Intelligence Based Error Detection and Classification for 3D Measurement Systems

By

Iván Jan-Richard van Rooyen

Submitted in fulfilment of the requirements for the degree of Master of Engineering in Mechatronics to be awarded at the Nelson Mandela Metropolitan University.

April 2017

Supervisor: Prof Theo Ian van Niekerk

Declaration

I, Iván Jan-Richard van Rooyen, hereby declare that the work presented in this dissertation is my own and that all sources used and referred to have been documented and recognised.

Furthermore, this dissertation has not previously been submitted in full or in partial fulfilment of the requirements of another qualification.

Author's Signature

Date

Abstract

For many years 2D machine vision has been used to perform automated inspection and measuring in the manufacturing environment. A strong drive to automate manufacturing has meant improvements in robotics and sensor technologies. So has machine vision seen a steady movement away from 2D and towards 3D. It is necessary to research and develop software that can use these new 3D sensing equipment in novel and useful ways. One task that is particularly useful, for a variety of situations is object recognition.

It was hypothesised that it should be possible to train artificial neural networks to recognise 3D objects. For this purpose a 3D laser scanner was developed. This scanner and its software was developed and tested first in a virtual environment and what was learned there was then used to implemented an actual scanner. This scanner served the purpose of verifying what was done in the virtual environment. Neural networks of different sized were trained to establish whether they are a feasible classifier for the task of object recognition.

Testing showed that, with the correct preprocessing, it is possible to perform 3D object recognition on simple geometric shapes by means of artificial neural networks.

Acknowledgements

Pa, Ma: Baie dankie vir julle liefde, ondersteuning en oneindige geduld.

Roxanne en Renic: Dankie vir al die moed in praat; julle vertroue in my was ’n reuse aansporing.

Prof Van Niekerk: Prof se ondersteuning, positiwiteit en geduld sal my altyd by bly. Baie, baie dankie.

Vir:
R.R.C. van Rooyen
J.F. Westraadt

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

Nomenclature

$\boldsymbol{A}$ Camera matrix containing the intrinsic camera parameters; f_x, f_y, c_x, c_y

$\boldsymbol{M'}$ Homogeneous coordinate in the object coordinate system.

$\boldsymbol{m'}$ Homogeneous pixel coordinate,

$$\boldsymbol{m'} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$\boldsymbol{R_x}(\psi)$ Rotation transformation about the x axis.

$$\boldsymbol{R_x}(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix}$$

$\boldsymbol{R_y}(\phi)$ Rotation transformation about the y axis.

$$\boldsymbol{R_y}(\phi) = \begin{bmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix}$$

$\boldsymbol{R_z}(\theta)$ Rotation transformation about the z axis.

$$\boldsymbol{R_z}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\boldsymbol{R}$	Rotation matrix;

$$\boldsymbol{R} = \boldsymbol{R_z}(\theta)\boldsymbol{R_y}(\phi)\boldsymbol{R_x}(\psi) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$\boldsymbol{t}$	Translation vector;

$$\boldsymbol{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

μ	The arithmetic mean.
ϕ	Angle of rotation about the y-axis of a coordinate system.
ψ	Angle of rotation about the x-axis of a coordinate system.
σ	Standard deviation.
τ	Threshold light intensity value for performing a binary threshold operation on a monochrome image.
θ	Angle of rotation about the z-axis of a coordinate system, or the bias value of an artificial neuron, depending on the context.
ACC	Accuracy.
c_x, c_y	Centroid of image sensor.
f_x, f_y	Focal length along x and y axes. Measured in pixels.
FN	False negative. When a value expected to be true is observed to be false.
FP	False positive. When a value expected to be false observed to be true.
$k_1, \ldots, k_3$	Radial distortion coefficients.
NPV	Negative Predictive Value.
p_1, p_2	Tangential distortion coefficients.
PPV	Positive Predictive Value (Precision).

s	Arbitrary scalar value.
SPC	Specificity (True Negative Rate).
t_x, t_y, t_z	x, y, and z components of the translation vector.
TN	True negative. When a value expected to be false is observed to be false.
TP	True positive. When a value expected to be true is observed to be true.
TPR	True Positive Rate (Sensitivity).
u, v	Pixel coordinates

Part I

Background

Chapter 1

Introduction

> *"Any customer can have a car painted any colour that he wants so long as it is black."* - Henry Ford, circa 1909 [1].

At the time when Henry Ford made this statement, owning a motor vehicle was still very much a novelty and the privilege of only those wealthy enough to afford such a luxury. Ford understood that the only way to reach a broader market, his company would have to design, produce and sell cars that were more affordable. To achieve this goal Ford set out to design a *"motor car for the great multitude"* based on the principle of simplicity. He believed:

> *"The less complex an article, the easier it is to make, the cheaper it may be sold, and therefore the greater number may be sold."*

This quest for simplicity culminated in the production of the Model T and the moving assembly line which gave Ford a competitive advantage, resulting in rapid growth for the company in the early 1900s.

These days, however, customers are no longer satisfied with products that are *"any colour you like as long as it is black"*. Customers want products that reflect their individuality. In [2], the author states that:

> *manufacturers in the automotive sector are experiencing an increased customer demand for* ***personalised****, high quality* ***products****.*

To obtain and maintain a competitive advantage it has become important for manufacturers to adapt to customer demands. Manufacturers that are able to adapt the most efficiently have the competitive advantage. The traditional Dedicated Manufacturing Systems (DMSs), like that pioneered by Ford, are too rigid to respond efficiently to rapid market changes brought on by customer demands. Manufacturers and researchers have developed, and are continually researching and improving, adaptive manufacturing techniques. These adaptive manufacturing systems, also known as agile manufacturing systems, can be divided into two categories; Flexible Manufacturing Systems (FMSs) and Reconfigurable Manufacturing Systems (RMSs). While FMSs are more flexible than RMSs and thus capable of producing a wider range of products or parts, FMSs are more suited to low production volumes. RMSs are better suited for larger production runs. However, both these manufacturing systems are designed to facilitate the production of a variety of different parts or part families.

Revisiting the statement by [2]:

> *manufacturers in the automotive sector are experiencing an increased customer demand for personalised,* ***high quality products****.*

– it is clear that customers also value products of a good quality. To ensure that products are of a high quality manufacturers inspect the products or, in some cases like [2] the manufacturing equipment itself, for anomalies. [3] reports that current inspection technologies used most frequently to provide a form of flexible quality control include, Coordinate Measuring Machines (CMMs) and even manual inspection [4]. These methods, however, are slow, expensive and can cause bottlenecks [3], [5]. For this reason a number of researchers are developing Automated Inspection Systems (AISs), and specifically flexible and reconfigurable inspection systems based on machine vision. These inspection systems or optical measurement systems can perform non-contact measurements and are considerably faster than traditional CMMs.

The majority of research conducted on the topic of automated inspection for agile manufacturing systems focusses on the physical reconfigurability of the inspection system. [5] claims that optical metrology techniques are still lacking intelligence. The research presented in this dissertation is an attempt to make optical measurement systems more flexible by introducing heuristic object recognition.

1.1 Problem Statement

The introductory paragraphs of this chapter introduced the following two concepts:

- Agile manufacturing; flexible- and reconfigurable manufacturing
- Automated inspection.

It is where these two concepts intersect where the inspiration for this research originated. To illustrate the problem to be addressed in this research consider the following scenario:

An FMS consisting of a material transport system, four robotic arms for material handling, four Computerised Numerical Control (CNC) machines, and a sorting station; produces parts A, B, C, and D. Since flexibility is the defining characteristic of FMSs, suppose that all four CNC machines are able to produce any of the four parts, and that the CNC machines can be reassigned to produce a part according to the demand for said part. Now, consider the issue of quality; it is required that the parts be measured and compared to Computer Aided Design (CAD) designs to ensure that each part is within allowable tolerances. It is also a requirement that the quality inspection be done as quickly as possible, ruling out the use of CMMs in favour of optical measurement techniques.

Following, are two solutions to the scenario proposed above:

Solution 1

This solution, illustrated in figure 1.1, consists of introducing automated, optical measurement/inspection stations at each CNC machine in the FMS. Each CNC machine is supplied material from the conveyor system by means of a dexterous robotic arm. Once a CNC machine has completed a part, the robot arm takes the part and positions it in the AIS. The AIS inspects the part which is then transferred onto the conveyor system by the robot arm.

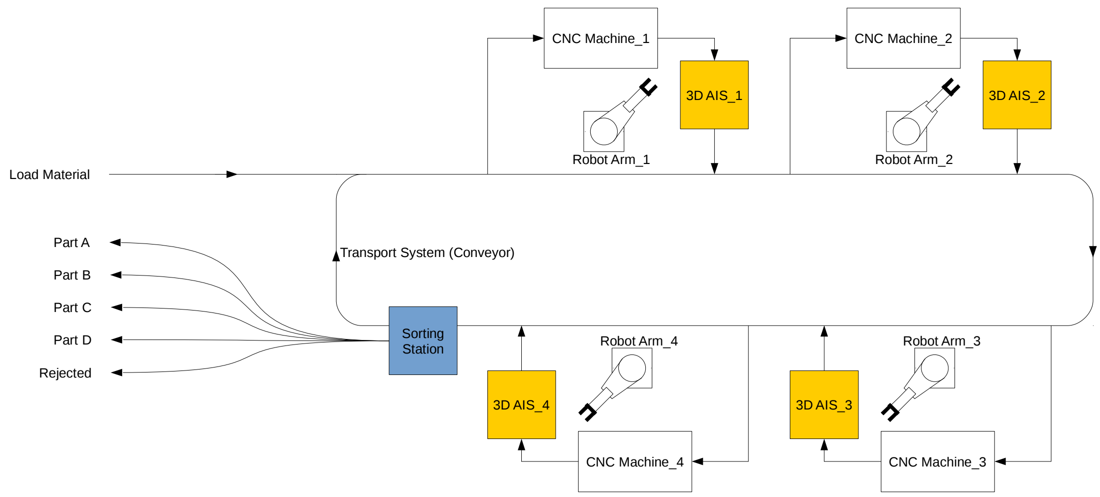


Figure 1.1: Schematic layout of a simple FMS with AIS at each machine station.

