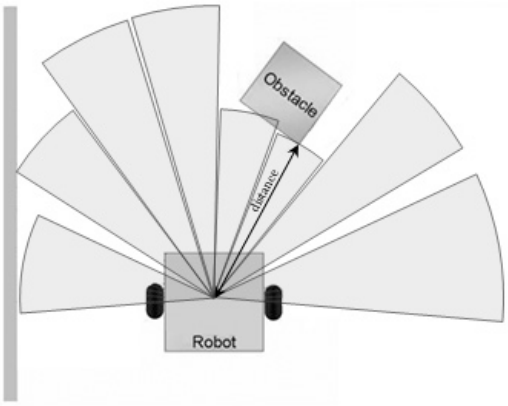


Figure 20: A mobile robot with 8 sonar sensors.

Range Scanner Sensor

In USARSim, the range scanner sensor is very similar to the range sensor as it is treated as a series of range sensors. The data for the range scanner sensor is obtained by rotating the range sensor from the start direction to the end direction in a fixed step where the step interval is calculated from the resolution. [89]

There are two types of range scanners; IRScanner and RangeScanner (also known as the laser range scanner). The IR scanner uses the IR sensor (which the detection line can cross transparent materials) to scan the environment. While the RangeScanner sensor uses the range sensor (only emits one detection line) to scan the environment (see Figure 21).

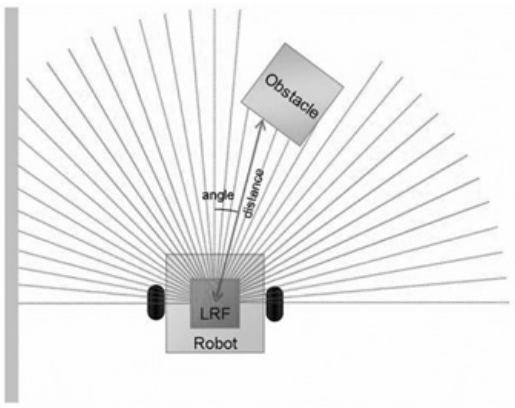


Figure 21: A mobile robot with a RangeScanner (Laser Range Finder). [91]

4.1.1.2 Robot Simulation

By employing the Karma rigid-body physics engine embedded in Unreal Tournament 2004, a robot model can be built to simulate a real world mechanical robot. The robot model comprises of "chassis, parts (tires, linkage, camera frame etc.), and other auxiliary items such as cameras, headlights, etc". All the chassis and parts are connected to each other by means of simulated joints that are driven by torques. Three kinds of joint control are supported in the robot model; the zero-order control, which makes the joint rotate by a specified angle; the first-order control, which lets the joint rotate under the specified rotational speed; finally, the second-order control which applies the specified torque on the joint.[89]

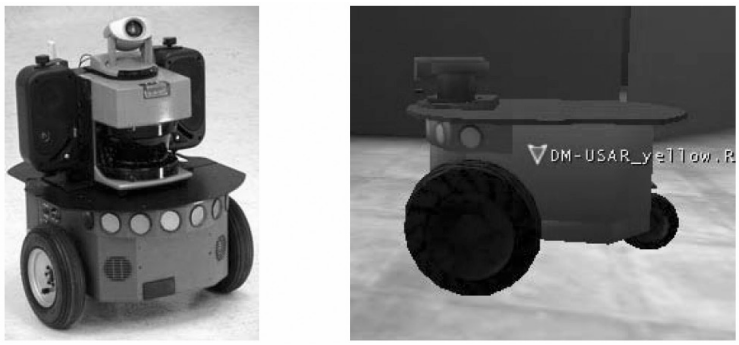
A list of USARSim robots is displayed below. Images of real world mechanical robots and the simulated versions of them are also shown.

- **P2AT** is a 4-wheel drive all-terrain pioneer robot from ActivMedia Robotics, LLC [92].

(a) (b)

Figure 22: a) Mechanical P2AT. b) Simulated P2AT. [89]

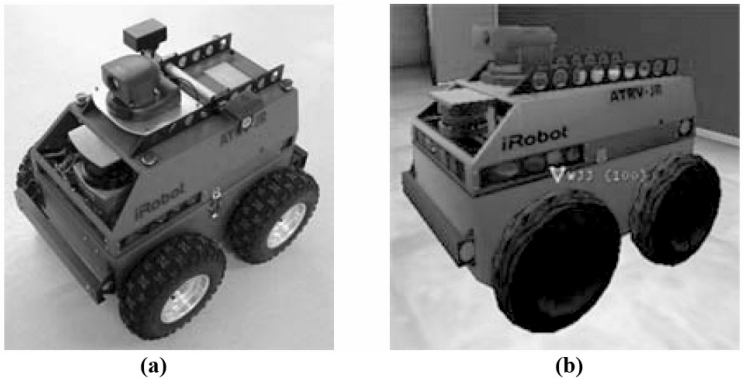
- **P2DX** is the 2-wheel drive pioneer robot from ActivMedia Robotics, LLC [92].



(a) (b)

Figure 23: a) Mechanical P2DX. b) Simulated P2DX. [89]

- **ATRV-Jr** is a 4-wheel drive outdoor all terrain robot vehicle developed by iRobot [93].



(a) (b)

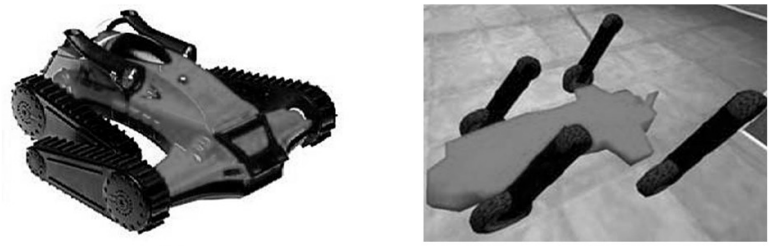
Figure 24: a) Mechanical ATRVJr. b) Simulated ATRVJr. [89]

- **Zerg** is a 4WD (4-wheeled) robot, which also has been developed and deployed by the team “Rescue Robots Freiburg” [94] during RobotCup05. The simulation model was also developed at University of Freiburg, and has been further improved and merged into USARSim by the University of Pittsburgh. [89]

(a) **(b)**

Figure 25: a) Mechanical Zerg. b) Simulated Zerg. [89]

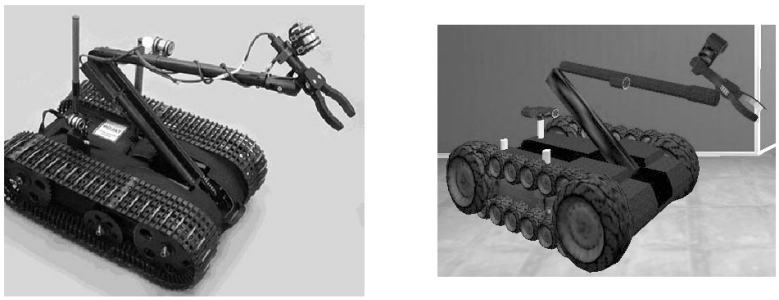
- **Tarantula** is a toy-based robot which was first turned into a robot platform named "Lurker" by the team "Rescue Robots Freiburg" [94]. They used the modified version in the Rescue Robot League during the RoboCup 2005 competition. The Tarantula model, which is now part of the USARSim package, was originally developed at the University of Freiburg and has been further improved and merged into USARSim by the University of Pittsburgh [89].



(a) **(b)**

Figure 26: a) Mechanical Trantula. b) Simulated Trantula. [89]

- **Talon** is a lightweight tracked vehicle built by Foster-Miller for missions ranging from reconnaissance and weapons delivery to rescue.



(a) (b)

Figure 27: a) Mechanical Talon. b) Simulated Talon. [89]

For the convenience of our project we have only used some of these robots.

4.1.2 Programming Environment

A schema of our programming structure can be seen in Figure 28. Typically, a controller is the user side application that is used for research, such as robotics study, team cooperation study, human robot interaction study etc. Usually, the controller works in a way that it first connects with the Unreal server. Then it sends commands to USARSim to spawn a robot. After the robot is created on the simulator, the controller listens to the sensor data (sent every third of a second from the server) and sends commands to control the robot.

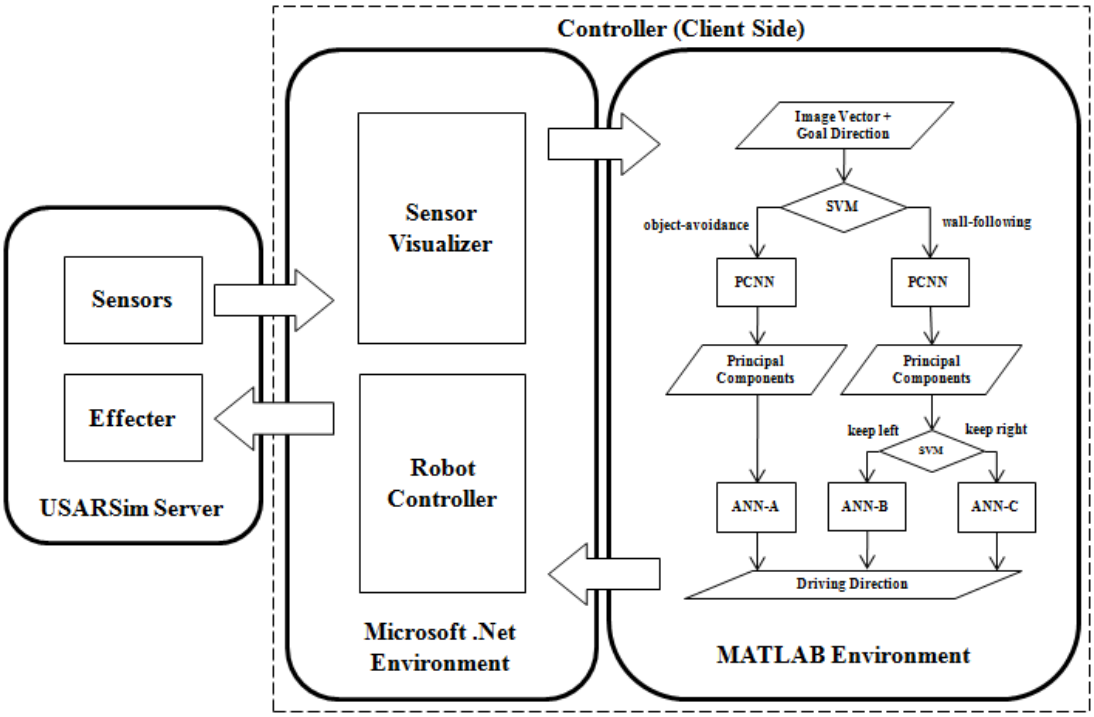


Figure 28: Programming Structure Schema

To reduce the complexity of programming, we have used MATLAB and its powerful toolboxes to implement the core decision making part of our algorithm. In order to connect to the USARSim server and to send and receive commands, Microsoft .Net (C#) is used. Therefore, we use MATLAB as an automation server from C# using the engine interface via com automation. This allows us to simultaneously debug our application from both the C# side and the MATLAB side, using debuggers on each side.

As depicted in Figure 28, after a sensor reading is retrieved from the USARSim server, the sensory data is visualized into a binary image in the "*Sensor Visualization*" section of the .Net Environment. This image is then be converted into a vector of zeros and ones and sent to MATLAB along with the goal direction. In MATLAB, firstly we decide which action to take based on the data introduced from C#. If the object-avoidance action is selected, then the features of the binary image are extracted and introduced to a

pre-trained neural network (ANN-A). We have trained the networks to provide the steering direction to the controller to safely navigate the mobile robot in the environment. On the other hand, if the first SVM decides that a wall-following action is required, then after extracting the features of the image, another SVM decides on the direction of the wall-following. A brief overview of the algorithm implemented in MATLAB is given in section 3.3 and is explained in details in the following sections.

4.1.3 Sensor Data Visualization

As mentioned in Chapter 3, the visualization part is the most important part of our algorithm. By visualizing the sensors' data retrieved from the server, the whole structure will have the ability to generalize for different types of sensors. In other words, if the networks are trained to perform the navigation task using one type of sensor, e.g. RangeScanner, then they will also be functional for other types of sensors, e.g. Sonar, without requiring any further training.

For this purpose the data from different sensors are visualized in 50×50 pixels bitmaps covering a 25 meter area (5×5 meters). Figure 17 and Figure 29 illustrate examples of how the sensor readings are visualized into images. Therefore, each pixel of the image represents 10 centimetres in the simulation environment (1:10 scale factor). Typically, for local navigation and obstacle avoidance, 5 meters of distance is sufficient. Unless objects are closer than 5 meters there is no need to change the bearing of the mobile robot to avoid them. Moreover, for detecting obstacles on either sides of the mobile robot, 2.5 meters will be satisfactory as the robot will only need the side sensors to safely rotate in both directions.

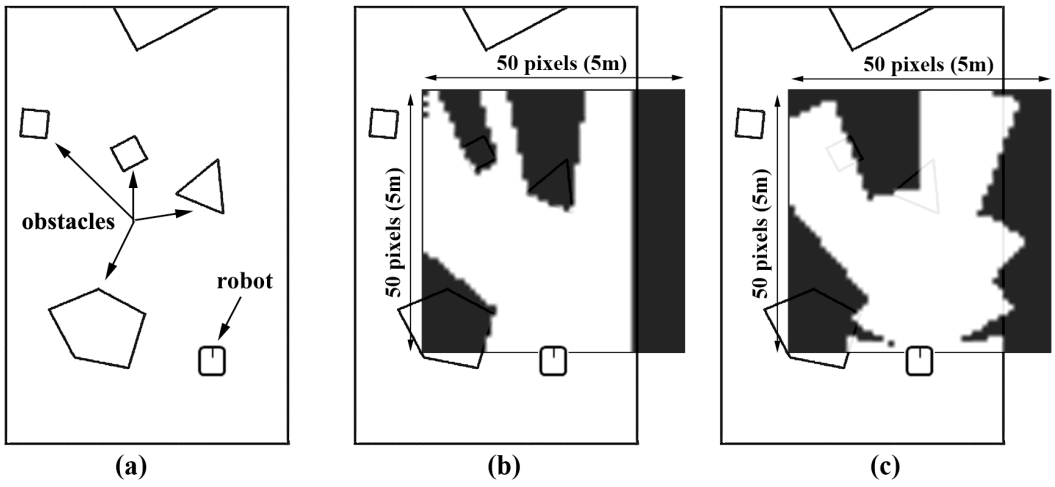


Figure 29: a) Environment Sample b)RangeScanner Visualization c) Sonar Sensor Visualization

For the RangeScanner, the server returns 181 values at every sensor reading, representing the distance of obstacles in 180 degrees in front of the robot (0° −180°). Therefore, this sensor can be visualized by drawing white lines, on a black image, from the origin of the sensor with a length equal to the distances in their corresponding directions (see Figure 29(b) and Figure 30(b)). Note that the data from the sensors needs to be scaled to match the 1:10 scale factor mentioned above. The maximum range of obstacles detected with the laser range finder is 20 meters. Thus, any object that is detected outside of the 5 meter range will be ignored as it will not affect the obstacle avoidance algorithm.

Visualization of the sonar sensor is somewhat different than the laser range finder. Each value from the sonar sensors represents the distance from the closest object in a 20° cone in the orientation of that sensor. Therefore, in the direction of each sonar sensor we have to draw a 20° pie, where the length of each pie will be a scaled value of the distance from the detected object. Figure 29(c) and Figure 30(c) depict visualizations of the environments in Figure 29(a) and Figure 30(a) respectively, using 8 sonar sensors

mounted in front of the robot. The sonar sensors we use can detect obstacles up to 5 meters. Due to the noise in their values, the accuracy of the detected objects is not as high as laser range scanners. Therefore, there are some errors, specially, when objects are farther than 3 meters. For example, in Figure 30(c), the obstacle on almost 45° to the left side of the robot has not been detected even though there are two sonar sensors pointing in that direction. A similar distribution of the sonar sensors which has been used to generate this image can be seen in Figure 20 on page 56.

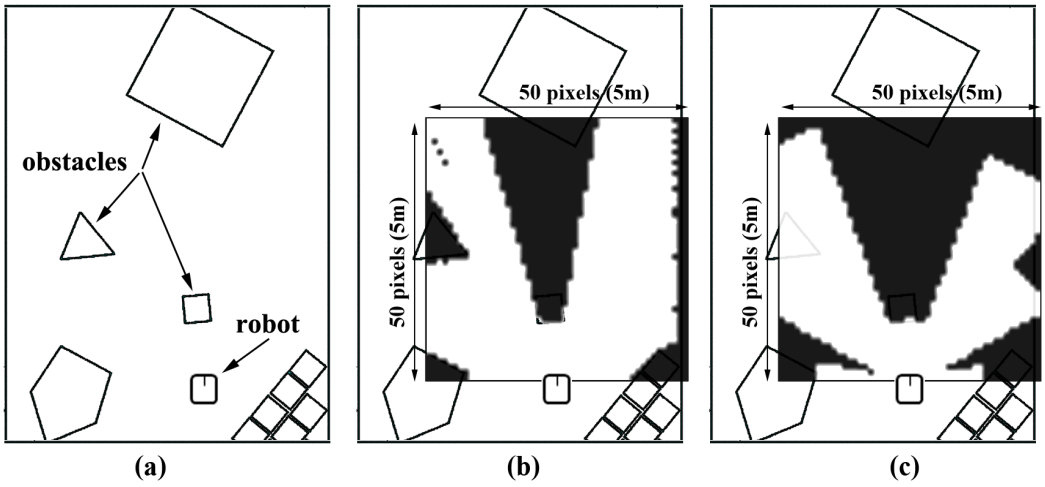


Figure 30: a) Environment Sample b)RangeScanner Visualization c) Sonar Sensor Visualization

The binary image is a 2-dimensional representation of safe (white pixels) and unsafe (black pixels) areas in the environment. In other words, Black represents obstacles, walls (in general any non-traversable area) and unknown areas, and, alternatively, traversable areas are represented by white. Before introducing these images to the next level, we need to convert them to vectors of values so they can be processed. To do this, in a 50×50 matrix we assign the values of 0 and 1 to represent black and white pixels respectively. Eventually, we can collapse the matrix to collect the row contents into a vector of 2500 binary values. This vector, along with the goal direction,

which has also been retrieved from the server, is passed to the MATLAB environment for further processing.

4.1.4 Algorithm Implementation

To benefit from the toolboxes implemented in MATLAB and simplicity of programming, a major part of the algorithm has been implemented using MATLAB 2010b and its neural network and machine learning toolboxes.

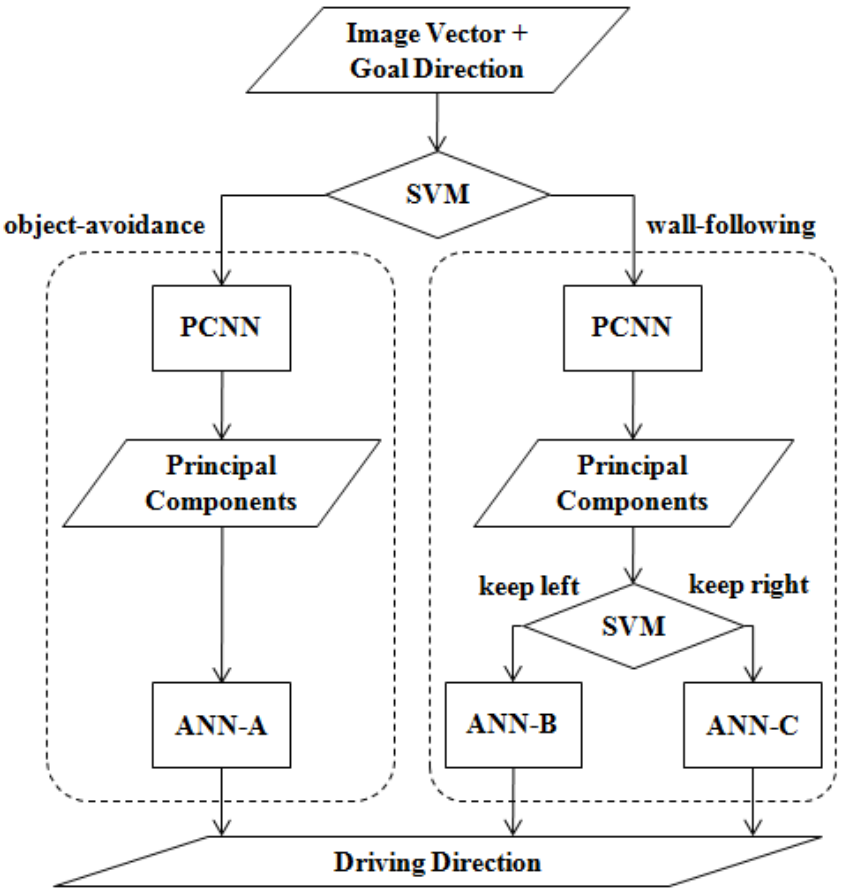


Figure 31: Flowchart of algorithm implemented in MATLAB.