However, there are a few problems with this solution:

- **Complex scheduling/communication -** Each AIS must communicate with the sorting station what part is on its way, and whether that part must be rejected or not. The sorting station must keep a schedule of these communications to operate correctly. The parts produced by the FMS each have their corresponding lead time, adding complexity to the communication and scheduling of the system.
- **High cost -** Although the cost of non-contact measuring systems like laser and camera based scanners are becoming less expensive as the technology matures, they remain a significant expense. Having duplicate AISs adds to the cost.

The next solution attempts to address these problems.

Solution 2

This second solution, illustrated in figure 1.2, attempts to improve upon the first solution by replacing the AISs with a single ***intelligent*** AIS. Figure 1.3 shows a flow chart of how such an intelligent AIS might function. The problem of high cost is addressed by the reduction of inspection units, while the communication complexity is reduced by making the AIS more intelligent; providing it with the ability to *see for itself* – to recognise – what part it is inspecting. And it is this idea – the ability to recognise a part – that lead to the main research question asked in the next section.

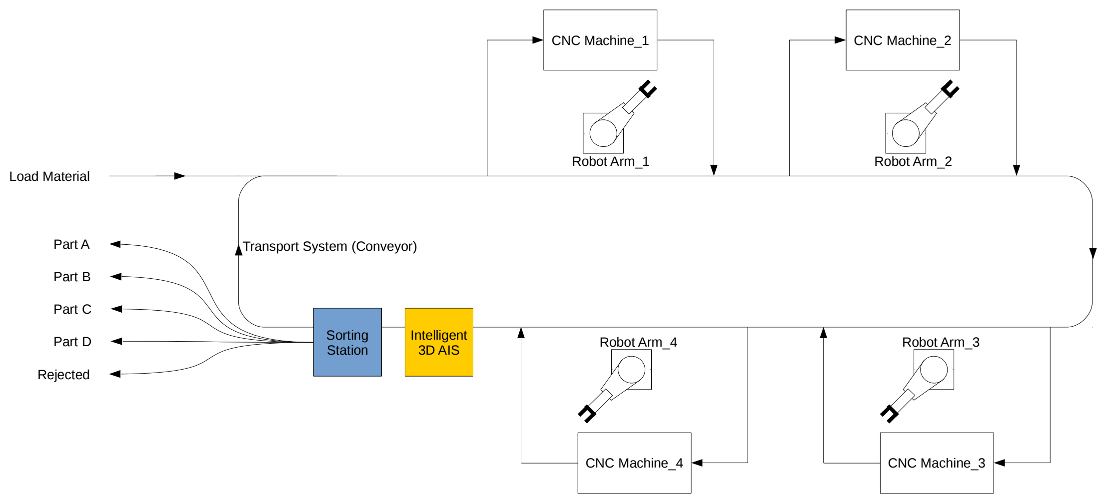


Figure 1.2: Alternative schematic layout of a simple FMS with a single, intelligent AIS.

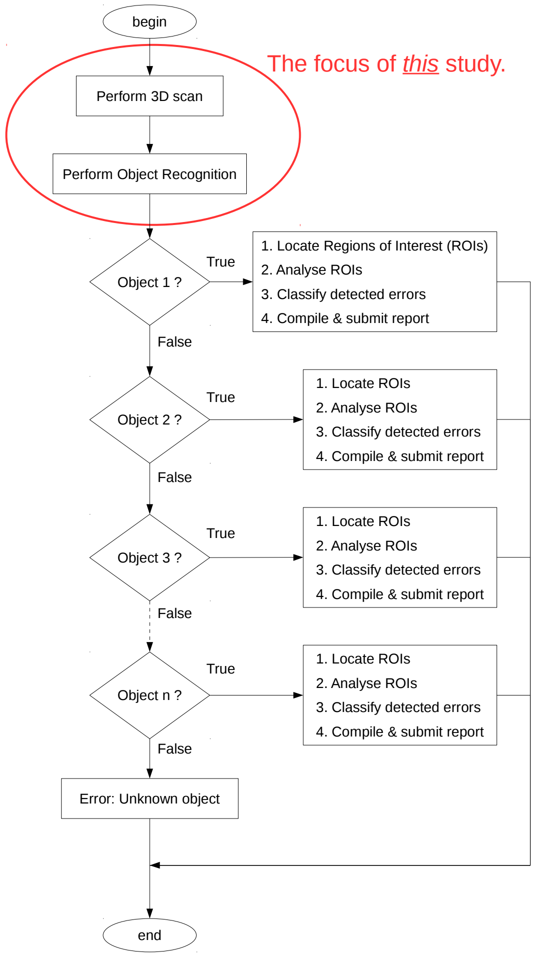


Figure 1.3: High level flow chart for intelligent AIS.

1.1.1 Research Question and Hypothesis

Research Question

Can a machine be made to recognise an object?

This question serves as the point of departure for the research described in this dissertation. However, in this form the question is too general to serve as the main research question. The question must be refined and focussed by answering the following:

- How does the machine perceive (sense/see) the object? What hardware provides the data of the environment - the input - to the machine? (A camera? Or some type of Three-dimensional (3D) sensor?)

- The answer to the previous question then leads to the following question: How is the input data represented and what does it represent? (2D data or 3D?)
- Once the data representing the object have been acquired; how will the recognition/classification be done? What techniques are available, and which are the prevailing ones?

The scenario described in the *Problem Statement* section required an automated inspection system to ensure that part dimensions are within specified tolerances. This implies measurements are made in three dimensions and consequently a 3D capable sensor like a 3D scanner is required. The preliminary research question can now be rewritten as:

> Is it possible to recognise an object from data obtained by a 3d scanner?

Surface data obtained by 3D scanners are typically presented in either a some mesh format (*.ply, *.stl, or similar) or as a point cloud (*.pcd in the case of Point Cloud Library (PCL)). Point clouds are the simplest 3D representation of an object and will be used in this research. The research question now becomes:

> Is it possible to recognise an object from point cloud data?

Popular techniques or classifiers used for 2D object recognition include; Support Vector Machines (SVMs), Kernel estimation (k-nearest neighbour) and ANNs. The research presented here wants to establish whether the use of ANNs are appropriate in the case of 3D object recognition, thus leading to the final iteration of the research question:

> **Is it possible to recognise/classify an 3D object from point cloud data using an ANN?**

Hypothesis

Rewriting the final version of the research question into a statement reveals the hypothesis:

> **It is possible, using ANNs, to classify, or recognise, 3D objects represented by point cloud data.**

1.1.2 Objectives

To achieve the ultimate goal of the answering the research question above, the following objectives were set out:

1. Obtain insight into 3D surface data capturing- and Object Recognition methods.
2. Implement a 3D scanner to demonstrate an understanding of the underlying principles.
3. Develop an ANN or ANNs to perform Object Recognition. This includes researching ways to reduce point cloud data to an input vector of manageable length for the proposed ANN(s).

1.2 Research Methodology

The research methodology can be divided into the following sections:

Literature Study In this part of the research the work of other researchers are considered. Knowledge of machine vision, laser triangulation, object recognition and artificial neural networks is gathered.

Implementation Here the gathered knowledge is used to implement a 3D scanner and to develop neural network based object recognition software.

Testing and Validation Finally, what results from the implementation stage is tested and validated.

1.3 Delimitations of the Research

In order to prevent inflating the scope of the project it was important to establish the scope, the domain, of the research presented in this dissertation. The following delimiting factors were identified:

- From the literature study it became evident that many 3D sensing technologies exist. The focus of the work undertaken in this particular project, however, was recognising 3D objects and not to conduct an in-depth study on sensing technologies. Yet a 3D acquisition device remained an integral part in performing this research. For this reason this study was limited to the development of a 3D scanning device based on the laser triangulation principle.
- The study was also limited to implementing only ANNs to perform object recognition or object classification. Other types of classifiers will be discussed in the next chapter, but these classifiers were not implemented.

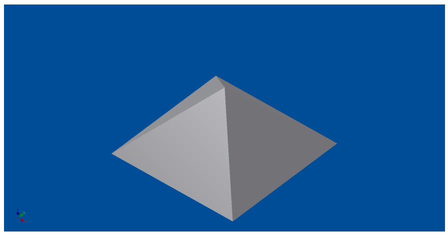
- ANNs can be implemented on a variety of hardware such microprocessors, Field Programmable Gate Arrays (FPGAs), and Programmable Logic Controllers (PLCs). In this study the ANN(s) was implemented on a personal computer using C++ and existing software libraries as far as practical to reduce development complexity.

- The objects to be recognised in this project were limited to the eight non-complex shapes shown in figure 1.4. These *non-complex* geometric objects were chosen since the goal of this study was not to find the upper limit of object complexity at which ANNs will fail to perform the recognising task, but rather to establish whether ANNs were a feasible approach. Note the similarity between *Object 6*, *Object 7* and *Object 8*. This was to subject the developed ANN(s) to some level of ambiguous data.

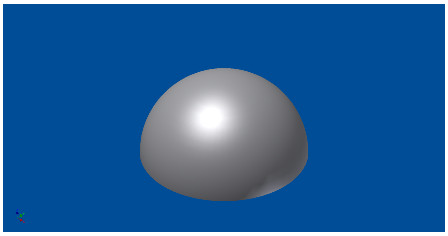
(a) Object 1: Cube

(b) Object 2: Cylinder

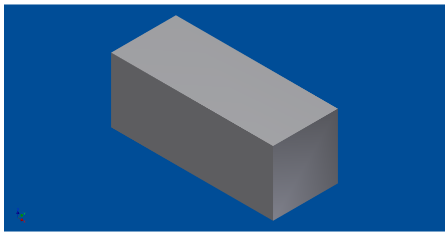
(c) Object 3: Triangular prism



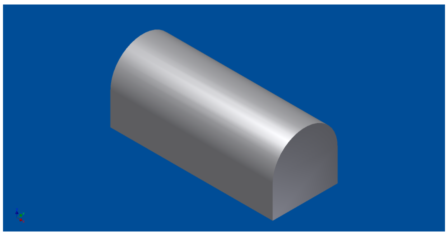
(d) Object 4: Square base pyramid



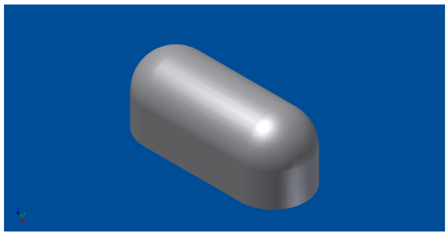
(e) Object 5: Hemisphere



(f) Object 6: Rectangular prism



(g) Object 7: Modified object 6



(h) Object 8: Modified object 7

Figure 1.4: The 8 objects used for recognition in this research.

1.4 Research Significance

1.4.1 Significance to Industry

The automotive industry is a major contributor to the Nelson Mandela Bay economy. Sustained research and innovation is necessary to ensure that the local industry remains globally competitive. The research done in this project offers two contributions to the local manufacturing industry:

1. A method or approach to recognise objects or parts from 3D scan data.
2. A low cost method to design and analyse machine vision systems and test related vision algorithms before procuring any vision hardware.

1.4.2 Personal Significance

This research undertaking presented an opportunity to develop and increase programming proficiency, especially in using the C++ language. At the same time a deeper understanding of machine vision and machine learning was obtained by studying the related literature and implementing what was learned there using OpenCV, PCL, Fast Artificial Neural Network (FANN) library, and C++.

1.4.3 Significance to the University

Over the years, and in more recent times, several research projects conducted at Nelson Mandela Metropolitan University (NMMU) contained elements of machine perception and machine learning. Research has been done on the use of a 3D scanning device to measure and detect tooling errors within an automotive production context [2]. Machine vision (stereo vision in particular) guided tracking of seams for robotic welders has also been studied in [6]. In [7] the author trained ANNs by means of a Genetic Algorithm (GA) to perform parts recognition using timed distance signals.

The research conducted in this project, and presented in these pages, supplements NMMU's existing body of knowledge pertaining to machine vision and machine intelligence by combining these paradigms to perform 3D object/shape recognition.

1.5 Organisation of the Dissertation

For clarity and readability this dissertation is divided into four parts:

- ***Part I***, the background, consists of three chapters. *Chapter 1 - Introduction* introduces the the problems of 3D data acquisition and object recognition. In *Chapter 2 - Machine Vision and Laser Triangulation* relevant machine vision literature is discussed, the laser triangulation principle is presented while *Chapter 3 - Object Recognition and Artificial Neural Networks* contains a literature review of research relting to object recognition, and the necessary theory on ANNs that will be used to perform the task object recognition.

- ***Part II*** contains two chapters focusing implementation of the theories presented in, and knowledge gained from, the previous three chapters. A 3D laser scanner is developed *Chapter 4 - 3D Laser Scanner Implementation*. This scanner is first implemented as a simulation using a software package called Blender, then an actual scanner is built to verify the simulated scanner. *Chapter 5 - Object Recognition Implementation* discusses the development and implementation of an ANN to perform object recognition.

- ***Part II*** consists of one chapter, *Chapter 6 - Results*, and details the experiments conducted in this study and presents the results thereof.

- Finally, ***Part IV***, consisting of *Chapter 7 - Discussion and Conclusion*, concludes the dissertation. This final chapter discusses the results shown in the chapter 7, presents some conclusions that can be made based on those results, and also suggests ways in which this study could be improved upon and future work that could stem from the research presented in this dissertation.

A visual representation of the structure of the dissertation can be seen in figure 1.5.

1.6 Summary

In conclusion, the main research problem, **3D object recognition**, was introduced in this chapter and the main research question was formulated:

> ***Is it possible to recognise/classify an 3D object from point cloud data using an ANN?***

Furthermore, this chapter outlined the objectives, methodology, scope and significance of the research, as well as the layout of this dissertation.

The next chapter explores the literature related to machine vision, in particular laser triangulation based scanners and related theory.

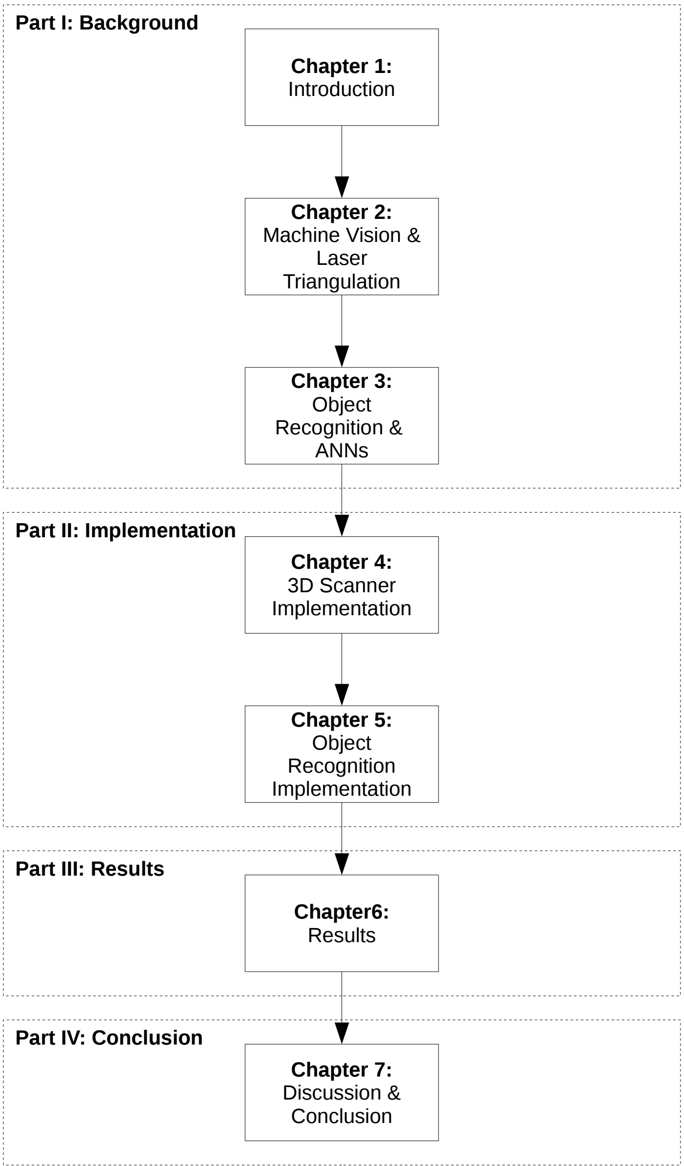


Figure 1.5: Layout of the dissertation.

Chapter 2

Machine Vision and Laser Triangulation

The aim of this chapter is to present a review of machine vision literature in section 2.1, with the remaining sections dedicated to machine vision theory relevant to the 3D scanner developed for this research project.

2.1 Machine Vision: A Review of the Literature

The authors of this paper, [8], proposed a system to inspect mass produced items – in this case, percussion caps. The vision component of the inspection system comprises of a camera and laser based 3D scanner. The paper does not discuss the vision system in great detail and is more focussed on the task of classifying the inspected pieces. This classification is done using an artificial neural network, warranting further mention of this paper in the next chapter.

To perform measurements for inspection- and quality purposes, often CMMs are used. These machines are slow and it is not always practical to obtain a dense set of measurements representative of the entire surface of the measured object – optical methods like laser triangulation scanners are much more suited for this scenario. The authors of [9] developed a dual camera 3D laser scanner and compared the performance of this scanner to that of a CMM. They found that the scanner they developed produces measurements with similar accuracy to those obtained with a CMM.

The paper, [10], documents the calibration of a 3D inspection system meant to inspect the completeness of assemblies on an assembly line. The inspection system is also based upon the principle of triangulation. To combat the issue of occlusion, this vision consists of two lasers and a camera, instead of the more conventional set-up with dual cameras and a single laser. This paper focusses on the calibration of said inspection system.

[11] is a paper on measuring the wheel alignment of a motor vehicle using laser triangulation. This paper proved useful because of the completeness of the mathematics presented therein.

The authors of [12] discuss improving the speed of a robot's ability to perform pick-and-place tasks by adding 3D machine vision. They claim that performing object recognition on 3D data is slow and therefore use only 2D information to classify the objects. 3D data is used to determine object pose aiding the robot in the pick-and-place procedure. 3D data is acquired using a laser triangulation type scanner.