A support vector machine (SVM) is trained by introducing it with 150 training samples gathered from the simulation environment using the laser range scanner. A Radial Basis Function (RBF) with sigma value equal to 10 is used for the kernel of this

SVM. It is noteworthy to mention the configurations and settings of separate components are found empirically. This SVM will classify any new sample to either object-avoidance or wall-following at each time step. The algorithm implemented in this section can be pictured as two modules; object-avoidance and wall-following (see Figure 31).

Object-Avoidance

The object avoidance module has the responsibility of producing a safe driving direction for the mobile robot to navigate towards the target while avoiding any obstacles in its path. For this purpose a multi-layer neural network is trained using Levenberg-Marquardt backpropagation algorithm in MATLAB (trainlm) to produce the steering direction. trainlm is a network training function that updates weight and bias values according to Levenberg-Marquardt optimization [95]. Validation vectors are used to stop training early if the network performance on the validation vectors fails to improve or remains the same for maximum fail epochs (number of iterations) in a row. Test vectors are used as a further check that the network is generalizing well, but do not have any effect on training. The network's performance is measured according to the mean of squared errors (mse function in MATLAB). The learning rate (η) and momentum (α) were found empirically to be 0.3 and 0.06 respectively.

However, when the number of input units of a neural network is too large, the training time will significantly increase. Therefore, in our case, where we have 2501 input units (2500 units for the image vector and 1 unit for the target direction), which is considered to be a very high number, we need to reduce the dimension of the data. For this, we have used a Principal Component Neural Network (PCNN), trained using the Generalized Hebbian learning Algorithm (GHA), to reduce the dimension from 2500 to

100 features (for more details about GHA see section 2.2.2). The PCNN has been trained using 100 training patterns from the laser range scanner of the P2AT mobile robot. Other algorithms such as Adaptive Principal component EXtractor (APEX) have also been tested but haven't performed as well as GHA in general. As mentioned in the previous chapter, there is a trade off between performance and speed when selecting the number of extracted principal components. Through experiments, we came to believe that 100 features are sufficient for our proposed method, which maintains a reasonable training speed and at the same time a high accuracy. Therefore, these 100 features (principal components) with the goal direction will form the inputs of our multi-layer neural network ANN-A (in Figure 31). A detailed structure of ANN-A is illustrated in Figure 32. This network consists of one input layer with 101 units, two hidden layers and an output layer. The output unit uses a linear transfer function (purelin function in MATLAB) where it just transfers the sum of its input values to the output (Figure 33).

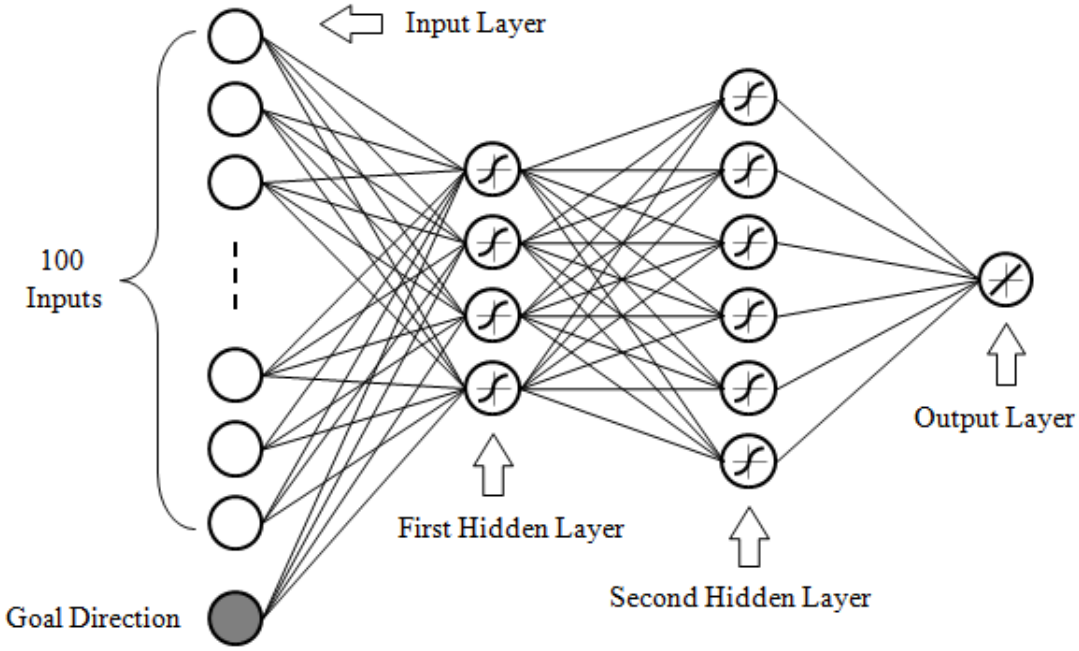


Figure 32: 3-layer Artificial Neural Network (ANN-A)

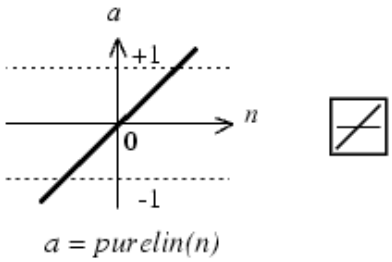


Figure 33: Linear Transfer Function

The first and second hidden layers are composed of 4 and 6 neurons respectively with a hyperbolic tangent sigmoid transfer function (Figure 34). In [96] the $tansig$ is defined as

$$a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$$

However, a look on the MathWorks homepage with the keyword *tansig* will show that $tansig(n)$ calculates its output according to:

$$a = \frac{2}{1 + e^{-2 \times n}} - 1$$

This is mathematically equivalent to $tanh(n)$. It differs in that it runs faster than the MATLAB implementation of $tanh$, but the results can have very small numerical differences. This function is a good trade off for neural networks, where speed is important and the exact shape of the transfer function is not.

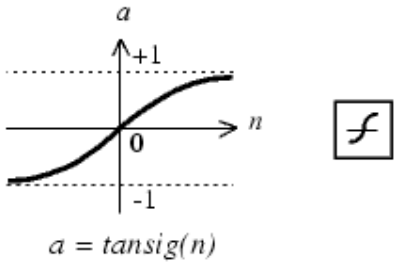


Figure 34: Tan-Sigmoid Transfer Function

Wall-Following

As mentioned before, in order to move around large objects or navigate from one room to another, a wall following method is required. In some cases, such as when the robot encounters U-shaped objects or should navigate to another room to reach its target, the mobile robot needs to drive in the opposite direction of the target. This is possible if the direction of the goal is not considered in the wall following method. Therefore, in this module the target's direction does not affect the decision made by the neural networks. Upon reaching a wall or a wide obstacle a left or right direction is selected for navigating around the wall. Until the object avoidance action is not triggered again, the direction of wall-following will be maintained. For example, if the robot is following the wall and keeping right (the walls will always be on the right side of the robot), then it will never change the wall-following direction unless the object-avoidance action is required. In other words, the direction to follow the wall is decided only at the time of changing from object-avoidance to wall-following.

In the wall following module same PCNN and neural network structures as in the object avoidance module have been used. Except that in this case we are not concerned with the goal direction. As a result, we extract the same number of features from the image vectors, as we did in object avoidance module, using PCNN trained by GHA. The PCNN in this section is trained using 300 training patterns, 150 patterns for each direction of wall-following, from the laser range scanner of a P2AT mobile robot.

An SVM is also trained with 200 training patterns to classify the new samples into *keep-right* or *keep-left* classes. When a robot is moving alongside a wall, while

maintaining a certain distance from the wall and keeping the wall on its left or right, we declare that it's *keeping-left* or *keeping-right* respectively.

Two neural networks, ANN-B and ANN-C, provide driving directions to *keep-left* and *keep-right* respectively, regardless of the goal direction. This ensures that, if required, the robot drives away from the target in order to go around large objects or move from one room to another without getting trapped in a deadlock (see Figure 35). Both these networks are trained by providing them with 1500 training patterns. Their network structure is similar to the structure of ANN-A with a difference in the number of input neurons. In the two wall-following neural networks (ANN-B and ANN-C) only the extracted features from the PCNN algorithm form the input layer. As mentioned before, the direction of the target has no affect on the final output of the network. Therefore, we do not consider it in our network structure. The transfer functions in different layers and the performance measurement of the whole network are the same as ANN-A. However, through empirical observation we found the learning rate (η) equal to 0.001 and momentum equal to 0.05.

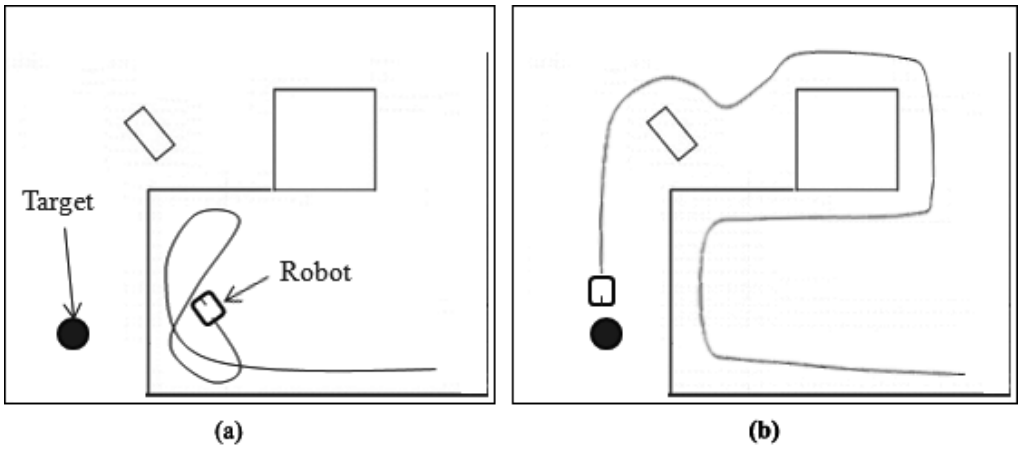


Figure 35: Mobile robot's path a) without a wall-following algorithm b) with a wall-following algorithm, when encountering a U-shaped obstacle.

After the driving direction has been calculated by the neural networks, the output value is passed back to the .Net environment. Using C#, a robot controller has been implemented to provide necessary commands to the server to drive the robot based on the driving direction.

4.1.5 Robot Controller

The robot controller calculates the velocity of the left wheel (V_{left}) and right wheel (V_{right}) based on the driving directions received from the MATLAB part of the algorithm, and then sends commands to the USARSim server to drive the robot. The output of the network has to be converted in a meaningful way into velocities for the left and right wheels. Figure 36 shows two situations where the robot has encountered an obstacle in front of it while moving towards the target. From the neural networks, driving directions of 90 degrees and 45 degrees have been calculated for Figure 36(a) and Figure 36(b) respectively.

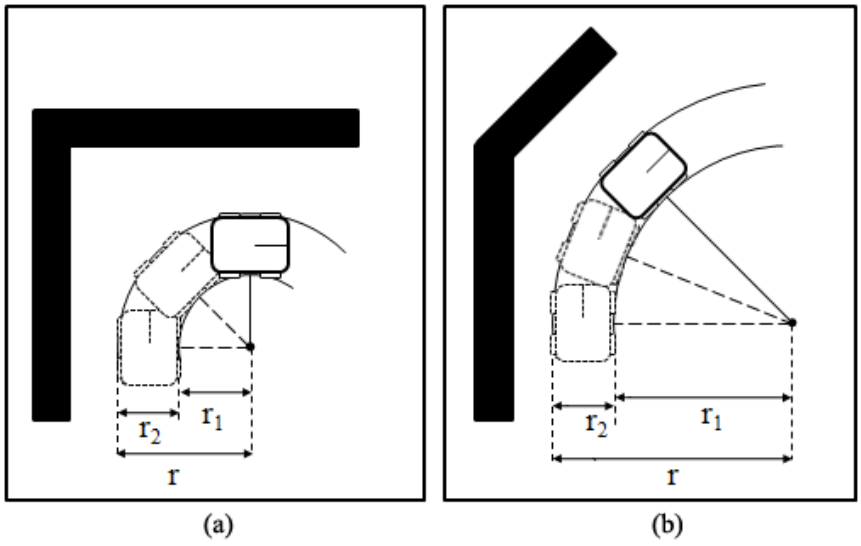


Figure 36: Two obstacle-avoidance examples in mobile robot motion control.

a) 90° rotation to the right b) 45° rotation to the right.

If the output angle is greater than zero which means the robot has to rotate to the right, the velocities are calculated as

$$\frac{V_{right}}{V_{left}} = \frac{r_1}{r}$$

where V_{right} is the velocity of the right wheels, V_{left} is the velocity of the left wheels, r is the curvature of the outside (in this case left side) wheels of the robot and $r_1 = r - r_2$ is the inside wheels curvature of the robot. The radius of the curvature is determined from

$$r = \frac{r_2 \times x}{|\theta|}$$

where x is an arbitrary value, r_2 is the width of the robot, θ is the steering angle (driving direction). The width of the robot is known, therefore x is used to determine the total curvature desired for rotating the robot. In our case we have $x = 90cm$. Hence, if the steering angle (θ) is equal to 90°, we will have $r = r_2,$ therefore, $r_1 = 0$. This means that the velocity of the right wheel is zero ($V_{right} = 0$). The velocity of the left wheel (V_{left}) is set to an arbitrary number which can be changed during the simulation. This value determines the movement speed of the mobile robot.

The same equations apply when steering to the left (steering angle is less than zero). But in this case the velocities have to be inverted

$$\frac{V_{left}}{V_{right}} = \frac{r_1}{r}$$

Note that the left hand side of the equation has been inverted. The equation to calculate r and r_1 is the same as rotating to the right.

By looking at the image and the given equations if $r_1 = \frac{1}{2}r_2$, then r will be smaller than r_2 which in this case the velocity of the right wheel will obtain a negative

value. Therefore, the robot will rotate in its current position without moving forward. This is mainly used when we want the robot to completely turn around or to avoid collision with very close obstacles.

Figure 19 on page 51 shows how the mobile robot steers, given different steering angles (driving directions). As the steering angle gets closer to zero, the curvature becomes straighter. This will positively affect the robot's movement to not to make sudden changes when it is approaching the target. On the other hand, when the steering angle gets closer to 180 degrees the circle becomes smaller and smaller, therefore affecting the robot to rotate on a very smaller curve.

This approach will positively affect the robot's movement resulting in smooth and continuous movements while navigating towards the target.

4.2 Experimental Results

In this section we present experiments conducted with the simulated robots. The central question driving our experiments is: to what extent can mobile robots successfully navigate to their targets without any collisions, when the algorithm is trained using a different type of sensor from another robot or the same robot.

In the following experiments we use USARSim server which is installed on top of the Unreal Tournament game engine. Under the help of this tool, we can test our proposed method in a variety of scenarios. We can also track the path traversed by a mobile robot within the environment.

4.2.1 Training

Our experiments were conducted with three different simulated robots; P2AT, Zerg, and Talon (see section 4.1.1.2 for details about these robots). Furthermore, three

different configurations of two types of sensors (laser range scanner and ultrasound sensor) have been used to perform navigation in different environments as shown in Figure 37. Because of the physical availability of actual robots and various sensors we have limited our research to the simulation environment. However, as previous experiments have shown, neural networks trained in simulation environments can also be applied to real world robots to perform navigation tasks [19-21].

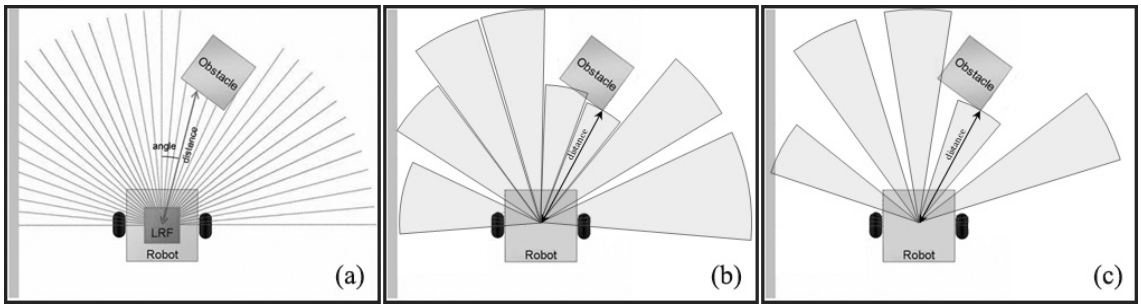


Figure 37: Different sensor distributions. a) laser range scanner.

b) 8 sonar sensors. c) 5 sonar sensors.

Figure 38 shows an example of the view area of a mobile robot which is used for the navigation and obstacle avoidance purpose.

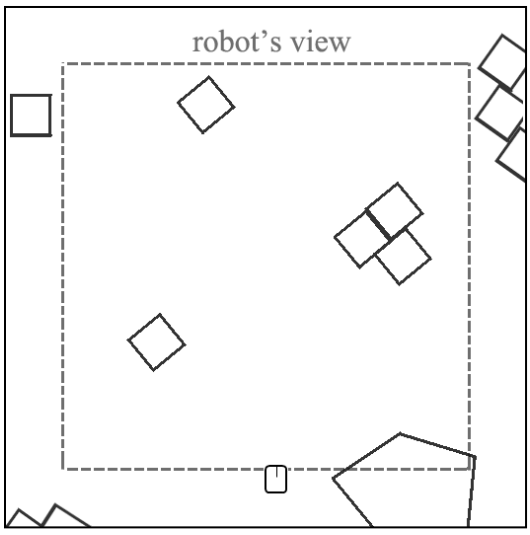
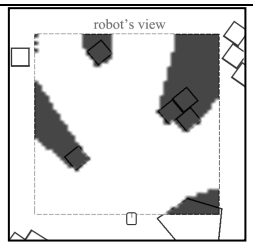
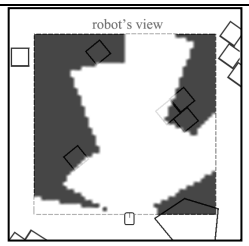
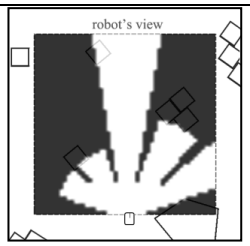
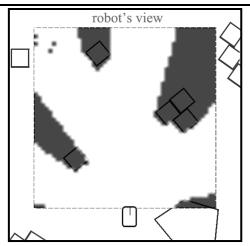
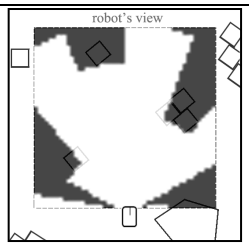
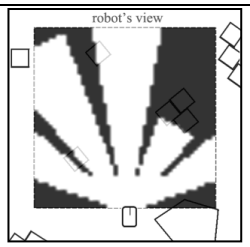
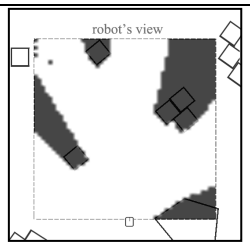
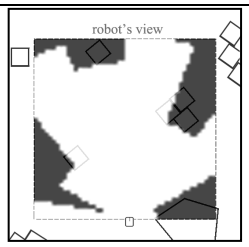
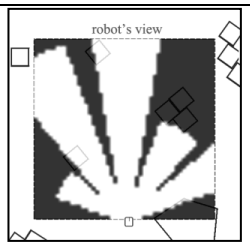


Figure 38: An example of robots view in an environment.

The generated image from the sensor readings which is covered by the view port will be used as the input to our algorithm. To demonstrate the differences between the three robots and their sensor settings, Table 2 depicts the sensor readings of the robots with three different configurations of sensors from the position shown in Figure 38.

Table 2: Examples of sensor readings of different sensors from three different mobile robots.

	Sensors		
Robot (dimensions)	Laser	8 Sonars	5 Sonars
P2AT Length=0.5239m Width=0.4968m Height=0.2914m			
Talon Length=0.9117m Width=0.5903m Height=0.3654m			
Zerg Length=0.3112m Width=0.4154m Height=0.1211m			

For training purposes, only the laser sensor readings and the P2AT mobile robot have been used. To train the object-avoidance network (ANN-A), 3000 training patterns are gathered from the environment shown in Figure 39 (Figure 40 shows the 3D view of the same environment) by manually navigating the robot from the starting positions to the target and saving the sensor readings, the target's direction and the robot's current steering angle at a certain time step (1 second).