In [13], the authors present 3D vision system to inspect the condition of the carbon contact strips on pantographs for electrical railway vehicles. These contact strips wear over time, and must be maintained to perform optimally. The vision system used in by the authors is based on laser triangulation. A simple laser line thinning approach was used: the centre of the laser line is assumed to be along the brightest part of the projected line.

Laser triangulation based machine vision systems are quite popular for performing seam tracking in automated welding tasks and has been the research topic of many researchers such as [14]–[16].

In [14], the authors present an automated welding system to perform welds on golf club heads. The solution relies on a 3D vision system to find the weld path. The vision system used is a camera-and-laser, laser triangulation based solution, augmented by using two cameras. This reduces the probability of occlusions occurring in the scans. The authors report that the precision of the vision system is acceptable for the welding task, with a 3D error of approximately 0.0314mm with a standard deviation of 0.0211mm.

2.2 The Camera Model

This section shows how a camera is modelled mathematically. Consider figure 2.1. This figure represents a pinhole camera capturing a scene containing point P situated at the coordinate (X, Y, Z). A ray of light is reflected off of P. This ray of light is then focused by the optics onto the image sensor to form an image of P at point (u, v) in

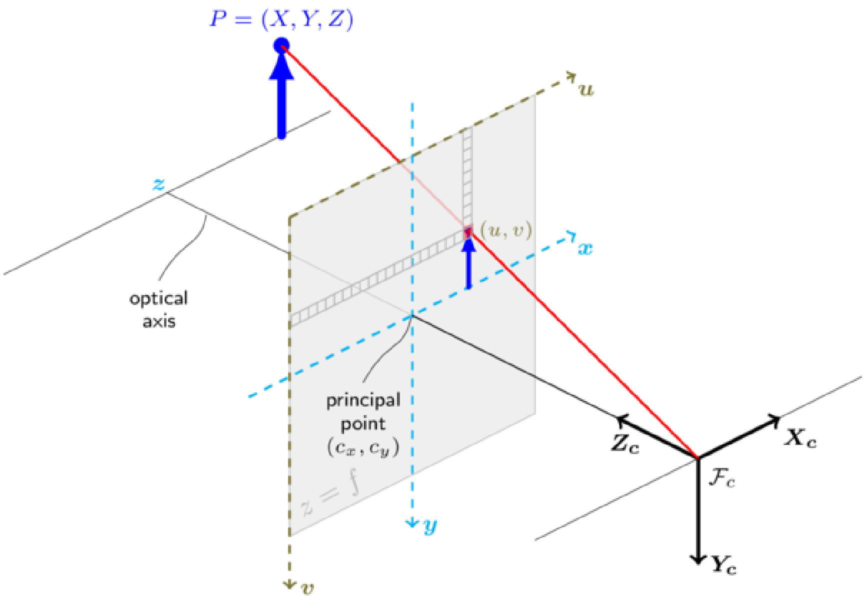


Figure 2.1: The pinhole camera model [17].

the image plane. Equation 2.1 describes this mathematically.

$$s\boldsymbol{m'} = \boldsymbol{A}\,[\boldsymbol{R}|\boldsymbol{t}]\,\boldsymbol{M'} \tag{2.1}$$

which, with the matrices expanded to expose the matrix elements, can be rewritten as:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.2}$$

Machine vision and photogrammetry literature refer to $\boldsymbol{A}$ as the *camera intrinsics matrix*, or simply, the *camera matrix*. Equations 2.1 and 2.2, shown using homogeneous coordinates, can be written as equations 2.3 and 2.4 respectively.

$$s\boldsymbol{m'} = \boldsymbol{A}\,(\boldsymbol{R}\boldsymbol{M} + \boldsymbol{t}) \tag{2.3}$$

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \right) \tag{2.4}$$

Provided that $z \neq 0$, the above equations are equivalent to:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \boldsymbol{R} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \boldsymbol{t} \tag{2.5}$$

$$x' = \frac{x}{z} \qquad\qquad y' = \frac{y}{z} \tag{2.6}$$

The coordinates, (u, v), of the projected point can then be calculated as:

$$u = f_x x' + c_x \qquad\qquad v = f_y y' + c_y \tag{2.7}$$

That then concludes the basic mathematical model for an ideal pinhole camera. The next section will look at:

- different types of distortion that may occur when capturing images, and
- how to incorporate distortion into the camera model presented above.

2.3 Distortion

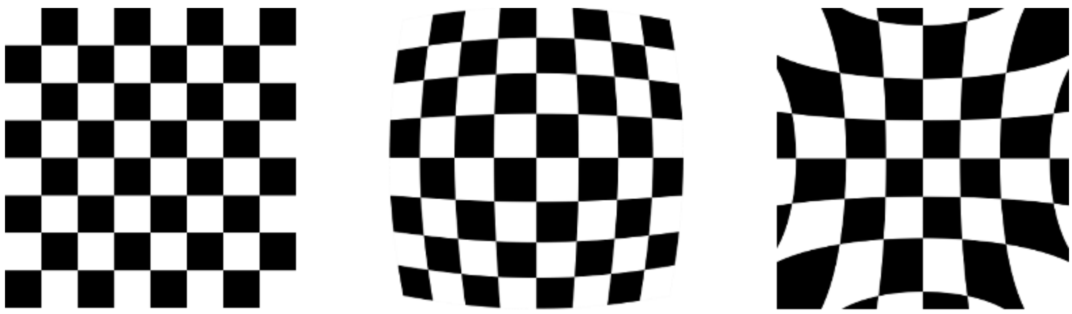
The previous section introduced the mathematical model for an ideal camera. This is a good starting point, but real world cameras are often far from ideal and may capture distorted images. The two most common types of distortion are:

1. radial, and
2. tangential distortion.

These two types of distortions are discussed below.

2.3.1 Radial Distortion

The ideal lens has a parabolic profile. However, real lenses – notably, inexpensive lenses – do not have true parabolic profiles. This deviation causes radial distortion, illustrated in figure 2.2. [18] states that radial distortion at the optical centre of the



(a) No distortion. (b) Positive radial distortion. (c) Negative radial distortion.

Figure 2.2: Radial lens distortion

image sensor is equal to 0, and the distortion increases as r – the distance from the optical centre – increases. Radial distortion can be modelled using a Taylor expansion about $r = 0$, shown in equations 2.8 and 2.9.

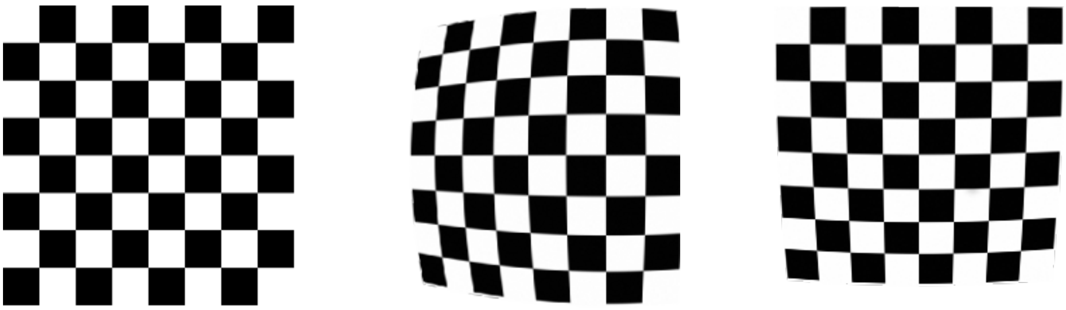
$$x'' = x'(1 + k_1r^2 + k_2r^4 + k_3r^6) \tag{2.8}$$

$$y'' = y'(1 + k_1r^2 + k_2r^4 + k_3r^6) \tag{2.9}$$

where $r^2 = x'^2 + y'^2$.

2.3.2 Tangential Distortion

Tangential distortion is caused by manufacturing defects, specifically when the camera lens and image sensor are not exactly parallel. This type of distortion can be modelled by equations 2.10 and 2.11. Tangential distortion is depicted in figure 2.3.



(a) No distortion. (b) Tangential distortion (horizontal). (c) Tangential distortion (vertical).

Figure 2.3: Tangential distortion

$$x'' = x' + 2p_1x'y' + p_2(r^2 + 2x'^2) \tag{2.10}$$

$$y'' = y' + p_1(r^2 + 2y'^2) + 2p_2x'y' \tag{2.11}$$

Combining radial and tangential distortions:

$$x'' = x'(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2p_1x'y' + p_2(r^2 + 2x'^2) \tag{2.12}$$

$$y'' = y'(1 + k_1r^2 + k_2r^4 + k_3r^6) + p_1(r^2 + 2y'^2) + 2p_2x'y' \tag{2.13}$$

Using equations 2.12 and 2.13, the distorted pixel coordinates, $(u_{distorted}, v_{distorted})$, are then approximated with:

$$u_{distorted} = f_x x'' + c_x \qquad v_{distorted} = f_y y'' + c_y \tag{2.14}$$

2.4 Camera Calibration

This sections discusses the issue of camera calibration. Camera calibration is the process of determining a camera's intrinsic parameters, as well as the distortion coefficients of the lens being used. Knowing these parameters is important for performing accurate optical metrology type tasks. For example, by knowing the distortion coefficients of the lens, it possible to, at least partly, compensate for and eliminate distortion.

OpenCV was used in this research to implement all image acquisition and processing. For this reason the OpenCV literature, [17], [18], was consulted on the topic of camera calibration.

According to [18], the OpenCV camera calibration procedure estimates the camera intrinsics using an algorithm based on the research by [19]. The distortion coefficients are determined using an algorithm based on Brown's method presented in [20]. The calibration is performed by capturing images of a planar calibration pattern, usually a checker board pattern consisting of squares with known dimensions. These images are taken, each time the calibration pattern being held in a different pose (see figure 2.4). Given enough images, the OpenCV calibration algorithm then estimates the camera intrinsics, distortion coefficients and pose ($[\boldsymbol{R}|\boldsymbol{t}]$) for each image. This done by minimising the re-projection error using iterative Levenberg-Marquardt optimisation[17].

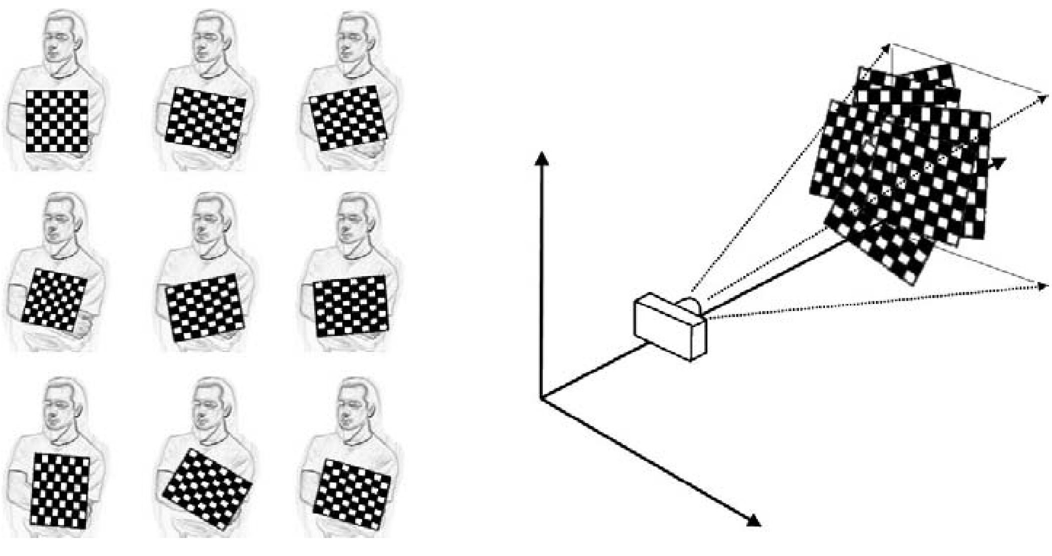


Figure 2.4: Camera calibration procedure[18].

2.5 Image Processing

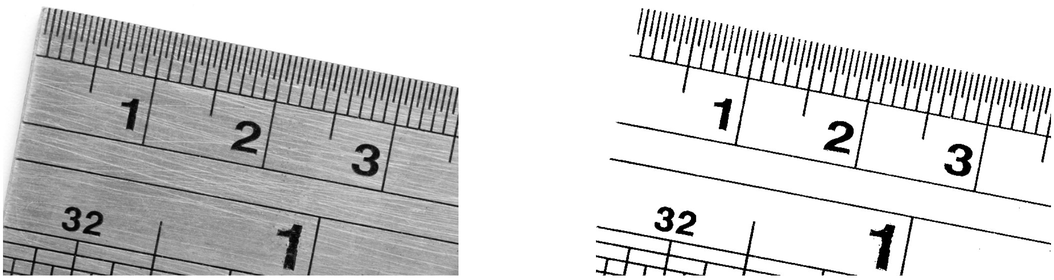
In machine vision applications, the images captured often contain more data or information than is required for the task at hand. Analysing this extra information may add to the complexity and operation time of the vision system. For this reason it is usually necessary to perform some kind of image processing, or preprocessing, to reduce the amount of data to only that what is useful. This is especially true for vision systems that need to operate at near real-time speeds.

This section introduces the two most popular techniques used in image preprocessing:

1. Segmentation (also called thresholding), and
2. Smoothing (sometimes referred to as blurring).

2.5.1 Segmentation

When performing basic segmentation on a monochrome image, the pixels with an intensity value lower than a specified threshold value is forced to 0. All pixels with intensities above the threshold are set to the maximum intensity (255 in the case of an image with 8 bit colour encoding). Figure 2.5 shows an example of this.

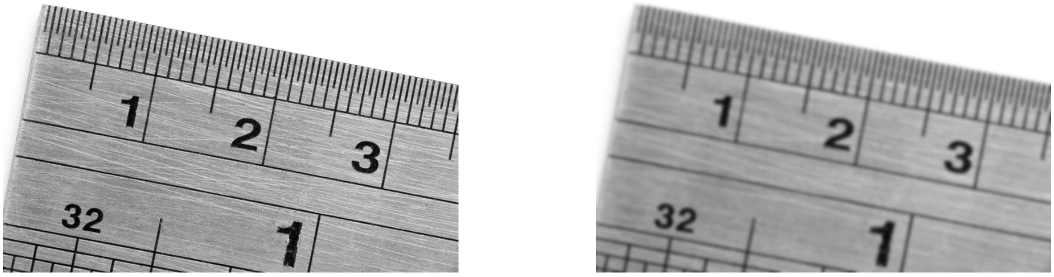


(a) Original image. (b) Thresholded image.

Figure 2.5: Example of segmentation (Threshold value set to 100).

2.5.2 Smoothing

The smoothing operation is mostly used to reduce noise, or fine details, that might be present in an image. Gaussian smoothing is one of the most well known smoothing operators. An example of Gaussian blur is shown in figure 2.6. In this example a 9×9 Gaussian kernel was used.



(a) Original image. (b) Blurred image.

Figure 2.6: Example of Gaussian blur (9 × 9 kernel).

2.6 Laser Triangulation

This section explains the principle of laser triangulation. In sub-section 2.6.2 the equations needed to implement a laser scanner are derived from the camera model.

2.6.1 The Basic Principle

A 3D scanner that uses laser triangulation typically consists of at least one camera and one laser. The laser is usually of the type that projects a sheet of light. The scanner is constructed in such a way that the camera is at an angle to the laser (see figure 2.7). When scanning an object, the object is passed through the sheet of laser (alternatively, the camera-laser assembly may be passed over the object). The laser light is reflected off of the object, producing a distorted line highlighting the profile of the object. These distorted lines are captured by camera. The 3D shape information of the object can then be obtained by processing the captured images. Equations 2.15 through 2.18, in conjunction with figure 2.7, shows how to calculate the height, z, of an object using basic geometry and trigonometry.

$$\alpha = \arctan\left(\frac{b}{h}\right) \qquad \beta = \arctan\left(\frac{u}{f}\right) \tag{2.15}$$

$$\gamma = \pi - (\alpha + \beta) \tag{2.16}$$

$$d = \sqrt{b^2 + h^2} \tag{2.17}$$

$$z = \frac{(d - f)\sin\beta}{\sin\gamma} \tag{2.18}$$

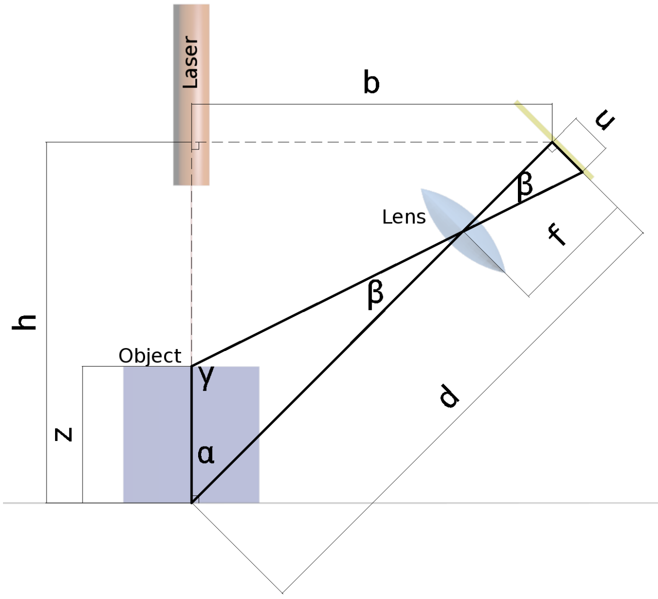


Figure 2.7: The laser triangulation principle.

The width of the object may be calculated following similar logic. The following subsection presents a more general approach to calculate the 3D data for laser scanning.

2.6.2 3D Reconstruction Using Inverse Camera Model

This approach is essentially the inverse of the camera model discussed earlier in this chapter, and was inspired by the work done in [21]. The camera model (equation 2.1) allows us to estimate the image point or pixel values of known object point or XYZ-coordinate, while 3D laser scanning aims to achieve the opposite or inverse: 3D coordinates are estimated from pixel values. If the projected plane of laser light is coincident with the YZ-plane of the object coordinate system, then $X = 0$, and equation 2.2 can be rewritten as:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{12} & r_{13} & t_x \\ r_{22} & r_{23} & t_y \\ r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} Y \\ Z \\ 1 \end{bmatrix} \tag{2.19}$$

Now, let:

$$\boldsymbol{H} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{12} & r_{13} & t_x \\ r_{22} & r_{23} & t_y \\ r_{32} & r_{33} & t_z \end{bmatrix} \tag{2.20}$$

By substitution equation 2.19 becomes:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} Y \\ Z \\ 1 \end{bmatrix} \tag{2.21}$$

This equation can be rearranged to give:

$$\begin{bmatrix} Y \\ Z \\ 1 \end{bmatrix} = s\boldsymbol{H}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \qquad \det(\boldsymbol{H}) \neq 0 \tag{2.22}$$

Let:

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \boldsymbol{H}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2.23}$$

Then equation 2.22 can be written as:

$$\begin{bmatrix} Y \\ Z \\ 1 \end{bmatrix} = s \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{2.24}$$

Finally:

$$X = 0 \qquad Y = sq_1 \qquad Z = sq_2 \tag{2.25}$$

where $s = \frac{1}{q_3}$.