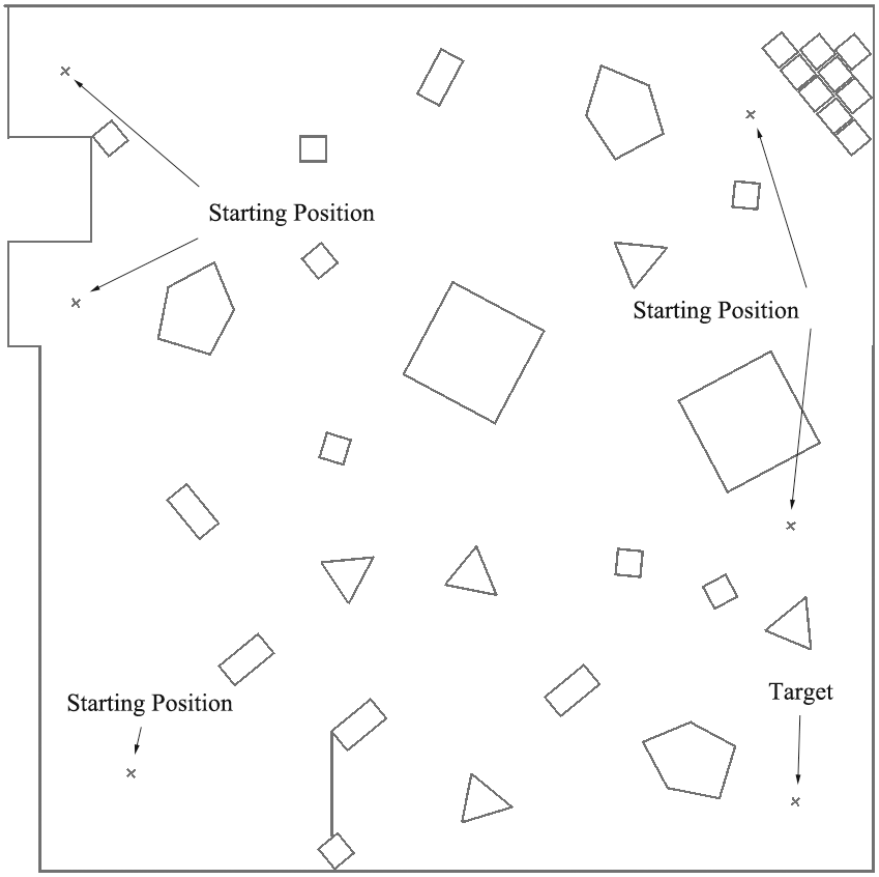


Figure 39: A 2D top view of our training environment for object-avoidance.

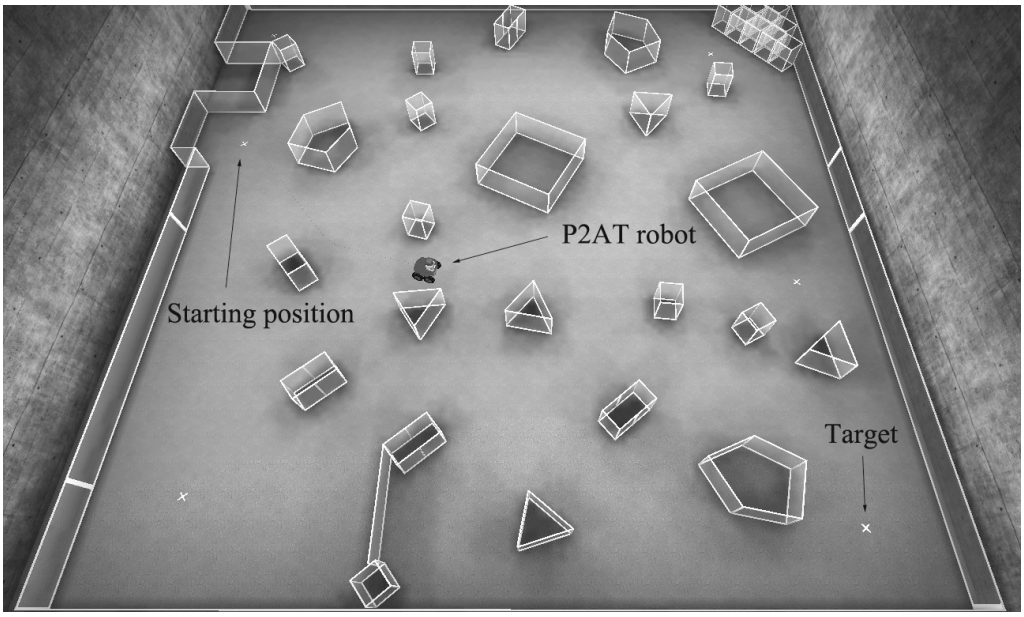


Figure 40: A 3D view of our training environment for object avoidance.

When training a Neural Network, in order to avoid overfitting, generalization is an important feature to consider. Overfitting may occur when the error on the training set is reduced to a very small value. Hence, the network will perform very well for that specific training set because it has memorized the training examples. However it cannot learn to adapt to new situations. In other words it is not generalized. [17,22]

There are several methods in which the generalization of the network can be improved without sacrificing accuracy [17,22,96]. A commonly used method is known as Early Stopping. This method employs validation to stop the training process when the network starts to overfit the data. By passing a validation set, the training function will test this new data set at certain points in the training phase to understand how the network is responding for other inputs. The training will stop when the error of the validation set starts to increase which generally indicates overfitting.

Thus, for testing and validation of our network, we used 3000 patterns collected using only the sonar sensors to prevent the networks from overfitting.

To train the wall-following network, ANN-B, we used a different environment, as shown in Figure 41. A 3D view of the simulated environment in Unreal Tournament can also be seen in Figure 42. For this purpose the robot was driven manually in this environment only following the wall and keeping-left. 1500 laser scanner readings from P2AT robot were collected. These data were gathered by navigating to the target from different starting positions moving alongside the wall and keeping-right at all times. Moreover, 1500 patterns from the sonar sensors were gathered for testing and validation purposes. The same environment, but mirrored, was used for collecting same amount of training data for ANN-C (keep-right).

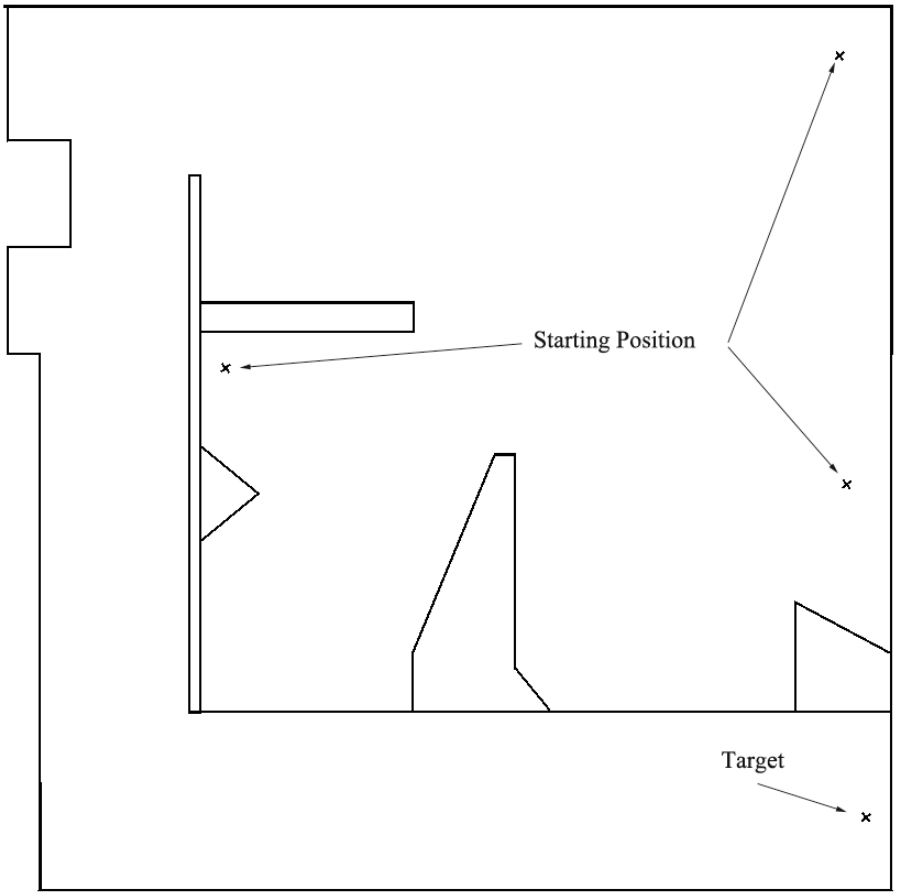


Figure 41: A 2D top view of our training environment for wall-following.

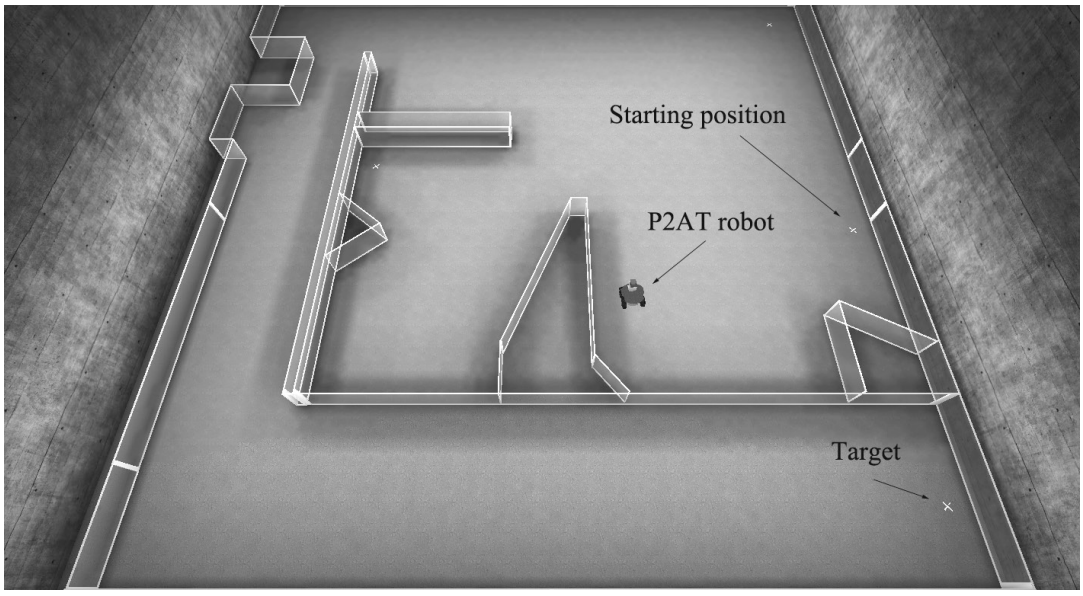


Figure 42: A 3D view of our training environment for wall-following

The *action-SVM,* which decides on which action to take at each time step, has been trained by providing it with 150 patterns; 75 patterns for object-avoidance and 75 patterns for wall-following. To train the *direction-SVM*, we have introduced it with 200 patterns (100 samples for each class) to classify keep-left and keep-right. A few examples of these training patterns are depicted in Table 3. Table 3(a) shows some training samples used for training the action-SVM. Table 3(b) also shows a few examples of patterns used for training the direction-SVM. Note that the direction of the target is not considered in the classification of the patterns in the direction-SVM. Due to the fact that keep-left and keep-right directions are opposite of each other, therefore mirrored patterns of keep-left patterns can be used for keep-right training patterns.

Table 3: Examples of training patterns for training a) the action-SVM, and b) the direction-SVM.

(a)

Action	Sample	Target's Direction
Object Avoidance		27°
		45°
		10°
		-55°
Wall Following		-1°
		1°
		4°
		-5°

(b)

Direction	Sample
Keep Left	
	
	
	
Keep Right	
	
	
	

Table 4 compares performances and training times of three feature extraction methods, GHA, APEX and NMF, for training object avoidance and wall following neural networks, ANN-A, ANN-B and ANN-C. The values in the table illustrate the mean of 10 performance tests carried out for each network and each feature extraction method. The performance columns show the mean squared error for training the neural networks. So, lower values in the performance columns show better performances of the trained neural networks. Also the regression columns describe the relationship between predictor and response variables. The bold values in the table highlight the best performances and training times for the networks using different feature extraction methods.

As shown in the table, the training speed of NMF is noticeably faster than the other two methods. The extracted features, using this method, even have some affects on reducing the training time of the neural networks. However, as there is always a trade-off between speed and accuracy, compared to the other methods, the NMF method has the lowest training performance. Typically, we are looking for a feature extraction method that extract features which can result in higher training performances with a view to better generalization. In our case the training time does not affect the overall performance of our proposed method. Therefore, based on our experiments shown in Table 4, we have chosen GHA for testing our algorithm in different environments. As it can be seen from the table, the GHA method has the highest performance in training the networks and generalizes very well to the validation set. The APEX method is not considered due to the fact that its performance is not as good as GHA and it also has a higher training time.

Table 4: Comparison of three feature extraction methods for training ANN-A, ANN-B and ANN-C using the average value of 10 performance trainings.

ANN	F. E. Method	Training Time (seconds)		Performance (mean squared error)				Regression		
		Feature Extraction	NN Training	Train	Validate	Test	Overall	Train	Test	Validate
A	GHA	114.19	14.82	**334.25**	**483.81**	**491.48**	**410.80**	**0.8265**	**0.741**	**0.7333**
	APEX	164.24	12.31	342.29	499.04	513.47	424.12	0.8218	0.734	0.7234
	NNMF	**20.254**	**10.18**	710.19	939.22	928.42	821.79	0.5685	0.351	0.3575
B	GHA	59.511	8.076	**116.83**	**301.55**	313.30	**210.57**	**0.931**	**0.827**	**0.8193**
	APEX	80.160	**6.956**	126.73	303.62	**310.29**	215.37	0.9244	0.820	0.817
	NNMF	**20.017**	7.381	159.79	483.70	512.20	326.12	0.906	0.679	0.6644
C	GHA	61.717	8.186	**128.34**	**310.12**	**322.63**	**220.82**	**0.9266**	**0.818**	**0.8123**
	APEX	89.854	7.762	132.26	334.12	354.03	236.44	0.9209	0.802	0.7907
	NNMF	**13.799**	**6.856**	143.23	483.90	501.75	315.18	0.9152	0.716	0.7048

Figure 43 and Figure 44 depict the regression and error plots for training ANN-A from our algorithm. As it can be seen from Figure 44, the mean squared error of the validation and test samples start to increase after epoch 13. Therefore to prevent the network from over fitting the training samples and to be able to generalize to new samples, training is halted at epoch 13. The regression plots in Figure 43 show the results of the networks outputs for the training patterns compared to the actual targets at step 13.

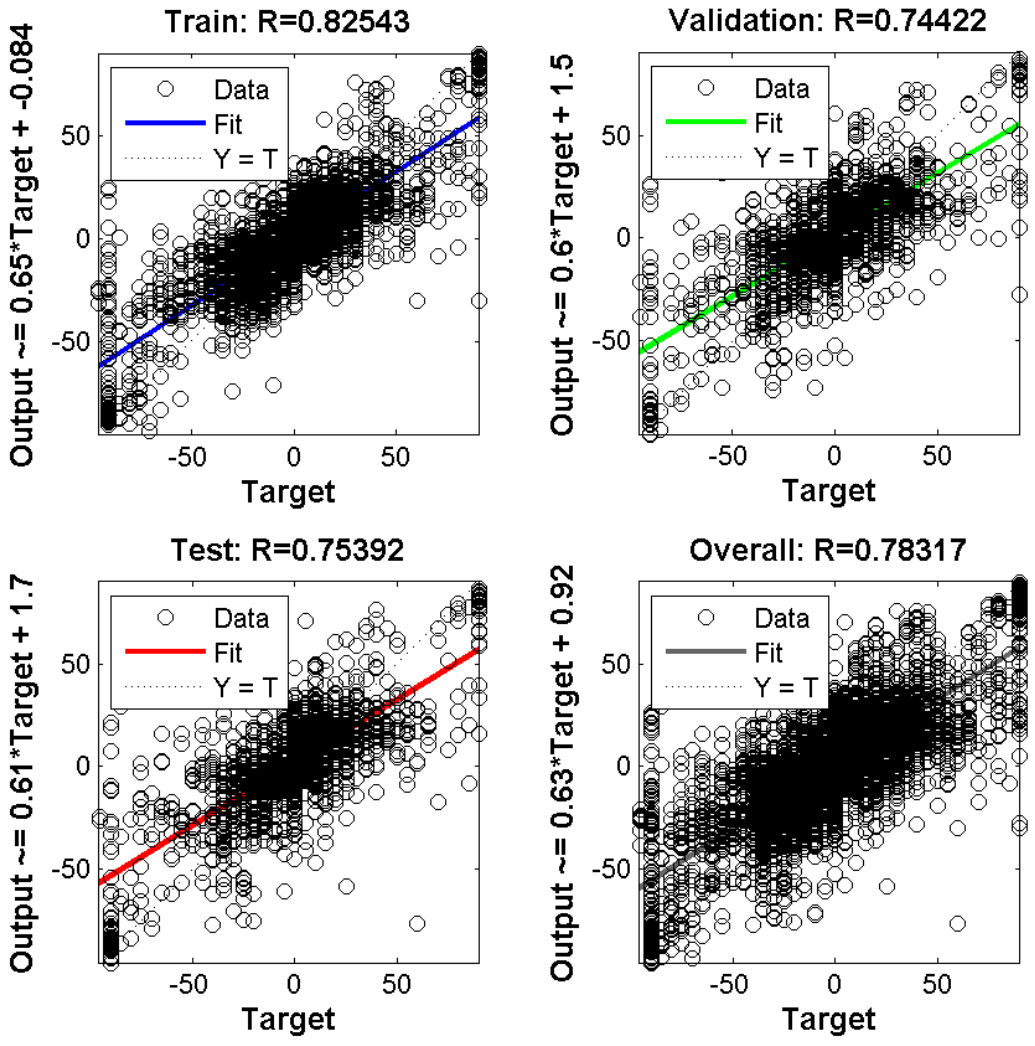


Figure 43: Regression plots for training ANN-A with 3000 training samples and 3000 samples for testing and validation.

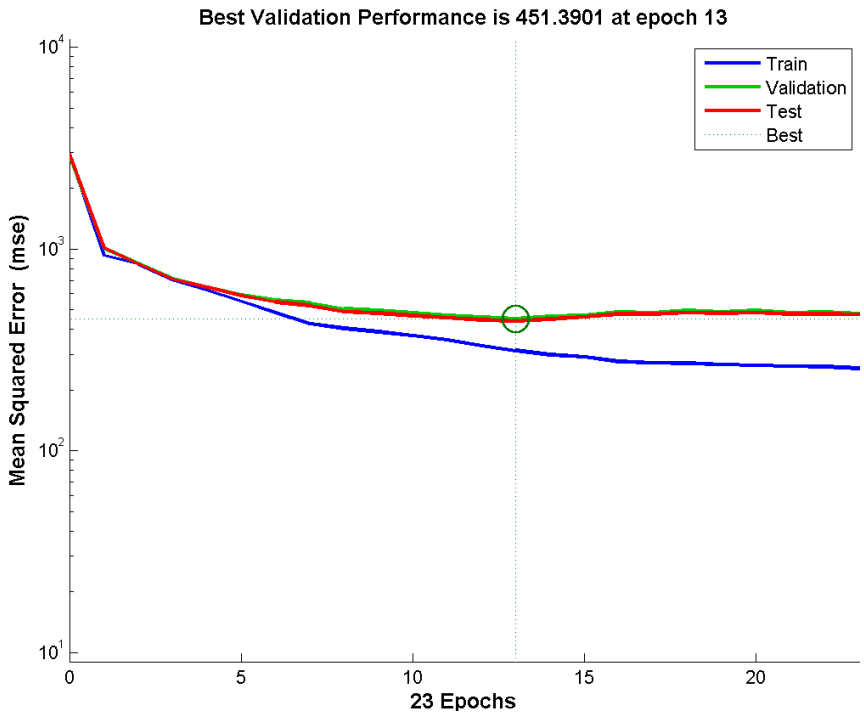


Figure 44: Performance plot for training ANN-A

Regression and performance plots of training results for keep-left and keep-right wall following networks are shown in Figure 45 through Figure 48. Based on the performance plots we can see that the networks have obtained the best validation performance for training ANN-B and ANN-C at epochs 16 and 12 respectively. The plots show very good results for the laser scanner patterns (training samples). Although the accuracy in comparison to the validation and testing patterns are a bit lower than the training samples, the final result is satisfying. As we will see further in this chapter, navigation results of different types of sensors are very satisfying. This shows the fact that our algorithm has the ability to generalize for different types of sensors.