That concludes this section on laser triangulation.

2.7 Summary

This chapter opened with a literature review on machine vision in partial fulfilment of the first objective of this study. This literature review highlighted the following:

- Laser triangulation remains an acceptable and popular method for acquiring 3D data, both in research and industry.
- The digital camera is an integral and complex component of laser triangulation based scanner.

- It is imperative to perform camera calibration to improve the accuracy of machine vision systems. This improvement is the result of adjusting for any misalignment and lens distortion that might be present in the vision system.

- To extract useful information from captured images it is necessary to perform image processing.

The chapter presented the mathematical model of a pinhole camera (refer to figure 2.1 and equations 2.1 to 2.7).

Next, the chapter discussed the main types and sources of distortions namely, radial and tangential distortion. Figure 2.2 and 2.3 illustrate the effect of radial and tangential distortion respectively.

Then followed a section on a camera calibration method based on the works of [19] and [20].

Some basic image processing concepts were introduced namely, segmentation and smoothing.

Finally, a section on laser triangulation presented:

- the laser triangulation principle using basic geometry and trigonometry (refer to figure 2.7 and equations 2.15 to 2.18).

- a more general case of laser triangulation derived from the camera model presented early in this chapter (see equations 2.19 to 2.25).

The following chapter looks at object recognition and then discusses artificial neural networks in some detail.

Chapter 3

Object Recognition and Artificial Neural Networks

3.1 3D Object Recognition: A Review of the Literature

This section presents a review of literature pertaining to object recognition.

Although not quite object recognition, [8] developed an intelligent system to inspect the quality of percussion caps. The system uses a 3D laser scanner to acquire the 3D data of a plate containing 120 percussion caps. Each percussion cap, or region of interest, is then processed resulting in a number of parameters that serve as the input vector for a back-propagation neural network. The neural network is trained to classify the percussion as one of the following classes:

1. *Dented central cap.*
2. *Poorly mounted central cap.*
3. *Inverted central cap.*
4. *Paper present in the joints of the cap.*
5. *Missing central cap.*
6. *Dirty central cap.*
7. *Dirty or dented external cap.*
8. *Missing percussion cap.*
9. *Central cap mounted above tolerance.*
10. *Central cap mounted above tolerance.*

The author reports that the neural network is able to correctly classify the percussion caps 93.05% of the time.

The authors of [22] proposes a deep learning model for recognising 3D objects and pose estimation from 2D images. They then use this model to implement a robotic system able to grasp objects. They apply the Max-pooling Convolutional Neural Network to perform both object recognition and pose estimation. The authors report that

> *our system can make accurate object recognition, pose estimation, as well as successful grasping.*

In [23], the authors discuss a 3D object recognition system using what they call an interactive open-ended learning approach. The system is able to learn new objects over time. They acquire 3D data of a scene using a RGB-D sensor, specifically Microsoft's Kinect. This sensor captures colour and depth information of a scene. The 3D data is then segmented in order to isolate the objects contained in the scene. For each object a *spin-image* – a type of descriptor – is then calculated. Using this spin-image, the objects are classified by means of a nearest-neighbour algorithm.

[24] proposes using Deep Belief Networks (DBN) to perform object recognition and pose estimation. DBNs are a type of deep learning model. The system developed by these authors uses two cameras to obtain a top and side view of the object to be recognised. A DBN is created for each camera and the outputs of these two DBNs are fed into a final output layer that performs the classification. This approach has the advantage of not relying on a 3D sensor of some kind, although this reduces the system's capability to perform measurement tasks.

In [25], the authors present a method to recognise 3D objects. The system is developed to improve factory automation, and is implemented on a robot arm with 7 degrees of freedom. The authors use a similar approach as [23]. Instead of performing a segmentation step and calculating spin-images for the detected object, [25] proposes using global descriptors discussed in [26]–[29]. Again, a nearest-neighbour algorithm is used to perform the classification.

[30] proposes a 3D object recognition method where a 3D scene is basically projected onto multiple 2D views, essentially images. These images are then searched for known objects.

In the paper, [31], the authors perform 3D object recognition by using a Convolutional Neural Network to detect features in RGB-D data. These detected features are then fed to a Support Vector Machine (SVM) that performs the task of classification.

[32] implemented a 3D object recognition system for mobile robots. 3D data is obtained using a RGB-D sensor. The 3D data is then processed to obtain features using descriptors developed by [26], [33]–[36]. Finally the 3D features obtained from the

descriptors are passed to a nearest-neighbour algorithm to perform classification.

This concludes the review of the object recognition literature. The next section introduces and discusses the artificial neural network theory relevant for purposes of object recognition in this research.

3.2 ANNs

ANNs are algorithms that attempt to solve a variety of problems by mimicking the way the biological neural systems, like the human brain, work. According to [37], ANNs are particularly useful for solving problems such as:

- classification
- pattern completion
- optimisation
- control
- function approximation
- data mining

It is exactly because of this ability to solve classification problems that ANNs were selected for this research.

The remainder of this chapter is dedicated to ANNs and the theory relevant to this research. Note that unless otherwise indicated, all theory presented further in this chapter is from [37] and all symbols and naming conventions follow those used by the author of that work.

3.2.1 The Artificial Neuron

As the name, ANN, suggests, an ANN is a network of interconnected Artificial Neurons (ANs). These are the building blocks of an ANN. An AN actually represents some function that maps some input $\mathbb{R}^I$ to $[0, 1]$ or $[-1, 1]$ depending on the type of activation function used. This is shown in 3.1 and 3.2.

$$f_{AN} : \mathbb{R}^I \mapsto [0, 1] \tag{3.1}$$

$$f_{AN} : \mathbb{R}^I \mapsto [-1, 1] \tag{3.2}$$

where I is the number of input values. These input values are typically presented to the AN in vector format e.g.:

$$\boldsymbol{z} = (z_1, z_2, \ldots, z_I) \tag{3.3}$$

$$\boldsymbol{v} = (v_1, v_2, \ldots, v_I) \tag{3.4}$$

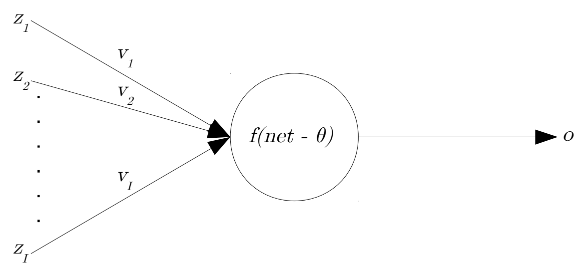


Figure 3.1: An Artificial Neuron, shown here with input vector $\boldsymbol{z}$, weight vector $\boldsymbol{v}$, and output signal o.

The AN also has a corresponding weight, v_i, for each input value, z_i. These weights strengthen or weaken the input signals in an attempt to emphasise the more important signals and to attenuate the less important input values. From the input signals and corresponding weights the total-, or net input signal is calculated. An activation function, f_{AN}, then calculates the output value, o. A threshold or bias, denoted by θ is applied to the net input signal to control the strength of the output signal. Figure 3.1 is a graphical representation of this process.

In the following three subsections the basic components of an artificial neural network shall be presented. These components are:

1. the net input signal,
2. the activation function, and
3. the threshold value.

Net Input

The most common way of calculating the net input signal is by computing the *weighted sum* of all the input signals. This is done using equation 3.5

$$net = \sum_{i=1}^{I} z_i v_i \tag{3.5}$$

An AN using equation 3.5 is called a summation unit (SU). Another method of calculating the net input signal is shown in equation 3.6. An AN implementing this method is called a product unit (PU).

$$net = \prod_{i=1}^{I} z_i^{v_i} \tag{3.6}$$

PUs are known to provide improved information capacity.

Activation Functions

This subsection documents the activation functions most commonly used. The purpose of the activation function is to determine an output value from the given inputs, weights and bias. This process is often referred to as *firing*.

Linear function The linear activation function is represented by the following equation, the graph of which is shown figure 3.2a:

$$f(net - \theta) = \lambda(net - \theta) \tag{3.7}$$

where λ is the gradient of the function.

Step function The step activation function is given as:

$$f(net - \theta) = \begin{cases} \gamma_1 & \text{if } net \geq \theta \\ \gamma_2 & \text{if } net < \theta \end{cases} \tag{3.8}$$

where γ_1 and γ_2 usually are 1 and 0 respectively. $\gamma_1 = 1$ and $\gamma_2 = -1$ are also not uncommon. Figure 3.2b shows illustrates this activation function.