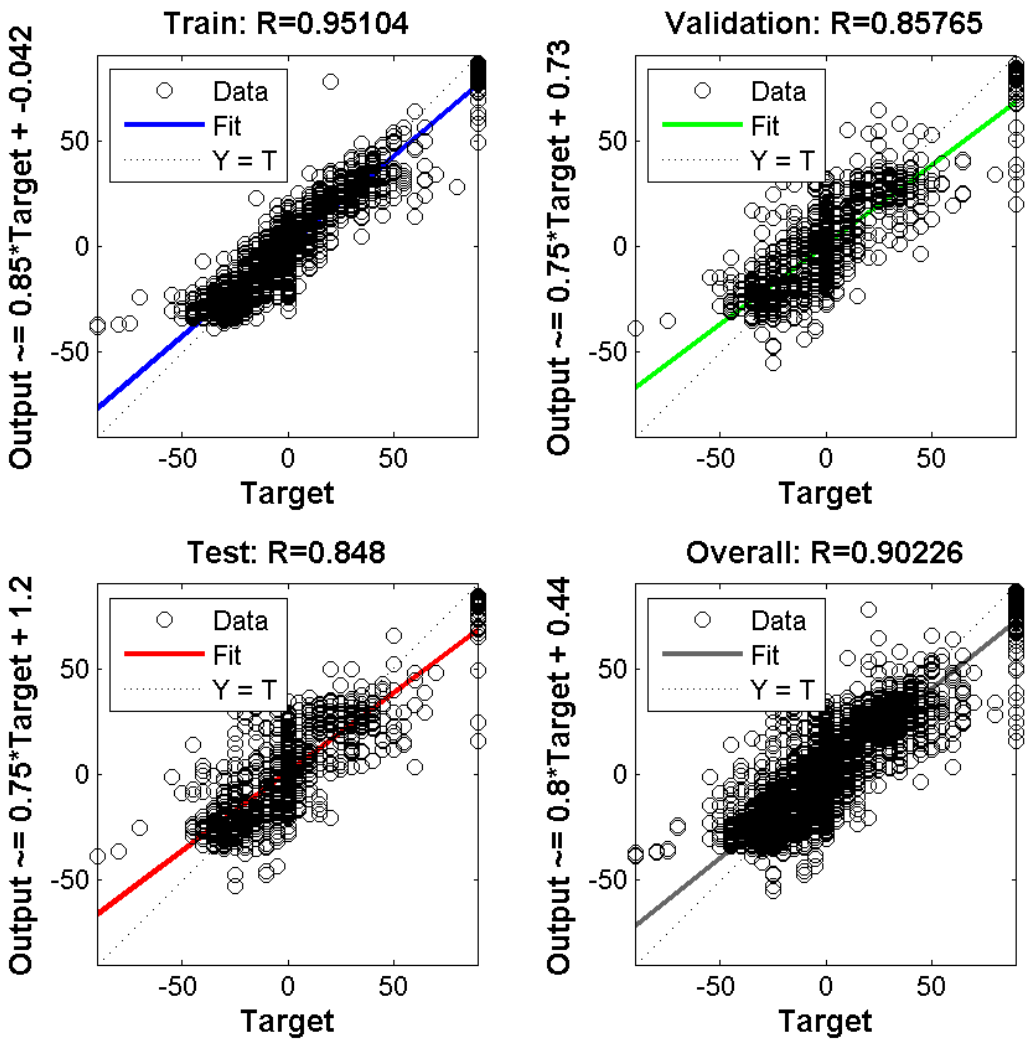


Figure 45**: Regression plots for training ANN-B with 1500 training samples and 1500 samples for testing and validation.**

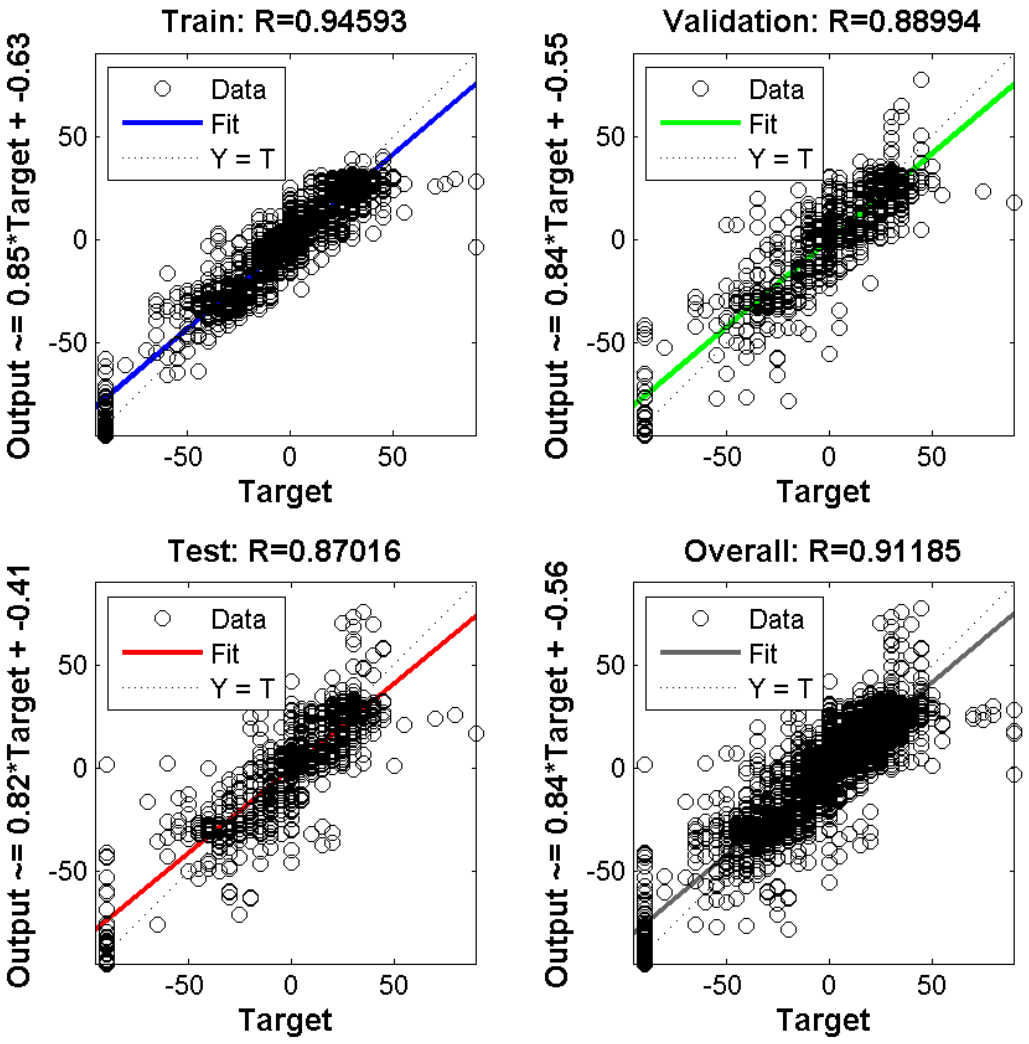


Figure 46: Regression plots for training ANN-C with 1500 training samples and 1500 samples for testing and validation.

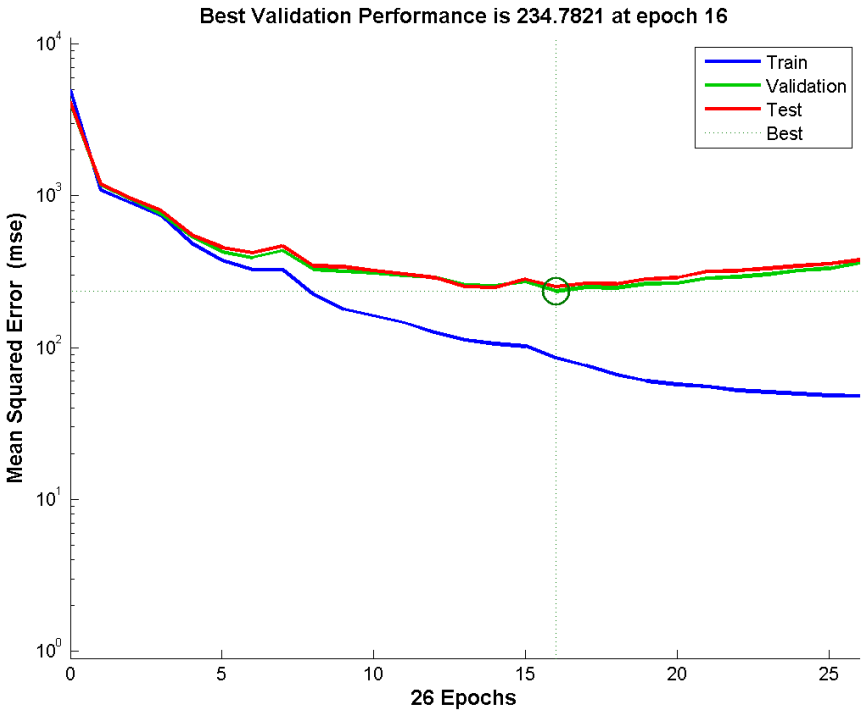


Figure 47: Performance plot for training ANN-B

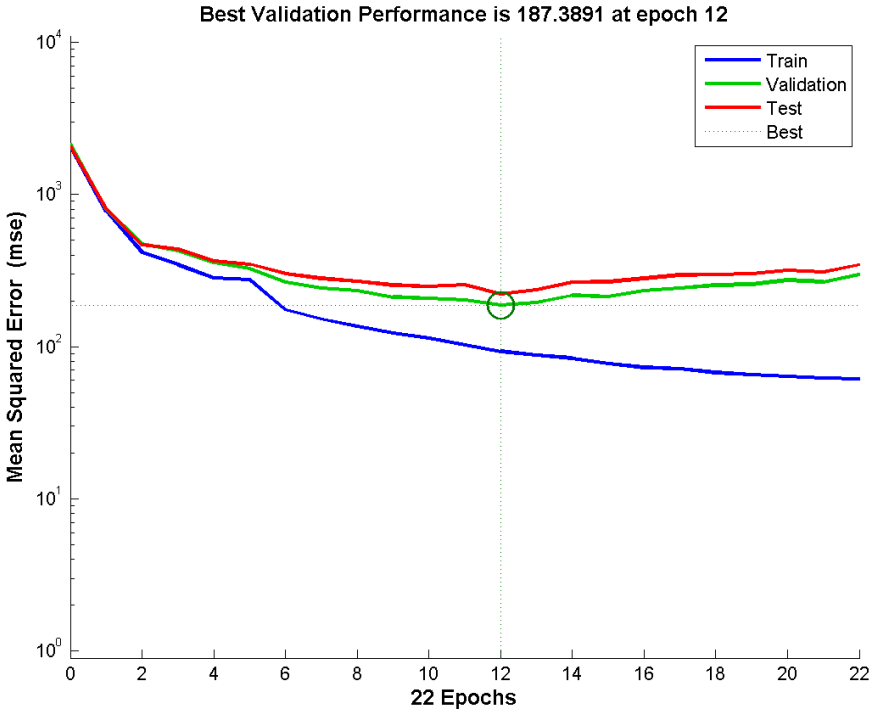


Figure 48: Performance plot for training ANN-C

4.2.2 Testing

To test our algorithm, we have conducted experiments in several environments as shown in Figure 49. Three environments have been replicated from [19] (Figure 49(a-c)), comparing their results to ours. Another environment created by ourselves (Figure 49 (d)) to show the wall following and object avoidance actions in a more difficult environment. Experiments are done using three different robots (P2AT, Talon and Zerg). Three different sensors (laser range scanner, 8-sonar sensors and 5-sonar sensors) have been mounted on to each robot. The orientations of the sensors are the same in all three robots. However, due to dissimilarity in the robots' dimensions, the locations of the mounted sensors are different.

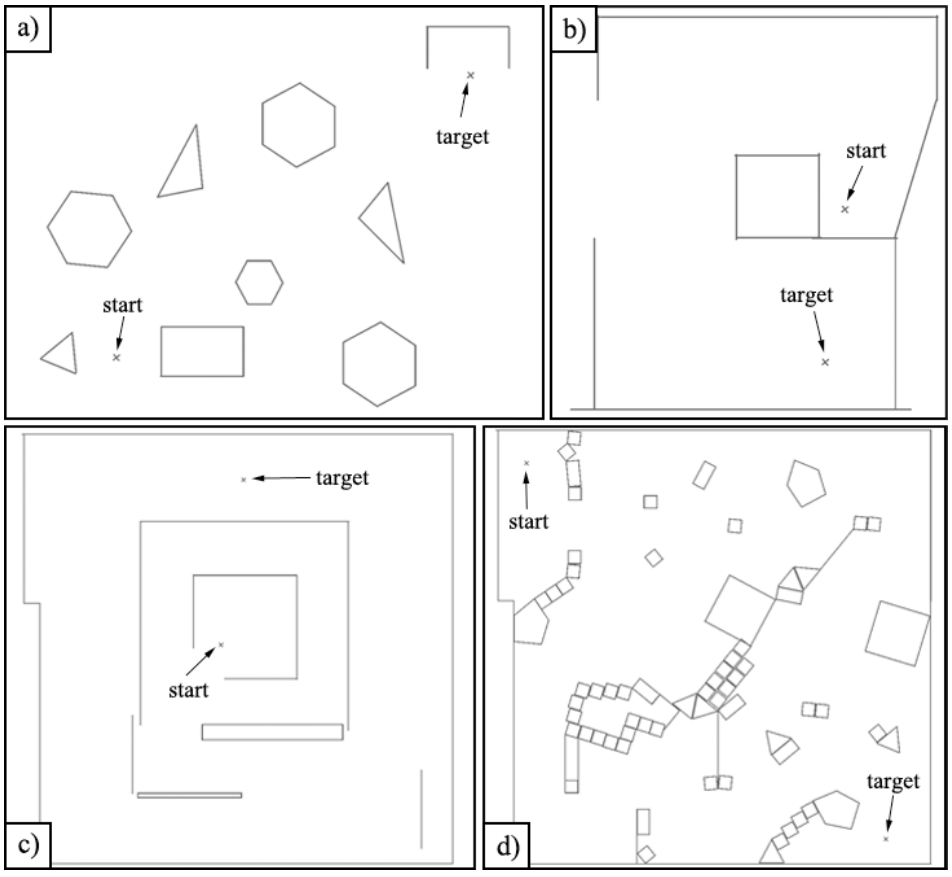


Figure 49: Four environments used for testing our algorithm

Environment 1: Figure 50 shows the first environment selected for our experiments. This figure shows the experimental result and simulation result conducted by Parhi and Singh in [19]. We have simulated this environment in Unreal Tournament's engine as shown in Figure 51. The dimensions of this environment are width=12.4m and height=9.2m. Figure 52 through Figure 57 show our simulation results for the three robots in the first environment. In this environment only the object avoidance action is used, as there are no wall shaped objects or very large obstacles in the path of the robot.

Figure 52, Figure 53 and Figure 54 display the navigation results (paths) from starting position to the target for all three sensor types for P2AT, Talon and Zerg respectively. As it can be seen from these images, the navigation paths of laser scanner and 8-sonar sensors are very similar. The path traversed while using 5-sonar sensors is to some extend different from the other two sensors. However, it still has a successful navigation from the starting point to the goal.

Figure 55, Figure 56 and Figure 57 show navigation paths of all three mobile robots using laser range scanner, 8-sonar sensors and 5-sonar sensors respectively. In these images we compare the paths generated using one type of sensor for different mobile robots. The results show how paths traversed by different types of robots using one kind of sensor are very similar regardless of the dimensions of the robots and how the sensors are mounted on the mobile robots.

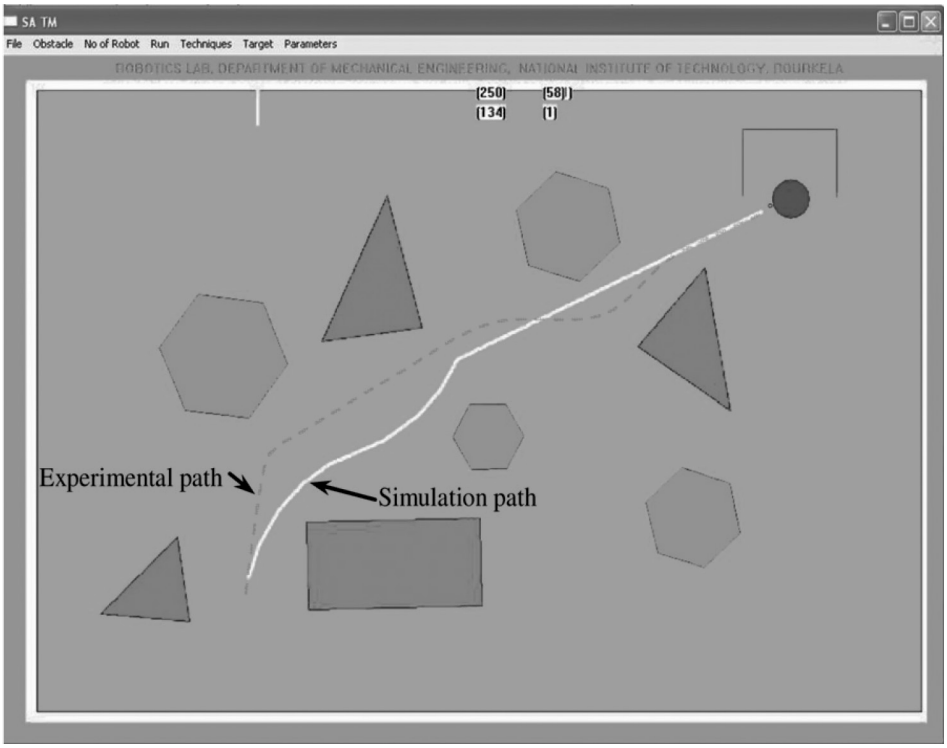


Figure 50: Experimental and simulation results from [19] **in environment 1**

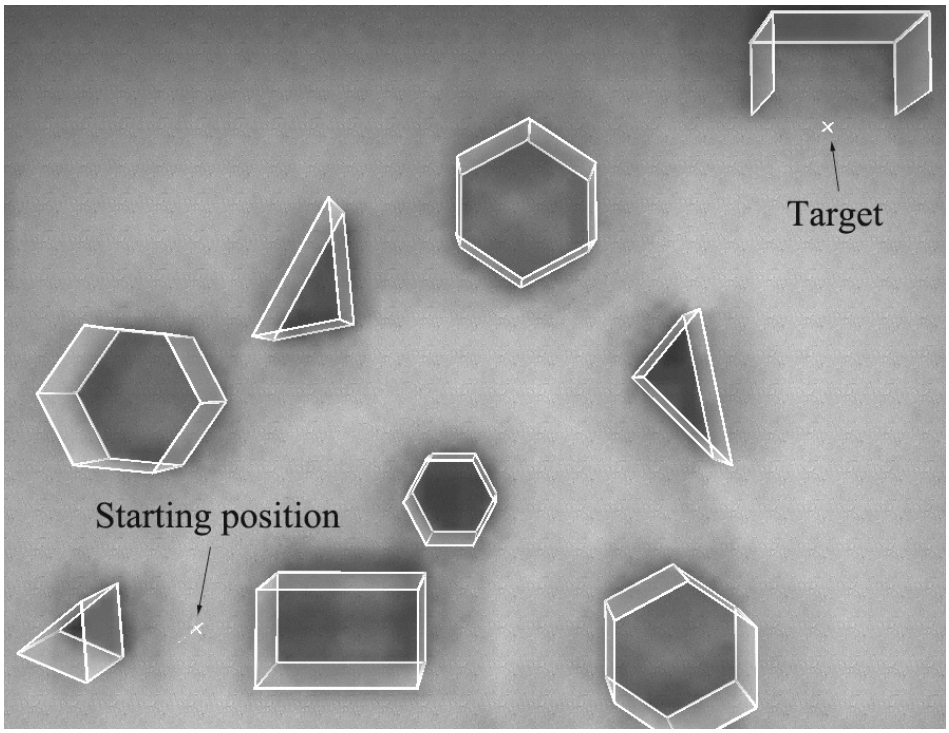


Figure 51: Simulated environment #1 in Unreal Tournament engine.

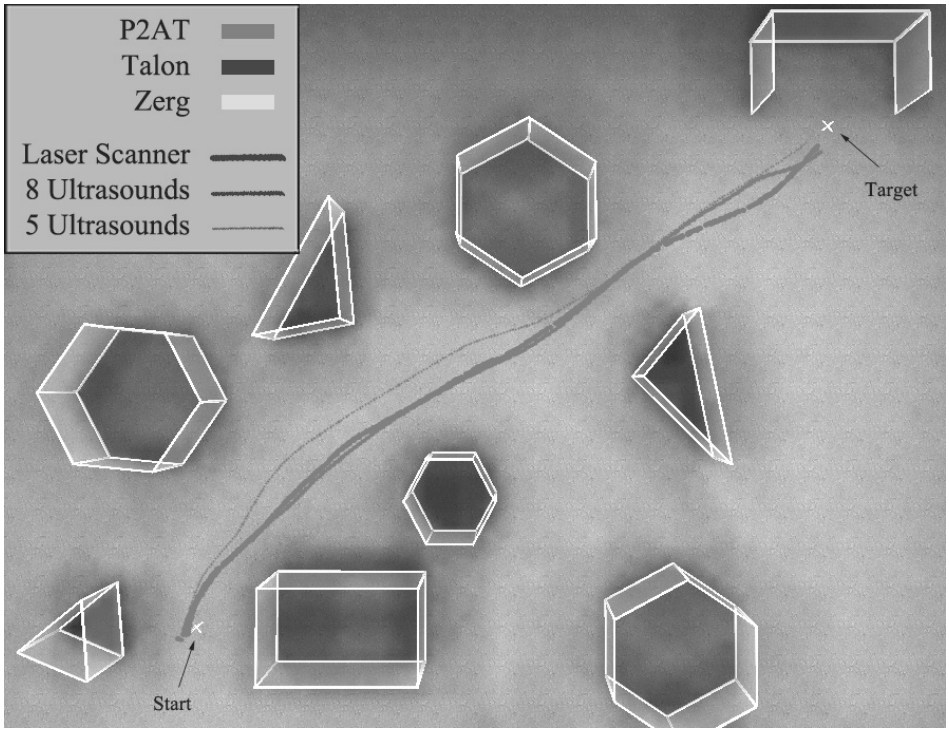


Figure 52: Simulation results for P2AT using three different sensors in environment 1.