Ramp function (See figure 3.2c)

$$f(net - \theta) = \begin{cases} \gamma & \text{if } net - \theta \geq \epsilon \\ net - \theta & \text{if } -\epsilon < net - \theta < \epsilon \\ -\gamma & \text{if } net - \theta \leq -\epsilon \end{cases} \tag{3.9}$$

Sigmoid function This is the most commonly used activation function. It can be written as:

$$f(net - \theta) = \frac{1}{1 + e^{-\lambda(net - \theta)}} \tag{3.10}$$

where λ controls the gradient of the function. (See figure 3.2d)

Hyperbolic tangent function The hyperbolic tangent activation function, given in equation 3.11, is yet another activation often used:

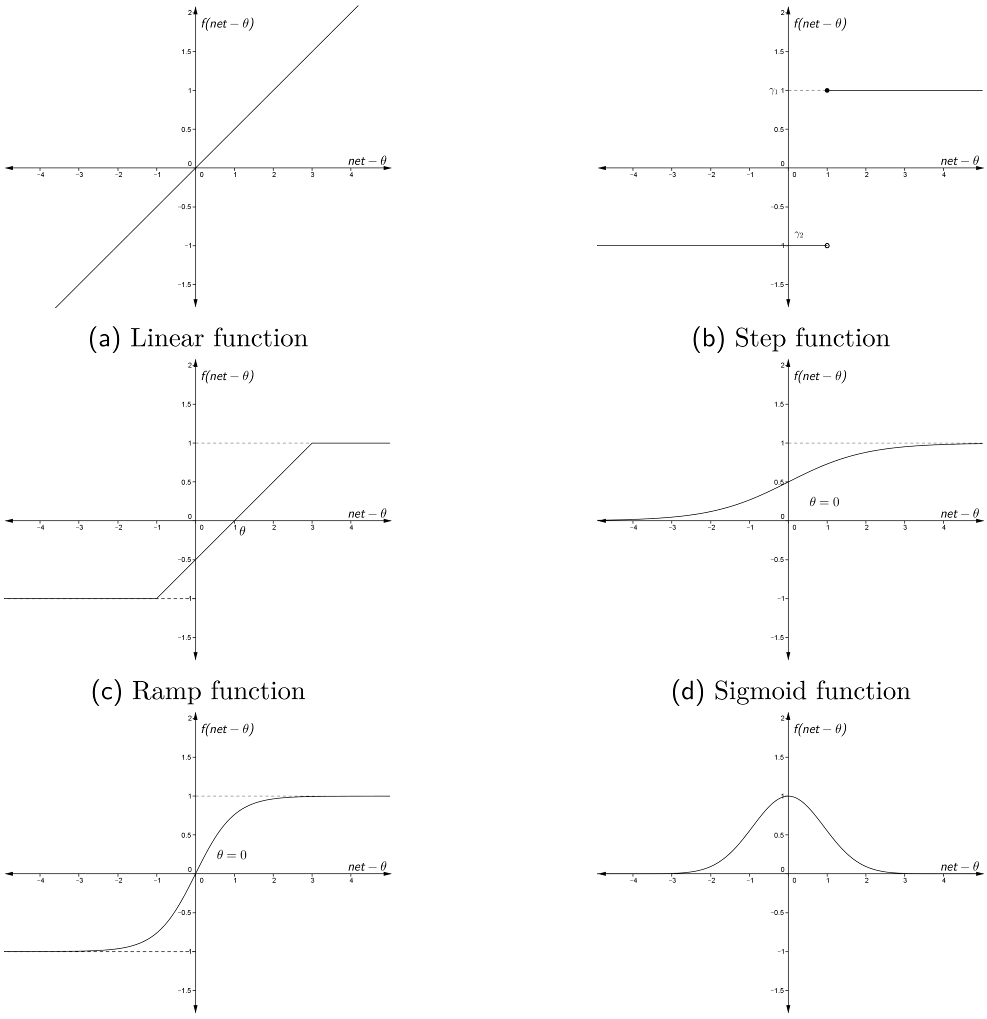
$$f(net-\theta) = \frac{e^{\lambda(net-\theta)} - e^{-\lambda(net-\theta)}}{e^{\lambda(net-\theta)} + e^{-\lambda(net-\theta)}} \tag{3.11}$$

where, again, λ dictates the steepness of the function. As seen in figure 3.2e, this function is similar to the sigmoid activation function but, the output ranges from -1 to 1 instead of from 0 to 1.

Gaussian function Shown in figure 3.2f, this activation function is given as:

$$f(net-\theta) = e^{-\frac{(net-\theta)^2}{\sigma^2}} \tag{3.12}$$

where the mean of the Gaussian function is $net - \theta$, and the standard deviation is σ.



(a) Linear function

(b) Step function

(c) Ramp function

(d) Sigmoid function

(e) Hyperbolic tangent function

(f) Gaussian function

Figure 3.2: Popular activation functions.

That concludes this subsection on activation functions. The next subsection discusses the threshold value, θ, and the purpose thereof.

The Threshold

The threshold, also referred to as the bias, is denoted by the Greek symbol θ. The purpose of this value is to amplify or attenuate the output signal of an artificial neuron.

To simplify the learning algorithms (discussed later), the bias is incorporated into the *net* input signal using:

$$\begin{aligned} net &= \sum_{i=1}^{I} z_i v_i - \theta \\ &= \sum_{i=1}^{I} z_i v_i + z_{I+1} v_{I+1} \\ &= \sum_{i=1}^{I+1} z_i v_i \end{aligned} \tag{3.13}$$

where $\theta = z_{I+1} v_{I+1} = -v_{I+1}$.

3.2.2 The Artificial Neural Network

Now that the basic building block of the artificial neural network – the artificial neuron – has been discussed, this section will discuss in deeper detail the neural network.

Learning

Consider the feed-forward neural network shown in figure 3.3. For an artificial neural network to be of any use it is necessary to find right combination of connection weights such that the network produces the expected output for a given input. The process of finding these weights is known as *learning*. Neural networks typically learn by using one of the following methods:

- **Supervised learning:** The neural network is presented with training dataset. This dataset contains input patterns and corresponding desired outputs. During training the network estimates outputs for each input vector. These outputs are compared to the desired values and the connection weights are adjusted to minimise the error between the actual- and desired outputs.

- **Unsupervised learning:** Here the network is left to find to patterns in the input data on its own. There is no error minimisation step as in supervised learning.

- **Reinforcement learning:** Similar to supervised learning, here the network is rewarded (positive reinforcement) for performing well, and penalised (negative reinforcement) for not performing well.

A supervised learning strategy was used in this research, and shall therefore be discussed in further detail in the rest of this chapter. Supervised learning algorithms can be divided into the following two groups:

- **Stochastic learning:** With this learning approach the ANN's weights are adjusted after each training pattern is presented to the network. The following training pattern is randomly selected from the training set to prevent the network from developing any bias caused by the order of the training patterns.

- **Batch learning:** Here the weight changes are added up, and only once the network has been presented with the complete training set are the weights adjusted.

Batch back-propagation learning was used to train the neural networks implemented in chapter 5. The underlying mathematics for this particular learning method is presented next.

Batch Back-propagation Learning for Feed-forward Neural Networks Again consider the network shown in figure 3.3. Using the sigmoid activation function the following equations can be used to train the neural network under consideration. This training is also summarised in algorithm 3.1. Each output value in the output layer is

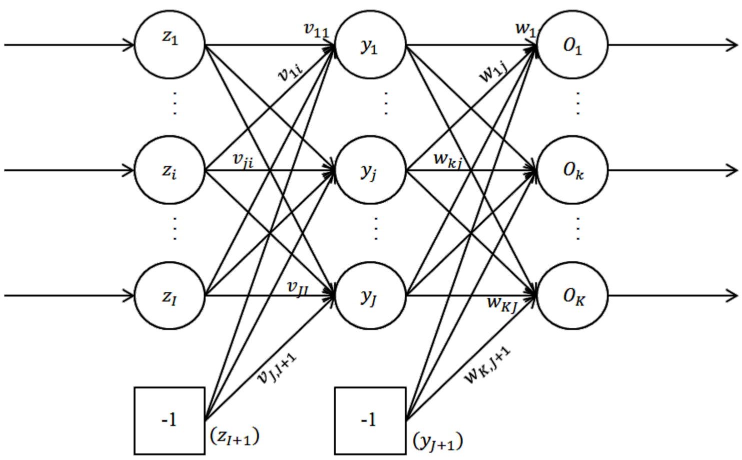


Figure 3.3: A general case FFNN showing the connection weights between layers.

calculated using:

$$o_k = f_{o_k}(net_{o_k}) = \frac{1}{1 + e^{-net_{o_k}}} \tag{3.14}$$

and the output of each hidden neuron is calculated with:

$$y_j = f_{y_j}(net_{y_j}) = \frac{1}{1 + e^{-net_{y_j}}} \tag{3.15}$$

Connection weight adjustments are made using:

$$w_{kj}(t) += \Delta w_{kj}(t) + \alpha \Delta w_{kj}(t-1) \tag{3.16}$$

$$v_{ji}(t) += \Delta v_{ji}(t) + \alpha \Delta v_{ji}(t-1) \tag{3.17}$$

where α is the momentum. Momentum attempts to counteract the occurrence of the network becoming trapped in a local minimum during training.