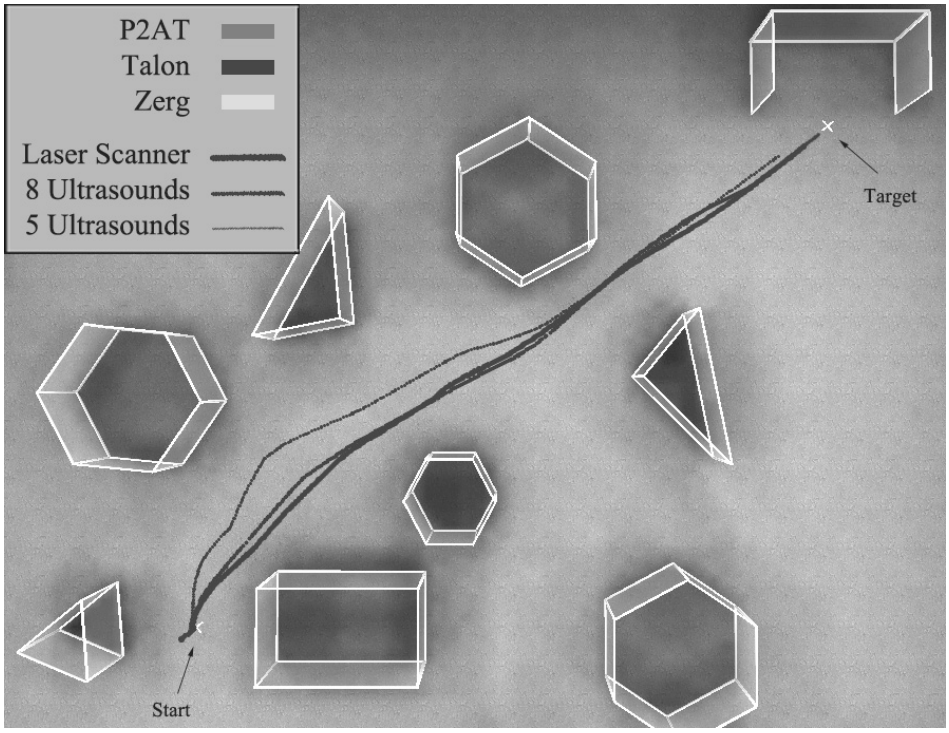


Figure 53: Simulation results for Talon using three different sensors in environment 1.

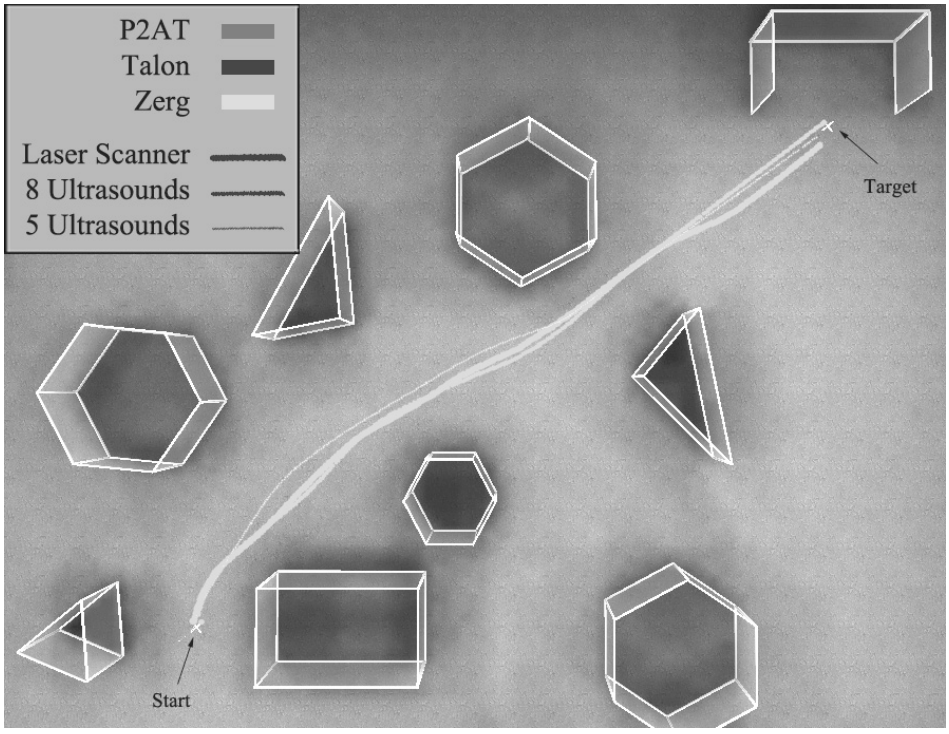


Figure 54: Simulation results for Zerg using three different sensors in environment 1.

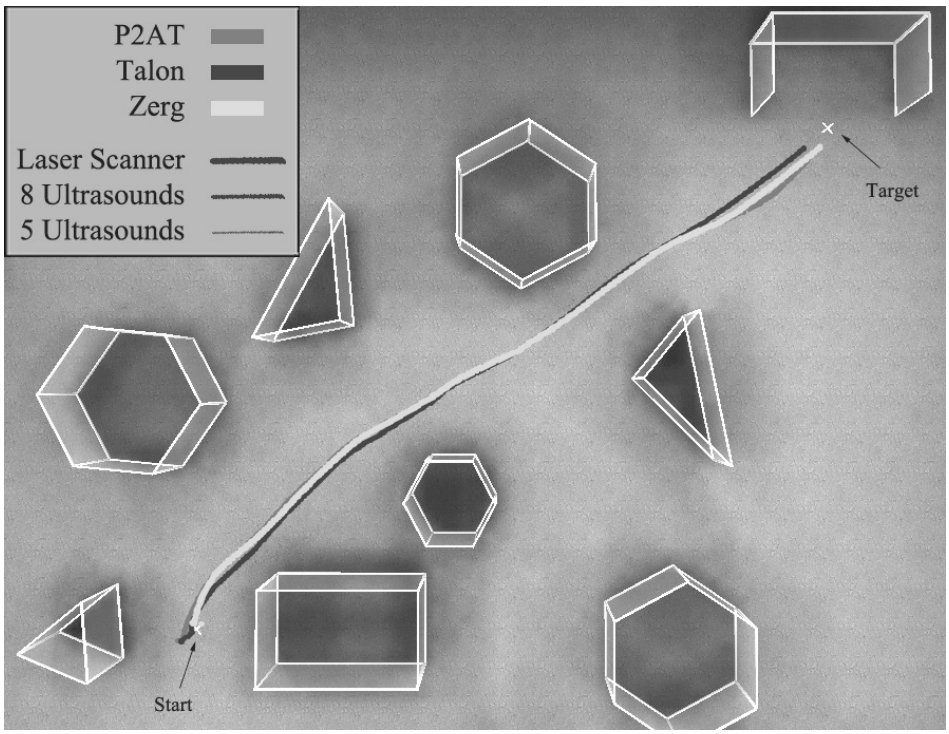


Figure 55: Simulation results for P2AT, Talon and Zerg using laser range scanner in environment 1.

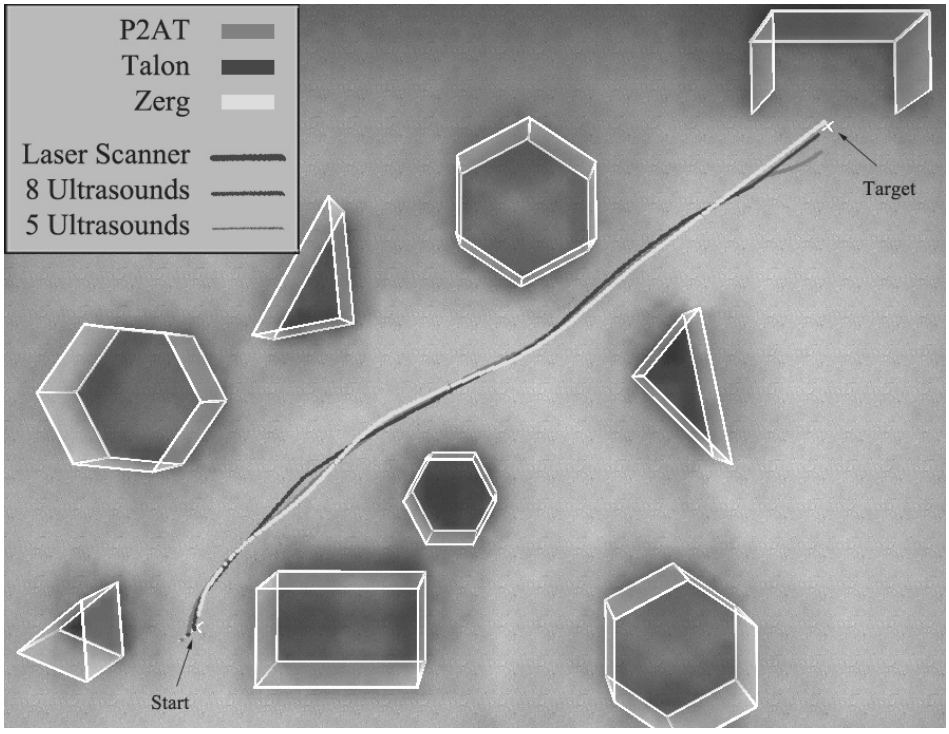


Figure 56: Simulation results for P2AT, Talon and Zerg using 8-sonar sensors in environment 1.

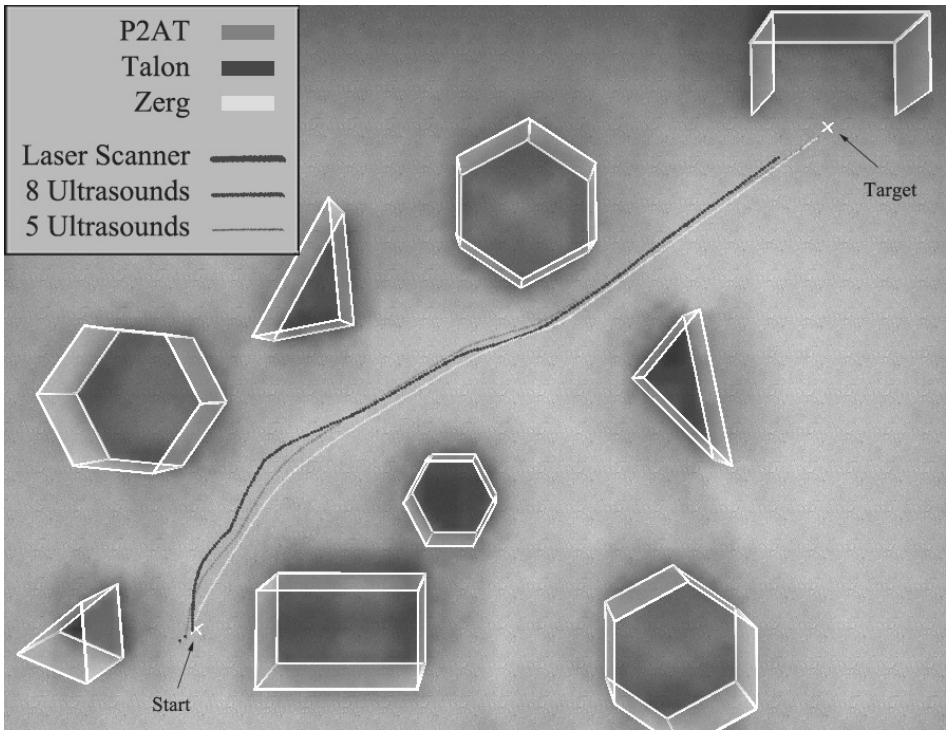


Figure 57: Simulation results for P2AT, Talon and Zerg using 5-sonar sensors in environment 1.

Environment 2: Figure 58 shows the simulation path conducted by Parhi and Singh in [19] in the second environment. Figure 59 depicts the simulation of the second environment in Unreal Tournament's engine. The dimensions of this environment are width=8m and height=9.2m. Figure 60 through Figure 65 show our simulation results for the three robots in the second environment. This environment is designed to test the ability of the robot in wall following in a very simple scenario.

Figure 60, Figure 61 and Figure 62 display the navigation results (paths) from starting position to the target for all three sensor types for P2AT, Talon and Zerg respectively. As it can be seen from these images, the navigation paths of laser scanner and 8-sonar sensors are very similar. The path traversed while using 5-sonar sensors is to some extend different from the other two sensors. However, it still has a successful navigation from the starting point to the goal. From these results we can see that the navigation path using different sensors are very different. As it can be seen the paths traversed using the laser sensors are shorter than the other two sensors.

Figure 63, Figure 64 and Figure 65 show navigation paths of all three mobile robots using laser range scanner, 8-sonar sensors and 5-sonar sensors respectively. In these images we compare the paths generated using one type of sensor for different mobile robots. Same as the previous environment, these results also show how paths traversed by different types of robots using one kind of sensor are almost the same regardless of the dimensions of the robots and how the sensors are mounted on them.

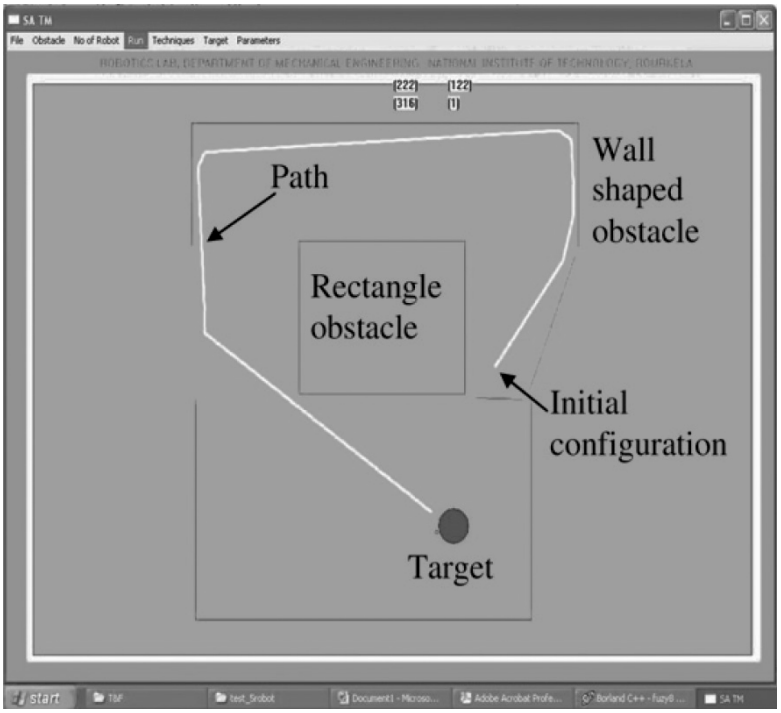


Figure 58: Simulation result from [19] **in environment #2**

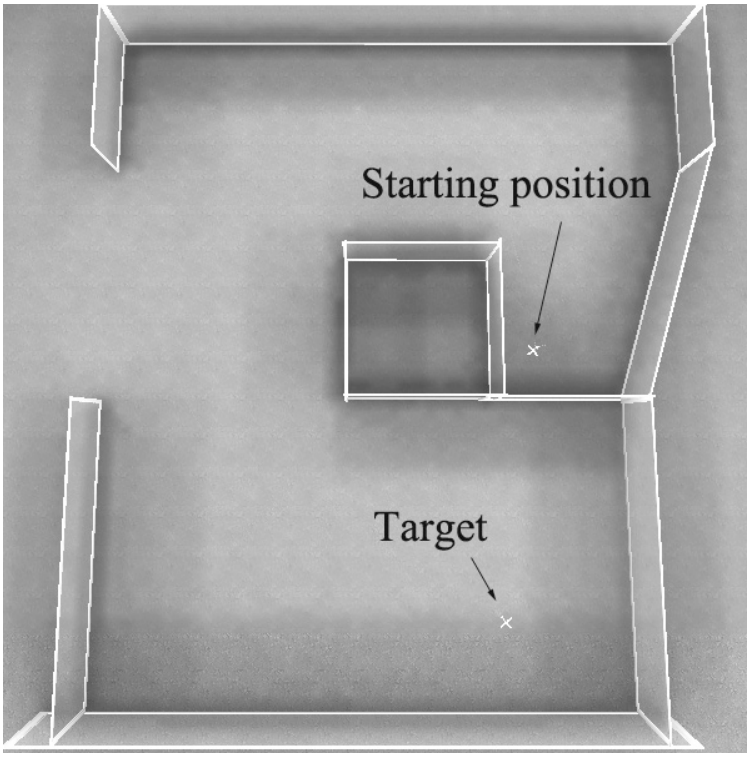


Figure 59: Simulated environment #2 in Unreal Tournament engine

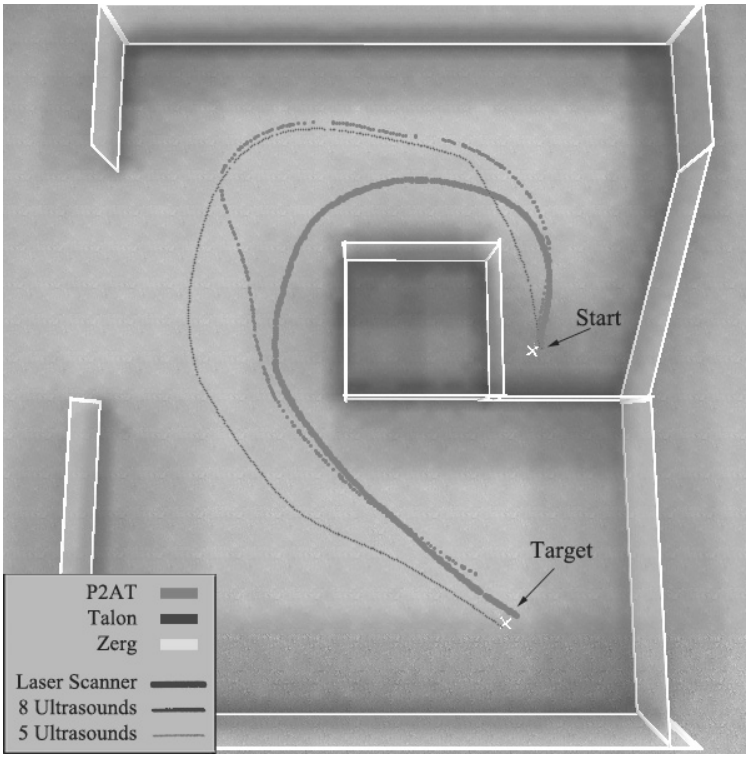


Figure 60: Simulation results for P2AT using three different sensors in environment #2

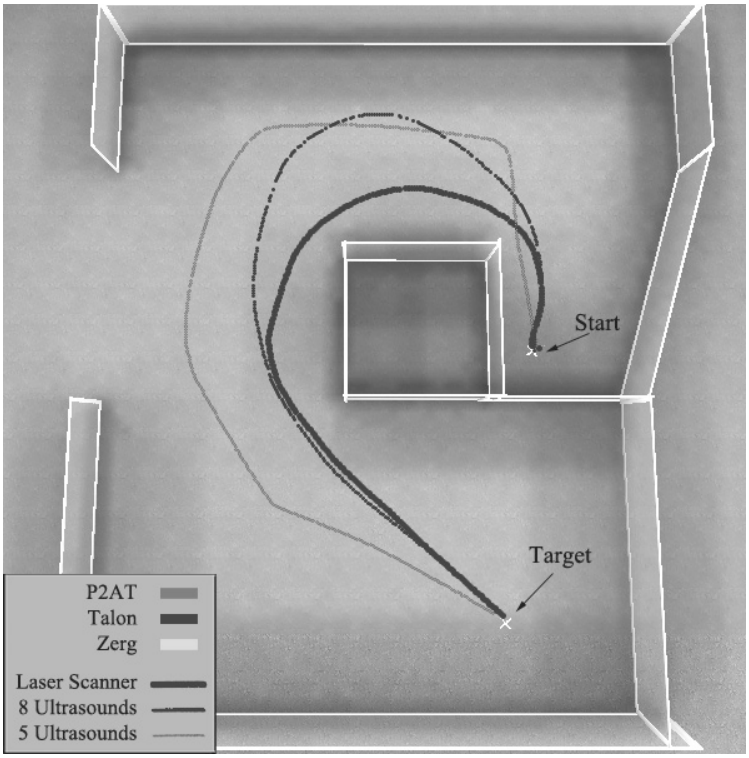


Figure 61: Simulation results for Talon using three different sensors in environment #2

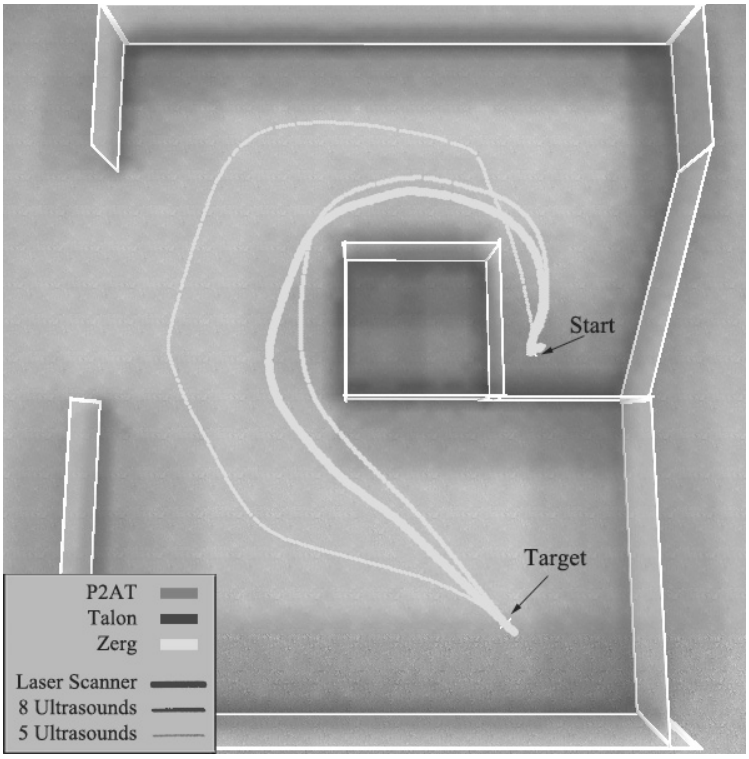


Figure 62: Simulation results for Zerg using three different sensors in environment #2

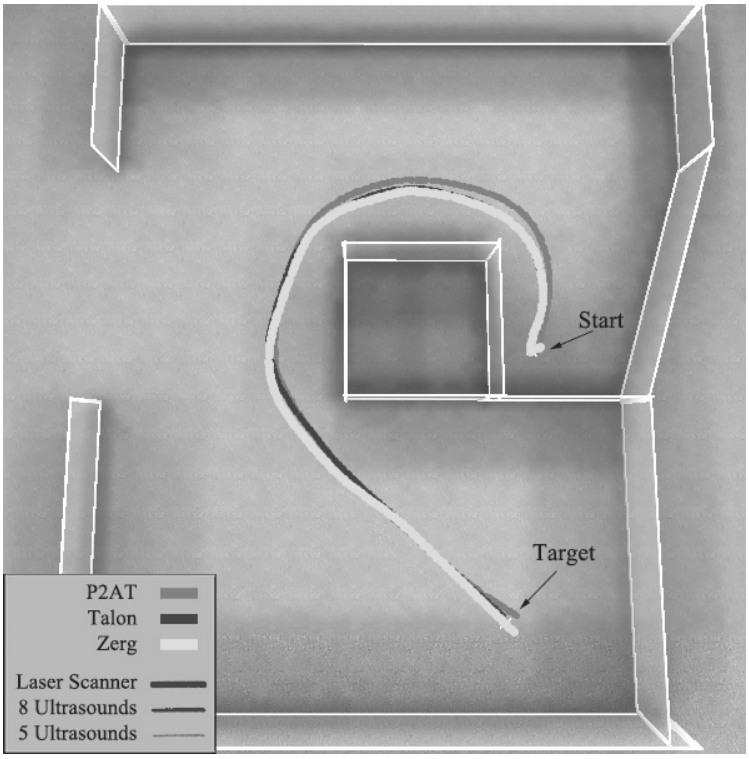


Figure 63: Simulation results for P2AT, Talon and Zerg using laser scanner in environment #2

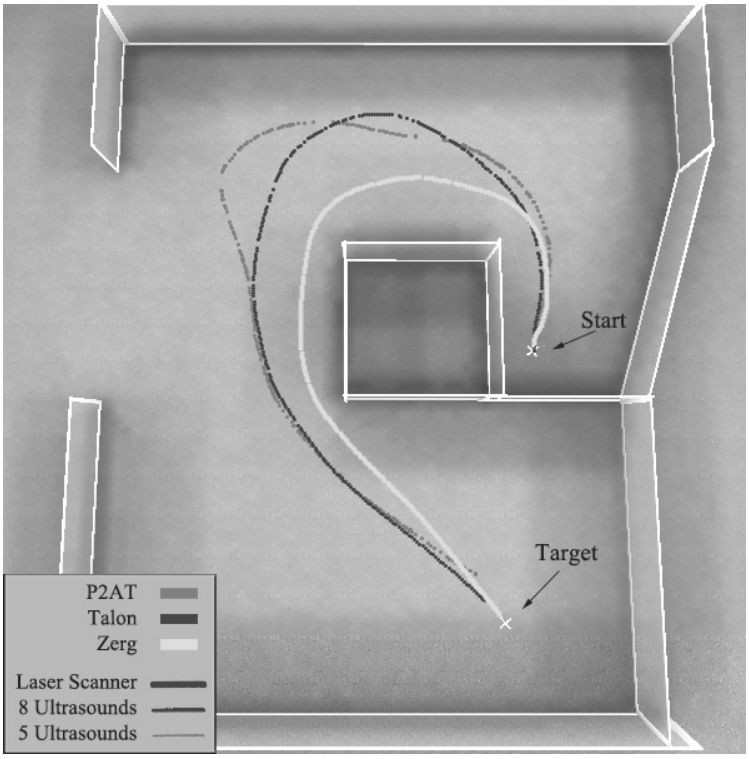


Figure 64: Simulation results for P2AT, Talon and Zerg using 8-sonar sensors in environment #2

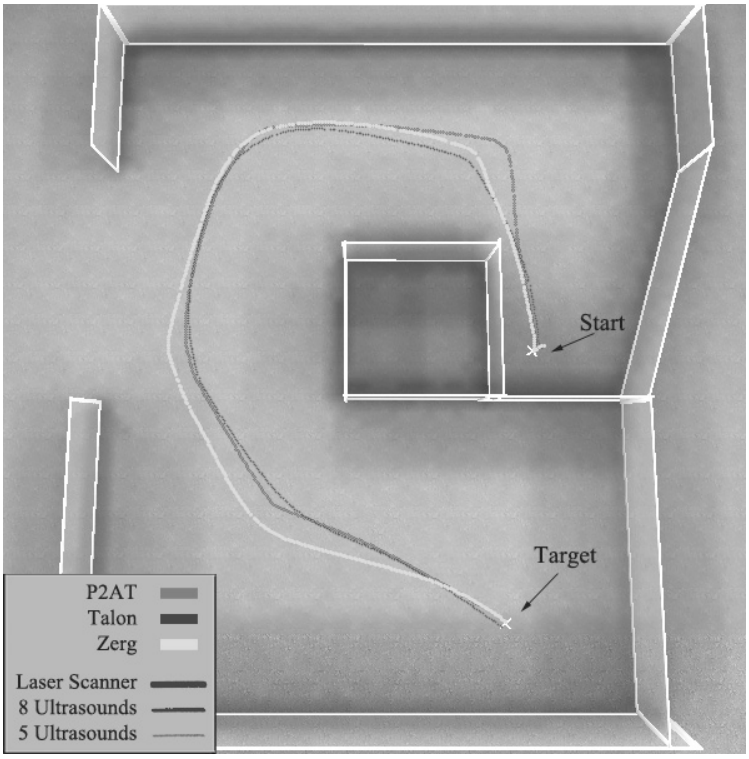


Figure 65: Simulation results for P2AT, Talon and Zerg using 5-sonar sensors in environment #2

Environment 3: Figure 66 shows the navigation path of the experiment performed by Parhi and Singh in [19] in the third environment. Figure 67 illustrates the simulation of the third environment in Unreal Tournament's engine. The dimensions of this environment are width=16m and height=16.2m. Figure 68 through Figure 73 show our simulation results for the three robots in the third environment. This environment is designed to test the ability of the robot in wall following in a more complicated scenario than the last one.

Figure 68, Figure 69 and Figure 70 display the navigation results (paths) from starting position to the target for all three sensor types for P2AT, Talon and Zerg respectively. By comparing the paths of laser range scanner, 8 sonar sensors and 5 sonar sensors in these images, we can see that the results are not very similar as it was the case in the previous two environments. This illustrates the fact that the size and complicity of the environment has a direct affect on the paths traversed using different sensors. However, we can still see a successful navigation from the starting point to the target. From these results we can conclude that the navigation path using different sensors are very different. Also this might cause different navigation routes as depicted in Figure 70.

Figure 71, Figure 72 and Figure 73 show navigation paths of all three mobile robots using laser range scanner, 8-sonar sensors and 5-sonar sensors respectively. In these images we compare the paths generated using one type of sensor for different mobile robots. Contrary to what is seen in the previous environments, these results show paths traversed by different types of robots in large and complicated environments which are not that similar. In fact, the dimensions of the robots and how the sensors are mounted on them have some affects in these kinds of environments.

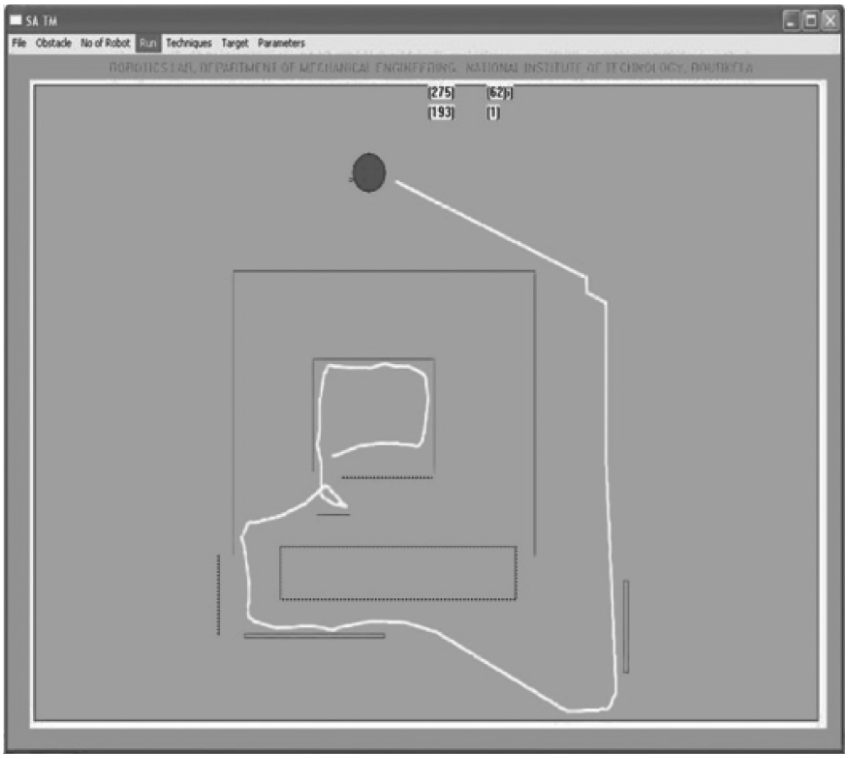


Figure 66: Simulation result from [19] **in environment #2**

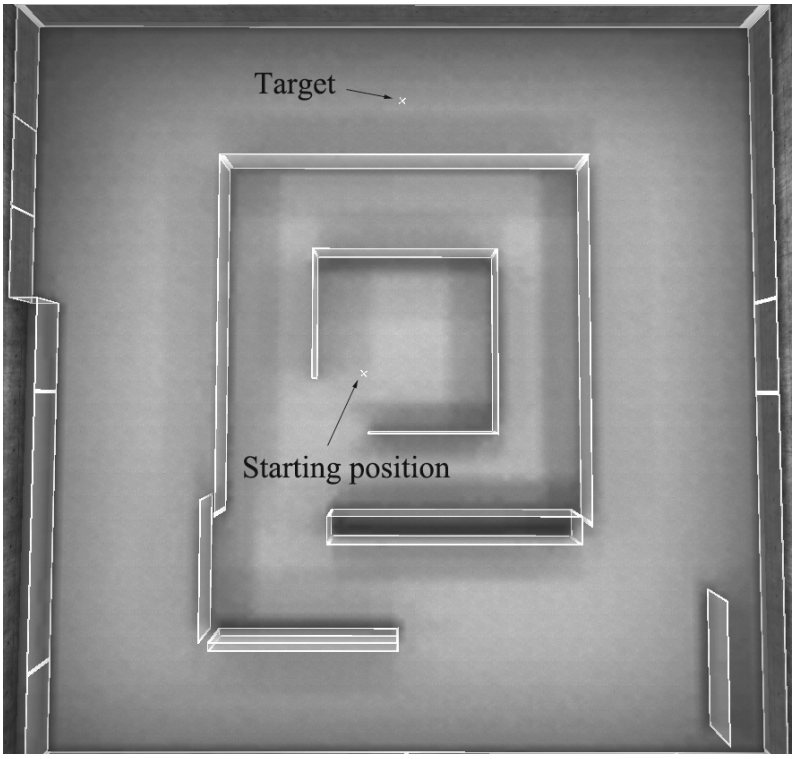


Figure 67: Simulated environment #3 in Unreal Tournament engine

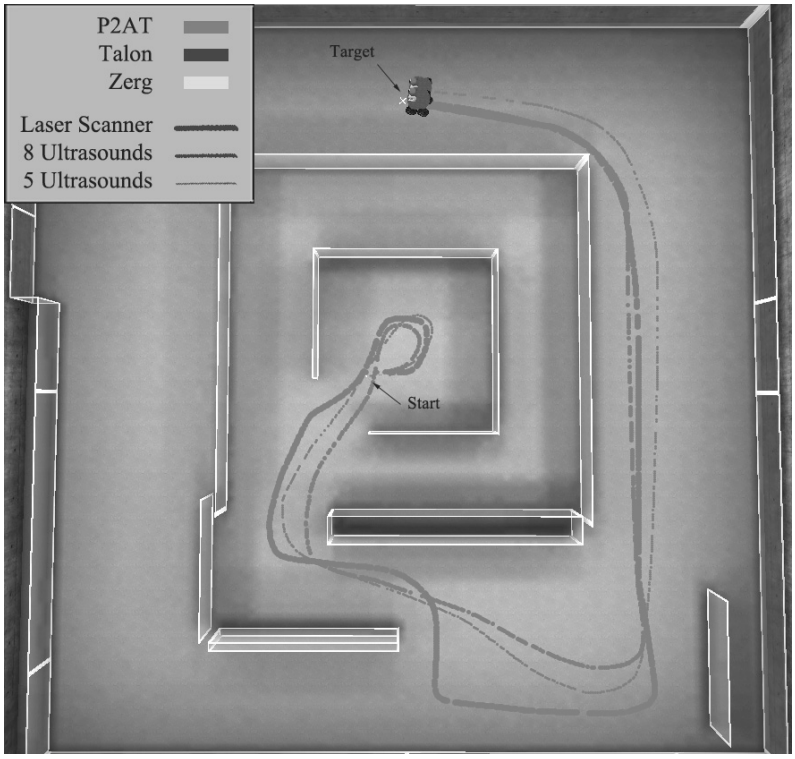


Figure 68: Simulation results for P2AT using three different sensors in environment #3

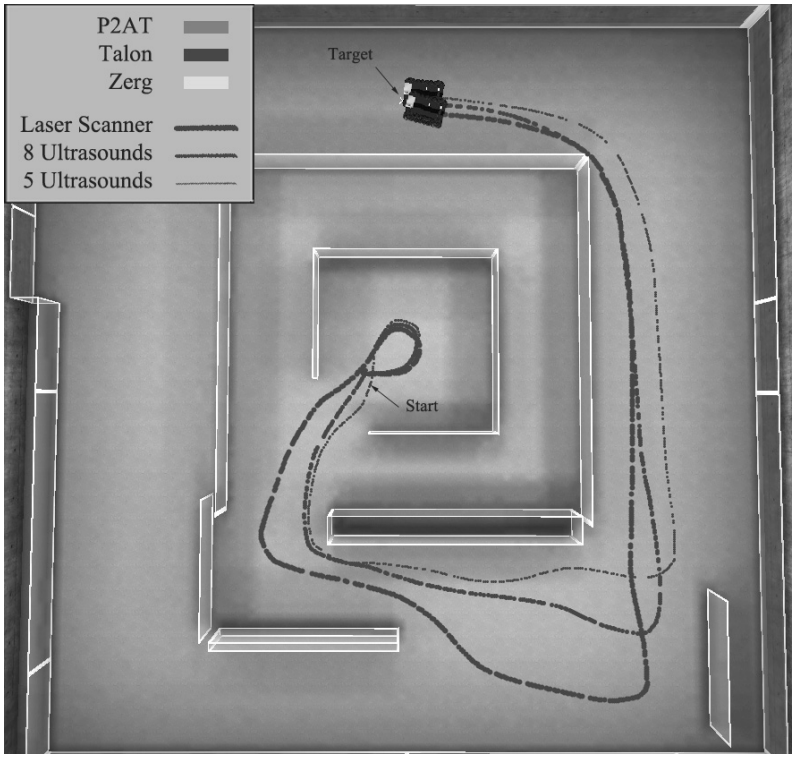


Figure 69: Simulation results for Talon using three different sensors in environment #3

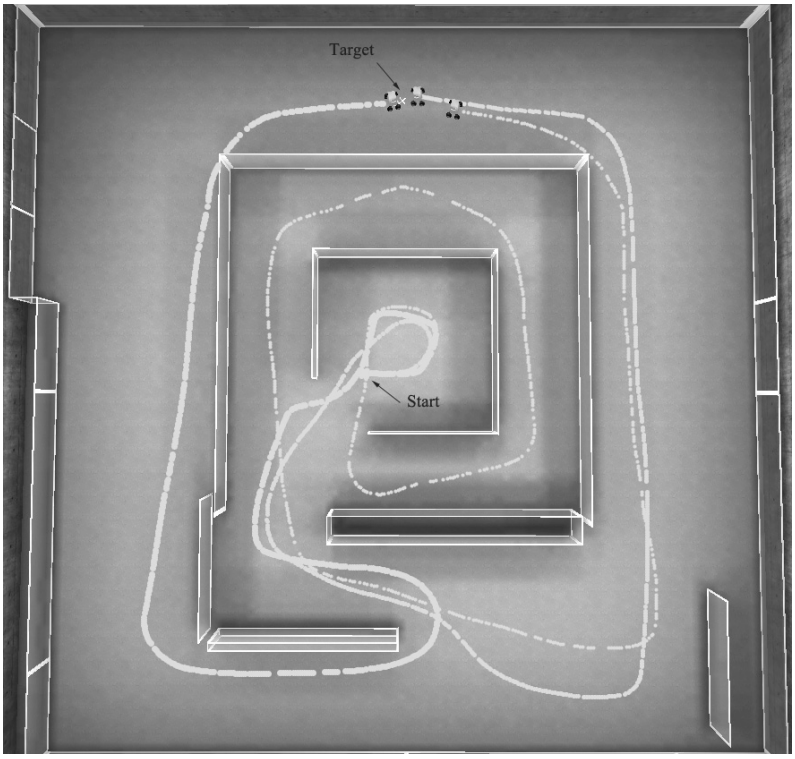


Figure 70: Simulation results for Zerg using three different sensors in environment #3

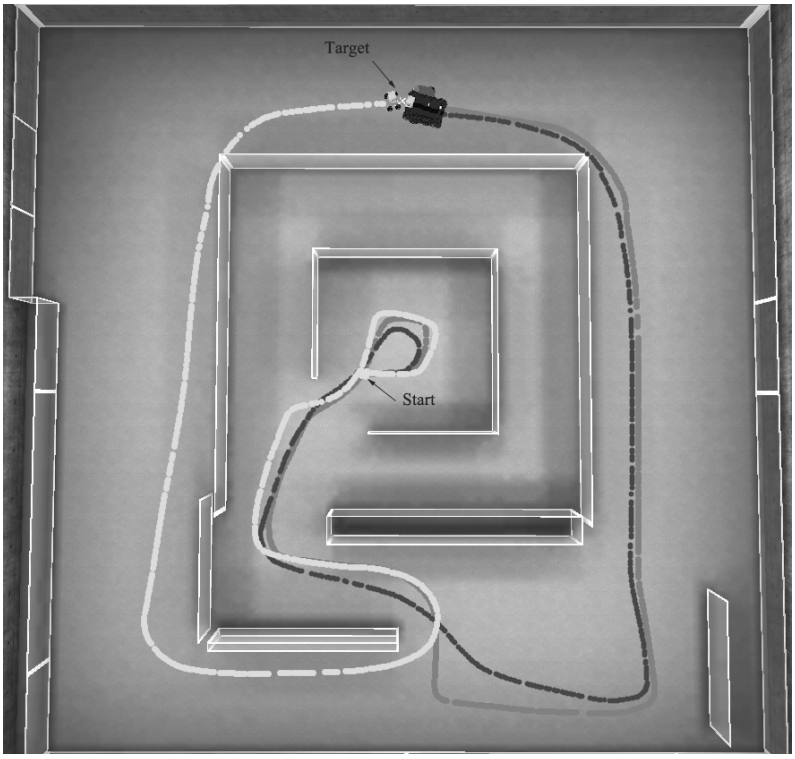


Figure 71: Simulation results for P2AT, Talon and Zerg using laser scanner in environment #3

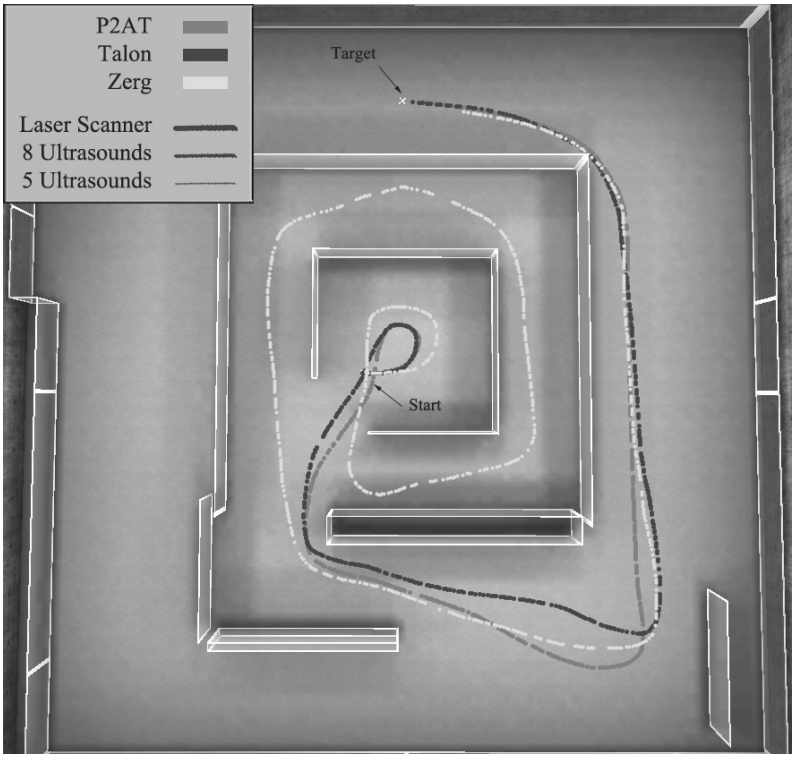


Figure 72: Simulation results for P2AT, Talon and Zerg using 8-sonar sensors in environment #3

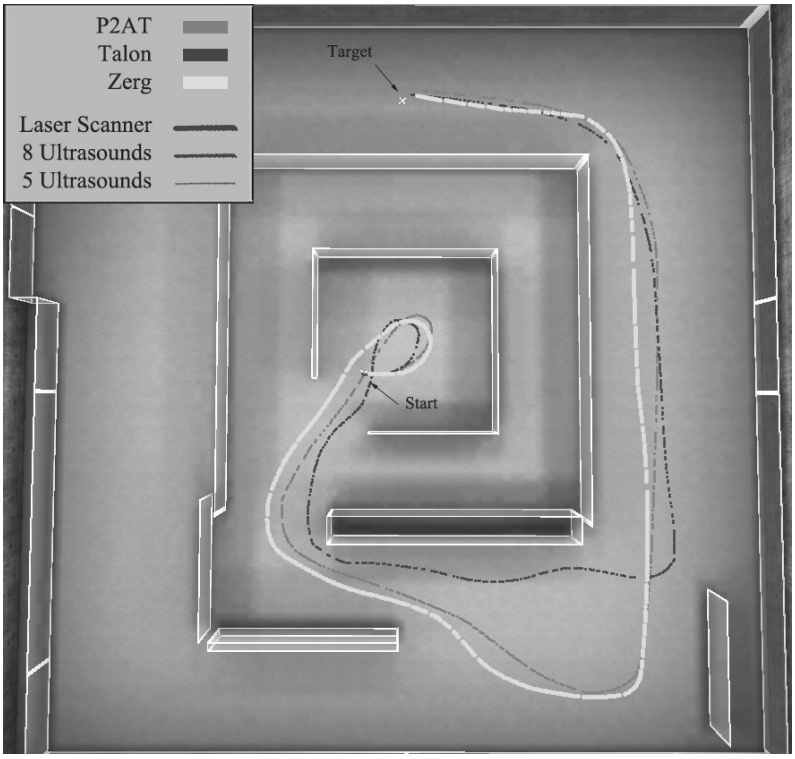


Figure 73: Simulation results for P2AT, Talon and Zerg using 5-sonar sensors in environment #3

Environment 4: We have designed a more complex scenario where the robot constantly needs to switch between wall-following and object-avoidance actions in order to safely reach its destination. Figure 74 shows a 3D view of the simulated environment in Unreal Tournament's engine. The dimensions of this environment are 16m (width) by 16.2m (height). Figure 75 through Figure 80 show our simulation results for the three robots in this environment.

Figure 75, Figure 76 and Figure 77 display the navigation paths of P2AT, Talon and Zerg respectively, from the starting position to the target for all three sensor types. By comparing the paths of laser range scanner, 8 sonar sensors and 5 sonar sensors in these images, we can see similar results as in previous environments. Also a successful navigation can be seen for all three types of robots. It is noteworthy to mention that as depicted in Figure 77, we can notice that the Zerg robot tends to take different routes than the other two mobile robots while using sonar sensors.

Figure 78, Figure 79 and Figure 80 show navigation paths of all three mobile robots using laser range scanner, 8-sonar sensors and 5-sonar sensors respectively. In these images we compare the paths generated by the three mobile robots using one type of sensor for each image. Similar to what we saw in the previous environment, these results also show that paths traversed by different types of robots in large and complicated environments are not very alike.

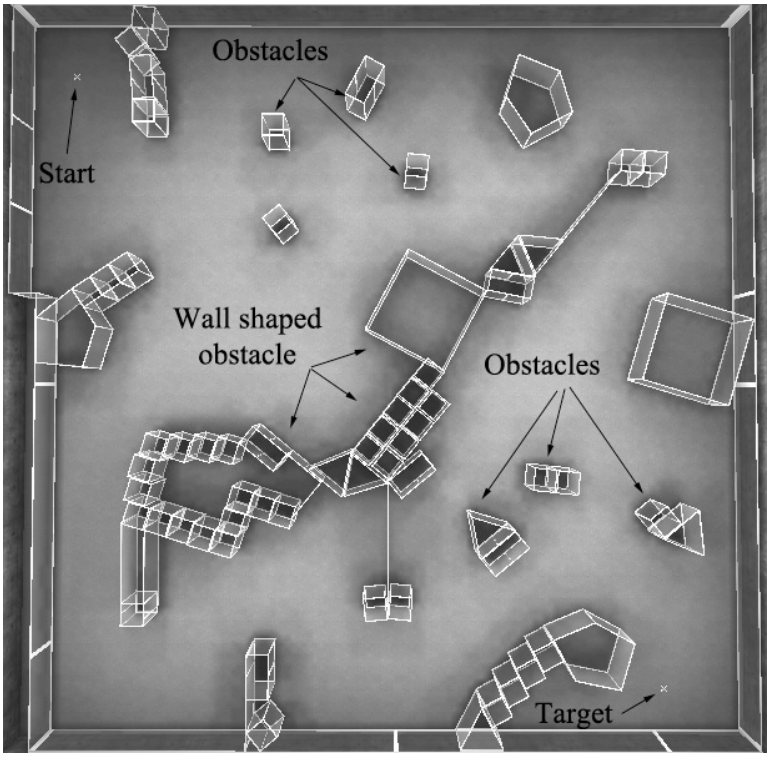


Figure 74: Simulation of environment #4 in Unreal Tournament engine

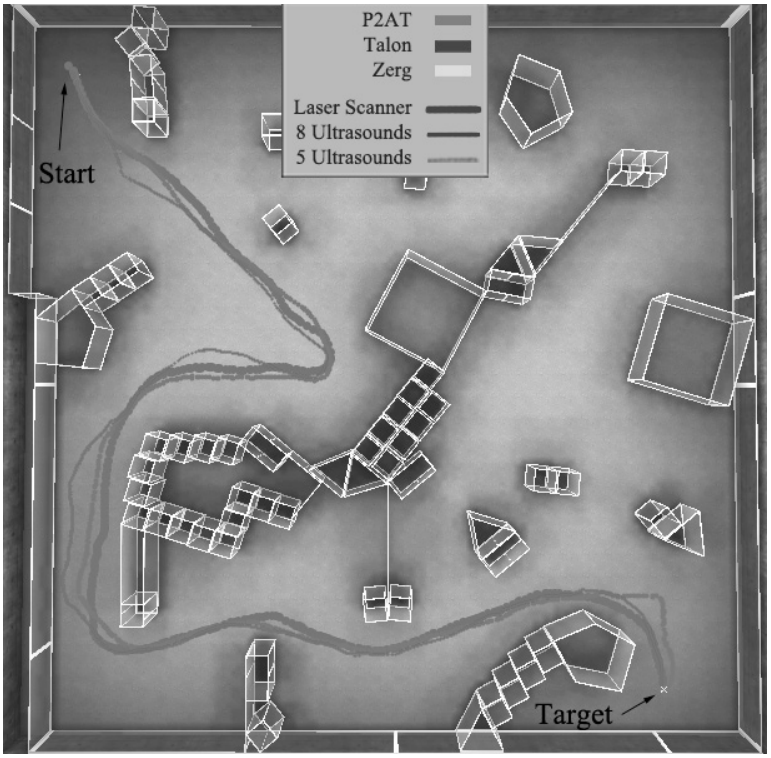


Figure 75: Simulation results for P2AT using three different sensors in environment #4

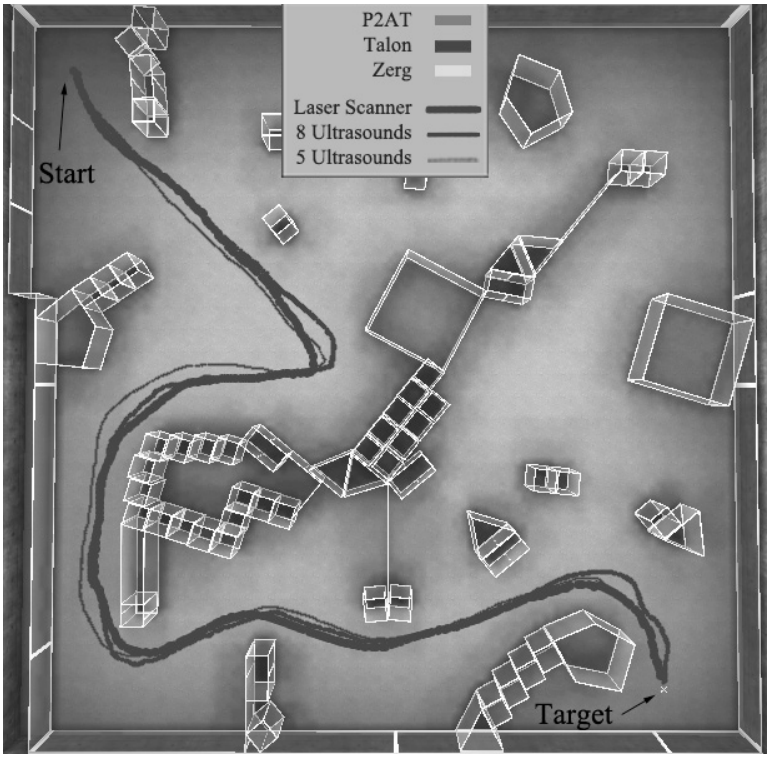


Figure 76: Simulation results for Talon using three different sensors in environment #4

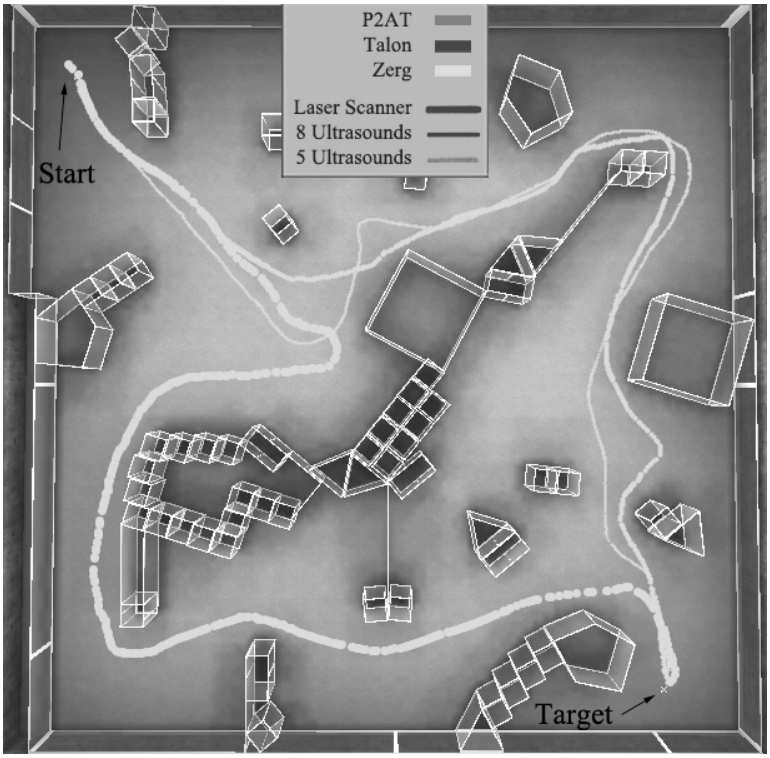


Figure 77: Simulation results for Zerg using three different sensors in environment #4

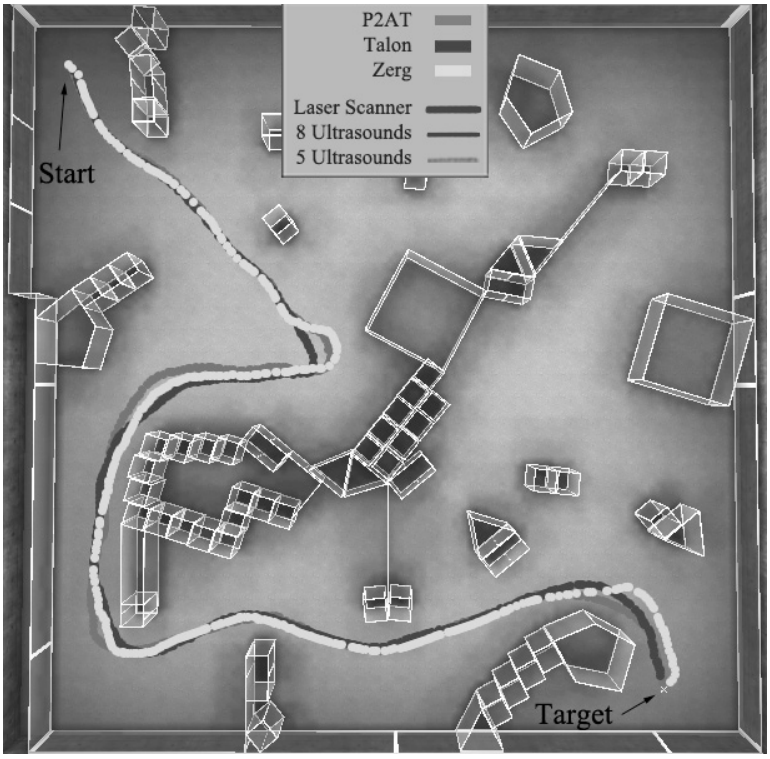


Figure 78: Simulation results for P2AT, Talon and Zerg using laser range scanner in environment #4

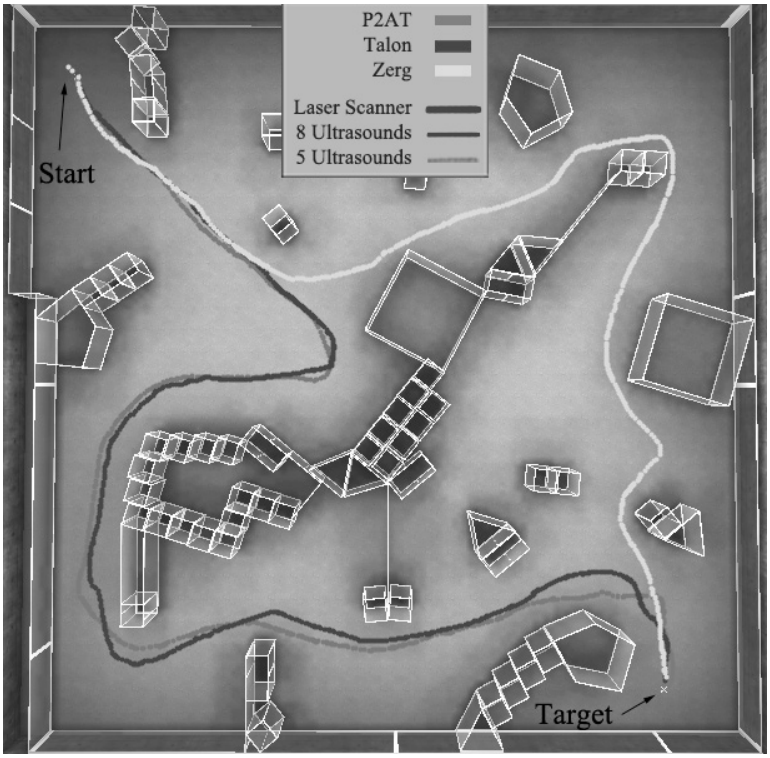


Figure 79: Simulation results for P2AT, Talon and Zerg using 8-sonar sensors in environment #4

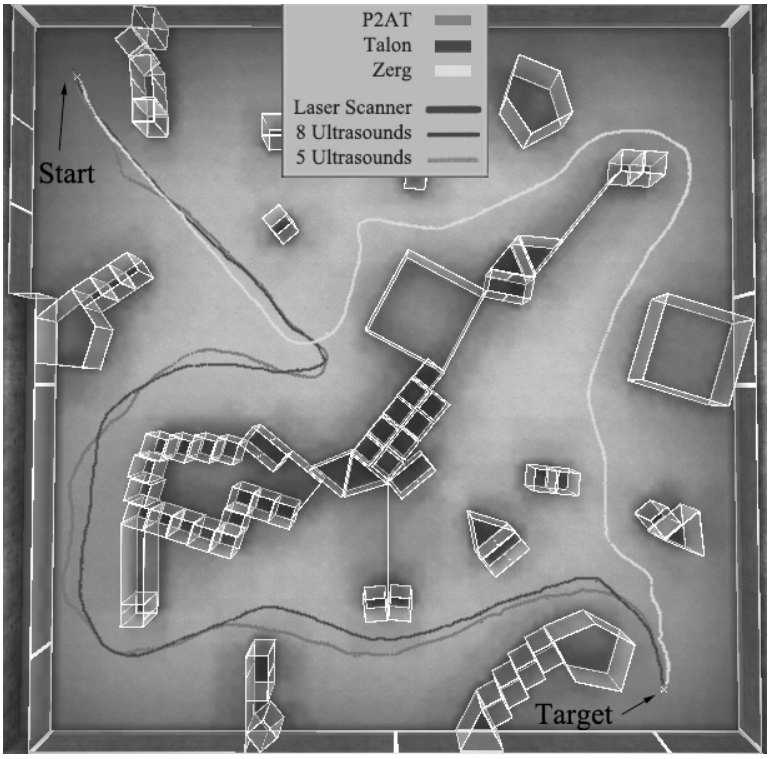


Figure 80: Simulation results for P2AT, Talon and Zerg using 5-sonar sensors in environment #4

Chapter 5

Conclusions and Future Works

5.1 Conclusion

In this thesis, a general approach for interpretation of different types of sensors, such as laser range scanner and ultrasonic sensors has been developed. A system for navigation of mobile robots using these interpretations has also been developed. Existing neural network navigation approaches only deal with one kind of robot and a specific sensor. Therefore, they will lead to different network structures for each type of sensor or robot and high overall training times. In our approach we proposed a generalized method to interpret different kinds of 2-dimensional- sensors. Instead of training the networks for all types of sensors, our approach only trains the networks using one of the most accurate 2-dimensional sensors. Therefore, it avoids unnecessary training periods and the necessity to gather training samples for different kinds of sensors.

Experimental results, carried out in simulated environments, demonstrate that our approach can be positively affective in mobile robot navigation for different kinds of robots and sensors, when compared to previous works. Therefore, our proposed globalized navigation algorithm can yield significant navigation results – at less training time and lower sensor costs.

5.2 Future Work

The most difficult part of these experiments proved to be the gathering of the training samples. This is a very time consuming and frustrating task and requires very accurate definitions of the navigation problems. Rather than manually gathering the

training samples from the environment and introducing all the training patterns to the network at once, one could use online learning algorithms [97,98]. In online learning, the aim is to predict labels for samples. The main defining feature of online learning is that after a prediction is made, the true desired output of the sample is discovered. This information, therefore, can then be used to enhance the prediction hypothesis used by the algorithm. Another characteristic of online algorithm is that it can process inputs in the order that they are provided to the algorithm, one fraction at a time in a serial fashion, without having the entire input available from the start. Presently, our networks produce outputs based on offline learning where we provide all the training samples at the beginning. Hence, an enhancement to our algorithm can be improving the neural network structures to comply with online learning algorithms.

Another possible solution for automatically training the networks would be reinforcement learning [99]. In reinforcement learning, data are normally generated by the interactions of an agent with the environment. At each point in time, the environment generates an observation and an immediate cost, according to some (usually unknown) dynamics based on the actions that a robot performs. The main goal is to discover a rule for deciding on which actions to select which will minimize expected cumulative cost which is some measure of a long-term cost. Therefore, instead of offline or online learning, employing reinforcement learning for training the neural networks will be an improvement to our proposed method.

However, neural networks have also some drawbacks. For instance, a neural network cannot explicitly explain its results. Furthermore, convergence to an optimal solution may not be guaranteed by the learning algorithms. Hybrid approaches, which

combine neural networks with other artificial intelligence algorithms such as fuzzy logic, knowledge-based systems and genetic algorithms, have proved to be more affective in some cases. Therefore these methods can also be tried as an improvement to our proposed navigation algorithm.

In this thesis we have only used three kinds of simulation robots and three types of sensors and configurations. Therefore, one objective for future research is to extend our algorithm to real world environments using real robots and sensors. Furthermore, other types of 2-dimensional sensors and robots can be put to test.

Another limitation of the current approach arises from the fact that the direction of the target must be known. In our experiments, we have only used scenarios that the target's direction is known. There might be cases in real world experiments that the robot is not able to recognize its target's direction. For example, assuming that there are signals being received from the target's location, which is determining the direction of that target, these signals can be blocked or interrupted by certain objects in the environment. Such limitations can be overcome by remembering the last bearing of the target. In case the target's signal is lost, then the robot will try to navigate to the last bearing of the target until it can receive the signals again. Also, the target's bearing needs to be updated based on the movement and rotations of the robot.

Despite these limitations, our approach does provide a good basis for a generalized interpretation of 2-dimensional sensors, and experimental results illustrate its effectiveness in practice. These results show that robots will perform very good navigation tasks even though the algorithm has not been trained specifically for that robot.

APPENDICES

APPENDIX A

Backpropagation Training Algorithm

Nomenclature

The nomenclature used in the training algorithm for the backpropagation net is as follows:

x Input training vector: $x = (x_1, \dots, x_i, \dots, x_n)$

t Output target vector: $t = (t_1, \dots, t_k, \dots, t_m)$.

δ_k Portion of error correction weight adjustment for w_{jk} that is due to an error at output unit Y_k; also, the information about the error at unit Y_k that is propagated back to the hidden units that feed into unit Y_k.

δ_j Portion of error correction weight adjustment for v_{ij} that is due to the backpropagation of error information from the output layer to the hidden unit Z_j.

α Learning rate.

X_i Input unit i:

For an input unit, the input signal and output signal are the same, namely, x_i.

v_{0j} Bias on hidden unit j.

Z_j Hidden unit j:

The net input to Z_j is denoted z_in_j :

$$z\_in_j = v + 0j + \sum_i x_i v_{ij}$$

The output signal (activation) of Z_j is denoted z_j:

$$z_j = j(z\_in_j)$$

w_{0k} Bias on output unit k.

Y_k Output unit k:

The net input to Y_k is denoted y_in_k:

$$y\_in_k = w_{0k} + \sum_j z_j w_{jk}$$

The output signal (activation) of Yk is denoted Yk:

$$y_k = f(y\_in_k)$$

Algorithm [17]

Step 0. Initialize weights. (Set to small random values).

Step 1. While stopping condition is false, do Steps 2-9.

Step 2. For each training pair, do Steps 3-8.

Feedforward:

Step 3. Each input unit ($X_i, i = 1, \ldots, n$) receives input signal x_i and broadcasts this signal to all units in the layer above (the hidden units).

Step 4. Each hidden unit ($Z_j, j = 1, \ldots, p$) sums its weighted input signals,

$$z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

applies its activation function to compute its output signal,

$$z_j = f(z\_in_j)$$

and sends this signal to all units in the layer above (output units).

Step 5. Each output unit ($Y_k,\ k = 1,\ldots,m$) sums its weighted input signals,

$$y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

and applies its activation function to compute its output signal,

$$y_k = f(y\_in_k)$$

Backpropagation of error:

Step 6. Each output unit ($Y_k, k = 1,\ldots,m$) receives a target pattern corresponding to the input training pattern, computes its error information term,

$$\delta_k = (t_k - y_k) f'(y\_in_k)$$

calculates its weight correction term (used to update W_{jk} later),

$$\Delta w_{jk} = \alpha \delta_k z_j$$

calculates its bias correction term (used to update W_{0k} later),

$$\Delta w_{0k} = \alpha \delta_k$$

and sends δ_k to units in the layer below.

Step 7. Each hidden unit ($Z_j, j = 1,\ldots,p$) sums its delta inputs (from units in the layer above),

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

multiplies by the derivative of its activation function to calculate its error information term,

$$\delta_j = \delta\_in_j \, f'(z\_in_j)$$

calculates its weight correction term (used to update v_{ij} later),

$$\Delta v_{ij} = \alpha \delta_j x_i$$

and calculates its bias correction term (used to update V_{0j} later),

$$\Delta v_{0j} = \alpha \delta_j$$

Update weights and biases:

Step 8. Each output unit ($Y_k, k = 1, \ldots, m$) updates its bias and weights ($j = 0, \ldots, p$):

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$$

Each hidden unit ($Z_j, j = 1, \ldots, p$) updates its bias and weights ($i = 0, \ldots, n$):

$$v_{ij}(new) = v_{ij}(old) + \Delta \mathrm{v_{ij}}$$

Step 9. Test stopping condition.

An epoch is one cycle through the entire set of training vectors. Typically many epochs are required for training a backpropagation neural net. The foregoing algorithm updates the weights after each training pattern is presented. A common variation is batch updating, in which weight updates are accumulated over an entire epoch (or some other number of presentations of patterns) before being applied. [17]

REFERENCES

[1] R. Siegwart and I.R. Nourbakhsh, *Introduction to autonomous mobile robots*, The MIT Press, 2004.

[2] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The international journal of robotics research*, vol. 5, 1986, p. 90.

[3] J. Borenstein and Y. Koren, "Histogramic in-motion mapping for mobile robot obstacle avoidance," *Robotics and Automation, IEEE Transactions on*, vol. 7, 1991, pp. 535-539.

[4] S. Nagata, M. Sekiguchi, and K. Asakawa, "Mobile robot control by a structured hierarchical neural network," *Control Systems Magazine, IEEE*, vol. 10, 1990, pp. 69-76.

[5] M. Meng and A.C. Kak, "Mobile robot navigation using neural networks and nonmetrical environmental models," *Control Systems Magazine, IEEE*, vol. 13, 1993, pp. 30-39.

[6] R. Glasius, A. Komoda, and S.C.A.M. Gielen, "Neural network dynamics for path planning and obstacle avoidance," *Neural Networks*, vol. 8, 1995, pp. 125–133.

[7] P. Gaudiano and C. Chang, "Adaptive obstacle avoidance with a neural network for operant conditioning: experiments with real robots," *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, IEEE, 1997, pp. 13–18.

[8] C. Silva, M. Crisostomo, and B. Ribeiro, "MONODA: a neural modular architecture for obstacle avoidance without knowledge of the environment," *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, 2000, pp. 334-339 vol.6.

[9] C.E. Thorpe and T. Foreword By-Kanade, *Vision and Navigation: The Carnegie Mellon Navlab*, Kluwer Academic Publishers, 1990.

[10] S.G. Goodridge and R.C. Luo, "Fuzzy behavior fusion for reactive control of an autonomous mobile robot: MARGE," *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, 1994, pp. 1622-1627.

[11] H. Beom and H. Cho, "A sensor-based obstacle avoidance controller for a mobile robot using fuzzy logic and neural network," *Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on*, IEEE, 1992, pp. 1470–1475.

[12] O. Azouaoui, M. Ouaaz, a Chohra, a Farah, and K. Achour, "Fuzzy ArtMap neural network (FAMNN) based collision avoidance approach for autonomous robotic systems (ARS)," *Proceedings of the Second International Workshop on Robot Motion and Control. RoMoCo'01 (IEEE Cat. No.01EX535)*, 2001, pp. 285-290.

[13] D.R. Parhi, "Neuro-Fuzzy Navigation Technique for Control of Mobile Robots," *Analysis*.

[14] C.M. Bishop, *Pattern recognition and machine learning*, Springer New York, 2006.

[15] T.M. Mitchell, *Machine learning*, Burr Ridge, IL: McGraw Hill, 1997.

[16] A.-min Zou, Z.-guang Hou, S.-yao Fu, and M. Tan, "Neural Networks for Mobile Robot Navigation : A Survey," *Science And Technology*, 2006, pp. 1218-1226.

[17] L.V. Fausett, *Fundamentals of neural networks: architectures, algorithms, and applications*, Prentice-Hall Englewood Cliffs, NJ, 1994.

[18] D. Janglová, "Neural Networks in Mobile Robot Motion," *Advanced Robotic*, vol. 1, 2004, pp. 15-22.

[19] M.K. Singh and D.R. Parhi, "Path optimisation of a mobile robot using an artificial neural network controller," *International Journal of Systems Science*, vol. 42, 2011, pp. 107-120.

[20] M.K. Singh and D.R. Parhi, "Intelligent neuro-controller for navigation of mobile robot," *Proceedings of the International Conference on Advances in Computing, Communication and Control, ICAC3'09*, 2009, pp. 123-128.

[21] D.R. Parhi and M.K. Singh, "Real-time navigational control of mobile robots using an artificial neural network," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 223, 2009, pp. 1713-1725.

[22] S. Haykin, *Neural networks: a comprehensive foundation*, Prentice hall, 1999.

[23] J. McClelland, "Explorations in Parallel Distributed Processing-IBM version," 1988.

[24] D.E. Rumelhart, *Learning internal representations by error propagation*, 1985.

[25] D. Rumelhart and G. Hintont, "Learning representations by back-propagating errors," *Nature*, 1986, pp. 323:533-536.

[26] D.E. Rumelhart and J.L. McClelland, “Parallel distributed processing,” vol. 1, 1986, pp. 318-362.

[27] K.V. Price, R.M. Storn, and J.A. Lampinen, *Differential evolution: a practical approach to global optimization*, Springer Verlag, 2005.

[28] H. Wold, “Partial least squares,” 1985.

[29] K.Q. Weinberger and L.K. Saul, “Unsupervised learning of image manifolds by semidefinite programming,” *International Journal of Computer Vision*, vol. 70, 2006, pp. 77-90.

[30] H. Lu, K.N. Plataniotis, and A.N. Venetsanopoulos, “A survey of multilinear subspace learning for tensor data,” *Pattern Recognition 44*, 2011, pp. 1540-1551.

[31] J.A. Lee and M. Verleysen, *Nonlinear dimensionality reduction*, Springer Verlag, 2007.

[32] A. HyvArinen, J. Karhunen, and E. Oja, *Independent component analysis*, Wiley-interscience, 2001.

[33] S.T. Dumais, “Latent semantic analysis,” *Annual Review of Information Science and Technology*, vol. 38, 2004, pp. 188-230.

[34] I.T. Joliffe, *Principal component analysis*, Springer-Verlag New York, 1986.

[35] J.E. Jackson and J. Wiley, *A user’s guide to principal components*, Wiley Online Library, 1991.

[36] M.E. Tipping and C.M. Bishop, “Probabilistic principal component analysis,” *Journal of the Royal Statistical Society. Series B, Statistical Methodology*, 1999, pp. 611-622.

[37] K. Pearson, “LIII. On lines and planes of closest fit to systems of points in space,” *Philosophical Magazine Series 6*, vol. 2, 1901, pp. 559–572.

[38] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern classification*, wiley New York:, 2001.

[39] P.J.A. Shaw, “Multivariate statistics for the environmental sciences,” A Hodder Arnold Publication, 2003, pp. 92-116.

[40] I. Jolliffe, “Principal component analysis,” Springer-Verlag New York, 2002, pp. 29-62.

[41] A. Weingessel and K. Hornik, “Svd algorithms: Apex-like versus subspace methods,” *Neural Processing Letters*, vol. 5, 1997, pp. 177-184.

[42] F. Palmieri, J. Zhu, and C. Chang, “Anti-Hebbian learning in topologically constrained linear networks: A tutorial,” *Neural Networks, IEEE Transactions on*, vol. 4, 1993, pp. 748-761.

[43] K. Hornik and C.M. Kuan, “Convergence analysis of local feature extraction algorithms,” *Neural Networks*, vol. 5, 1992, pp. 229-240.

[44] P.F. Baldi and K. Hornik, “Learning in linear neural networks: A survey,” *Neural Networks, IEEE Transactions on*, vol. 6, 1995, pp. 837-858.

[45] J. Karhunen, “Optimization criteria and nonlinear PCA neural networks,” *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, IEEE, 1994, pp. 1241–1246.

[46] L. Xu, “Theories for unsupervised learning: PCA and its nonlinear extensions,” *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, 1994, p. 1252a--1253.

[47] E. Oja, “Principal components, minor components, and linear neural networks,” *Neural Networks*, vol. 5, 1992, pp. 927-935.

[48] K.I. Diamantaras and S.Y. Kung, *Principal component neural networks: theory and applications*, John Wiley & Sons, Inc., 1996.

[49] T.D. Sanger, “Optimal unsupervised learning in a single-layer linear feedforward neural network,” *Neural networks*, vol. 2, 1989, pp. 459-473.

[50] L.N. Trefethen and D. Bau, *Numerical linear algebra*, Society for Industrial Mathematics, 1997.

[51] J. Rubner and P. Tavan, “A self-organizing network for principal-component analysis,” *EPL (Europhysics Letters)*, vol. 10, 1989, pp. 693-698.

[52] M. Turk and A. Pentland, “Eigenfaces for recognition,” *Journal of cognitive neuroscience*, vol. 3, 1991, pp. 71-86.

[53] D.D. Lee and H.S. Seung, “Algorithms for non-negative matrix factorization,” *Advances in neural information processing systems*, vol. 13, 2001.

[54] R.A. Fisher, “The use of multiple measurements in taxonomic problems. Annual Eugenics 7 (part II): 179-188. Reprinted in Contributions to Mathematical statistics, 1950,” 1936.

[55] C.J.C. Burges, “A tutorial on support vector machines for pattern recognition,” *Data mining and knowledge discovery*, vol. 2, 1998, pp. 1-47.

[56] N. Christianini and S.J. Taylor, “An introduction to support vector machines (and othre kernel-based learning methods),” 2000.

[57] V. Vapnik, *Statistical learning theory*, Wiley-Interscience, 1998.

[58] E. Osuna, R. Freund, and F. Girosi, “Support vector machines: Training and applications,” 1997.

[59] C.M. Bishop, *Neural networks for pattern recognition*, Oxford university press, 1995.

[60] “SVM Application List” Available: http://www.clopinet.com/isabelle/Projects/SVM/applist.html.

[61] D. Kortenkamp, R.P. Bonasso, and R.R. Murphy, *Artificial intelligence and mobile robots: case studies of successful robot systems*, AAAI Press/MIT Press, 1998.

[62] G.S. Sukhatme and M.J. Mataric, “Robots: Intelligence, Versatility, Adaptivity-Introduction,” *Communications of the ACM-Association for Computing Machinery-CACM*, vol. 45, 2002, pp. 30–32.

[63] C. Stachniss, D. Hahnel, and W. Burgard, “Exploration with active loop-closing for FastSLAM,” *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, 2004, pp. 1505-1510.

[64] S.B. Thrun, “Exploration and model building in mobile robot domains,” *Neural Networks, 1993., IEEE International Conference on*, 1993, pp. 175-180.

[65] M. Meng and A.C. Kak, “Fast vision-guided mobile robot navigation using neural networks,” *Systems, Man and Cybernetics, 1992., IEEE International Conference on*, 1992, pp. 111-116.

[66] R.P. Lippmann, “An introduction to computing with neural nets,” *ARIEL*, vol. 209, 1987, pp. 115-245.

[67] H. Hu and D. Gu, “Landmark-based navigation of mobile robots in manufacturing,” *Emerging Technologies and Factory Automation, 1999. Proceedings. ETFA’99. 1999 7th IEEE International Conference on*, 1999, pp. 121-128.

[68] N. Vlassis, Y. Motomura, and B. Kröse, “Supervised dimension reduction of intrinsically low-dimensional data,” *Neural Computation*, vol. 14, 2002, pp. 191–215.

[69] J.L. Crowley, F. Wallner, and B. Schiele, “Position estimation using principal components of range data,” *Robotics and Autonomous Systems*, vol. 23, 1998, pp. 267–276.

[70] S.K. Nayar, H. Murase, and S.A. Nene, “Learning, positioning, and tracking visual appearance,” *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, IEEE, 1994, pp. 3237–3244.

[71] M. Artac, M. Jogan, and A. Leonardis, “Mobile robot localization using an incremental eigenspace model,” *Robotics and Automation, 2002. Proceedings. ICRA’02. IEEE International Conference on*, IEEE, 2002, pp. 1025–1030.

[72] T. Fujii, Y. Arai, H. Asama, and I. Endo, “Multilayered reinforcement learning for complicated collision avoidance problems,” *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, IEEE, 1998, pp. 2186–2191.

[73] K. Ishii, S. Nishida, K. Watanabe, and T. Ura, “A collision avoidance system based on self-organizing map and its application to an underwater vehicle,” *7th International Conference on Control, Automation, Robotics and Vision, 2002. ICARCV 2002.*, 2002, pp. 602-607.

[74] S. Grossberg and D.S. Levine, “Neural dynamics of attentionally modulated Pavlovian conditioning: blocking, interstimulus interval, and secondary reinforcement,” *Applied Optics*, vol. 26, 1987, pp. 5015-5030.

[75] N.J. Nilsson, *Principles of artificial intelligence*, Springer Verlag, 1982.

[76] C. Kozakiewicz and M. Ejiri, “Neural network approach to path planning for two dimensional robot motion,” *Intelligent Robots and Systems’ 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, 1991, pp. 818-823.

[77] J. Sfeir, H. Kanaan, and M. Saad, “A neural network based path generation technique for mobile robots,” *Mechatronics, 2004. ICM’04. Proceedings of the IEEE International Conference on*, IEEE, 2004, pp. 176–181.

[78] P.S. Sastry, G. Santharam, and K.P. Unnikrishnan, “Memory neuron networks for identification and control of dynamical systems,” *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 5, Jan. 1994, pp. 306-319.

[79] P.K. Pal and A. Kar, “Mobile robot navigation using a neural net,” *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, IEEE, 1995, pp. 1503–1508.

[80] R. Fierro and F.L. Lewis, “Control of a nonholonomic mobile robot using neural networks,” *Neural Networks, IEEE Transactions on*, vol. 9, 1998, pp. 589-600.

[81] R. Fierro and F.L. Lewis, “Practical point stabilization of a nonholonomic mobile robot using neural networks,” *Decision and Control, 1996., Proceedings of the 35th IEEE*, 1996, pp. 1722-1727.

[82] R. Fierro and F.L. Lewis, “Control of a non-holonomic mobile robot using neural networks,” *Intelligent Control, 1995., Proceedings of the 1995 IEEE International Symposium on*, pp. 415-421.

[83] S.X. Yang and M. Meng, “An efficient neural network approach to dynamic robot motion planning,” *Neural networks : the official journal of the International Neural Network Society*, vol. 13, Mar. 2000, pp. 143-148.

[84] S.X. Yang and M. Meng, “An efficient neural network method for real-time motion planning with safety consideration,” *Robotics and Autonomous Systems*, vol. 32, Aug. 2000, pp. 115–128.

[85] S.X. Yang and M.H. Meng, “Real-time collision-free motion planning of a mobile robot using a Neural Dynamics-based approach.,” *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 14, Jan. 2003, pp. 1541-1552.

[86] A.L. Hodgkin and A.F. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” *The Journal of physiology*, vol. 117, 1952, pp. 500-544.

[87] S. Grossberg, “Nonlinear neural networks: Principles, mechanisms, and architectures,” *Neural Networks*, vol. 1, 1988, pp. 17-61.

[88] S. Carpin, M. Lewis, and J. Wang, “USARSim: a robot simulator for research and education,” *Robotics and*, 2007, pp. 1400-1405.

[89] S. Balakirsky and F. Proctor, “USARSim.”

[90] A. Jacoff, E. Messina, and J. Evans, “Reference test courses for autonomous mobile robots,” *Proceedings of SPIE*, 2001, p. 341.

[91] Microsoft, “Microsoft Robotics” Available: http://msdn.microsoft.com/en-us/library/bb483042.aspx.

[92] ActivMedia, “ADEPT mobile robots” Available: http://www.activrobots.com.

[93] iRobot, “Robots that Make a Difference” Available: http://www.irobot.com/.

[94] A. Kleiner, V.A. Ziparo, D. Meyer-Delius, B. Steder, M. Ruhnke, R. Kümmerle, C. Dornhege, and B. Nebel, "RescueRobots Freiburg" Available: http://gkiweb.informatik.uni-freiburg.de/~rescue/robots/index.php.

[95] J. More, "The Levenberg-Marquardt algorithm: implementation and theory," *Numerical analysis*, 1978, pp. 105-116.

[96] M.T. Hagan, H.B. Demuth, M.H. Beale, and others, *Neural network design*, PWS Boston, MA, 1996.

[97] D. Saad, *On-line learning in neural networks*, Cambridge Univ Pr, 1998.

[98] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press New York, 1998.

[99] R.S. Sutton and A.G. Barto, *Reinforcement learning: An introduction*, Cambridge Univ Press, 1998.

VITA AUCTORIS

NAME	Seyyed Hamid Dezfoulian
PLACE OF BIRTH	Hamedan, Iran
YEAR OF BIRTH	1985
EDUCATION	Computer Engineering Department Payame-Noor University Hamedan, Iran 2004 – 2009 B.Eng.
	School of Computer Science University of Windsor Windsor, Ontario, Canada 2009 – 2011 M.Sc. (Co-op)