The weight changes for the input connections of the output layer calculated using the following two equations. These weight changes are calculated after each training iteration.

$$\begin{aligned} \Delta w_{kj} &= \eta \left(-\frac{\partial E}{\partial w_{kj}} \right) \\ &= -\eta \delta_{o_k} y_j \end{aligned} \tag{3.18}$$

where $E = \frac{1}{2} \sum_{k=1}^{K} (t_k - o_k)^2$, η is the learning rate. The learning rate determines how large an adjustment is made to weights during training. δ_{o_k} is the output error to be back-propagated and this is calculated as:

$$\delta_{o_k} = \frac{\partial E}{\partial net_{o_k}} \tag{3.19}$$

Similarly, the weight changes for the input connections of the hidden layer is calculated with:

$$\begin{aligned} \Delta v_{ji} &= \eta \left(-\frac{\partial E}{\partial v_{ji}} \right) \\ &= -\eta \delta_{y_j} z_i \end{aligned} \tag{3.20}$$

$$\delta_{y_j} = \frac{\partial E}{\partial net_{y_j}} \tag{3.21}$$

In batch training the network weights are adjusted only after all training patterns have been presented to the network. These weight adjustment are done using equations 3.16 and 3.17. The batch weight changes are calculated by accumulating the weight changes for each training pattern, using the equations below:

$$\Delta w_{kj}(t) = \sum_{p=1}^{P_T} \Delta w_{kj,p}(t) \tag{3.22}$$

$$\Delta v_{ji}(t) = \sum_{p=1}^{P_T} \Delta v_{ji,p}(t) \tag{3.23}$$

Algorithm 3.1 Batch/Offline Back-propagation Learning Algorithm

```
Initialise weights, learning rate (η), momentum (α), and the epochs (t = 0);
while stopping conditions not true do
   Let ε_T = 0;
   for each training pattern p do
      Feed-forward pass: calculate y_{j,p} ∀j = 1, ··· , J and o_{k,p} ∀k = 1, ··· , K;
      Compute output error signals, δ_{o_{k,p}}, and hidden layer error signals, δ_{y_{j,p}};
      Δw_{kj}(t) = Δw_{kj}(t − 1) + Δw_{kj,p}(t);
      Δv_{ji}(t) = Δv_{ji}(t − 1) + Δv_{ji,p}(t);
   end for
   Back-propagation of Errors: Adjust weights w_{kj} and v_{ji};
   ε_T = ε_T + Σ_{k=1}^{K} (t_{k,p} − o_{k,p})^2;
   t = t + 1;
end while
```

3.3 Summary

This chapter presented a review of object recognition related research. From this literature review it became apparent that artificial neural networks are actively being used in research to solve various classification problems. This fact justifies the use of artificial neural networks in this research project.

The rest of the chapter was dedicated to artificial neural network theory and presented:

- The Artificial Neuron, which consisted of:
  - an explanation on how the net input of a n artificial neuron is calculated (refer in particular to equations 3.5 and 3.6).
  - a section discussing the most commonly used activation functions (see figure 3.2 together with equations 3.7 to 3.12).

  - a section explaining the purpose of the bias value, θ. This section also showed how the bias is integrated into the net input signal of the artificial neuron (refer to equation 3.13).

- The Artificial Neural Network, which included:
  - the different types of learning methods used to train artificial neural networks. These are:
    - supervised learning,
    - unsupervised learning, and
    - reinforcement learning.
  - an explanation of stochastic and batch learning. These two learning strategies are types of supervised learning.
  - a relatively detailed description of the back propagation training algorithm typically used to train artificial neural networks (refer to figure 3.3, equations 3.14 up to 3.23 and, algorithm 3.1).

The next chapter looks at the implementation of a 3D scanner based on the principle of laser triangulation.

Part II

Implementation

Chapter 4

3D Laser Scanner Implementation

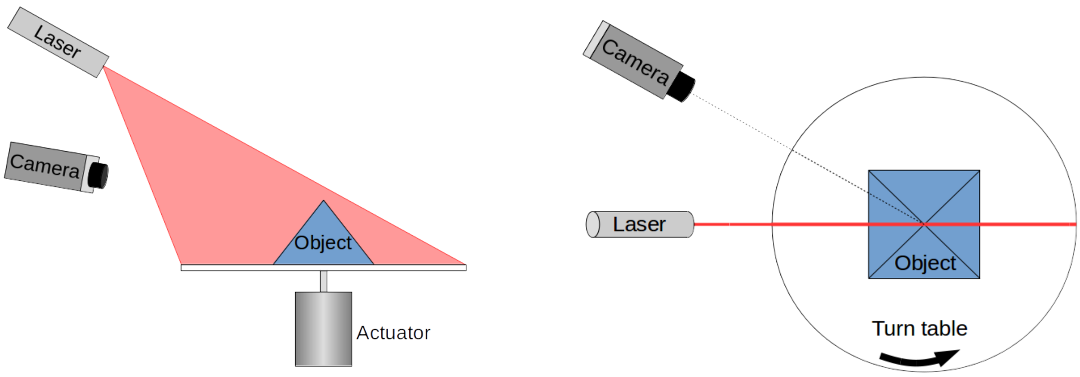
4.1 The Concept

This section will discuss, from a high level point of view, the concept, and the method of operation of the laser scanner that was developed during this research project. Subsequent sections will delve into the deeper details concerning the hardware and of software of the laser scanner.

The scanner operates in the following way: the object to be is placed on a rotating platform and is then rotated in view of a camera. This camera captures frames while a laser projects a plane of light onto the object. The laser highlights the outline of the object. Each captured frame is then analysed – for this research the red channel is analysed and the pixels with highest intensity found. The shape of the object, or the surface thereof, is then calculated from these pixel values using the laser triangulation equations discussed in section 2.6.2. Figure 4.1 shows the proposed configuration of the hardware.

From a Systems Engineering or architectural point of view the laser scanner can be represented by figure 4.2. At the highest level a PC

- captures of images from the camera,
- coordinates the actuation of the rotating platform,
- coordinates the state of the laser's illumination,
- performs the necessary image processing to generate the 3D scan.



(a) Side view. (b) Top view.

Figure 4.1: Concept configuration. This figure demonstrates the proposed orientation of the scanner hardware.

At a level below the PC, a microcontroller is used to control the actuator and laser.

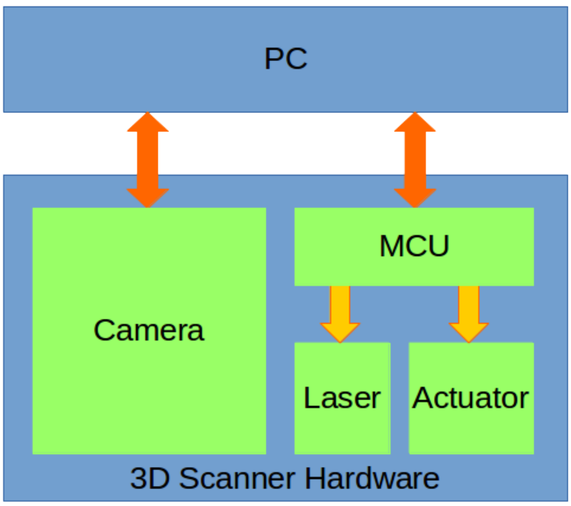


Figure 4.2: Concept Architecture

During this research the need arose to commence with developing the required software before any of the hardware needed for the scanner was available. To overcome this challenge it was decided to simulate a 3D scanner. The idea was to develop and test the image processing software on images obtained from this simulated, or virtual, scanner. At a later stage, once the required hardware was available, the performance of the software and algorithms developed for the simulation would be validated on an actual scanner. The next section discusses the simulated scanner in greater detail.

4.2 The Simulation

Developing a new product often relies on an iterative design process (see figure 4.3). Developing a 3D scanner requires the selection of hardware such as the type of camera, lens, laser and actuators to use. It is also necessary to establish where these hardware elements must be positioned. To follow an iterative design process, which includes the building of a prototype and related software after each iteration, can become an expensive endeavour. By replacing the physical prototype with a simulated one, it becomes possible to develop and test software, and various hardware configurations, before having to procure any expensive components. So, the advantages of prototype simulation are:

- it is financially less expensive than physical prototypes,
- software development and testing can start earlier.

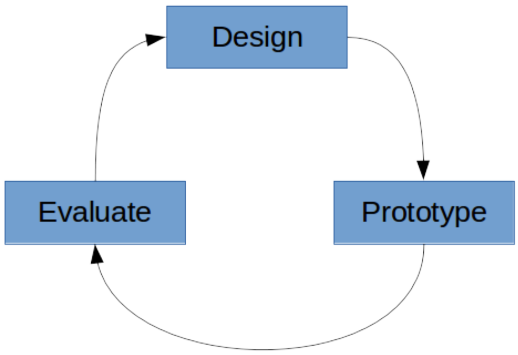


Figure 4.3: The iterative design process

For these two reasons it was decided to simulate a 3D scanner in order to determine:

- the size and geometry of the scanner (camera and laser pose).
- the specification for the camera (resolution, focal length).

Once all of this was done, a physical scanner was build to verify the algorithms developed for the simulation also work in the physical world. This verification platform will be discussed later in the chapter in section 4.3.

4.2.1 3D Model and Animation in Blender

Since, for this research, it became a requirement to develop and test image processing software with the aid of simulation, it was very important that this simulation be as realistic as practically possible. To realise this requirement a solution from the motion picture- and video gaming industries was employed. Video games and special visual effects in film are becoming increasingly difficult to distinguish from reality despite being computer generated. Software often used by animators, graphic designers and architects amongst others include:

- Autodesk 3ds Max,
- Autodesk Maya,
- MAXON Cinema 4D, and
- Blender.

These software packages can be used for 3D modelling, animation and the photo realistic rendering of said models and scenes. The open source product, Blender, was used in this project for the purpose of creating a virtual scanner. The output of this virtual scanner is a sequence of renderings to be processed by the software discussed in section 4.2.2.

Blender

Blender is a suite for creating 3D content. Blender is a product of the Blender Foundation and is free and open source software. Blender can be used to perform the following [38]:

- 3D modelling,
- rigging,
- animation,
- simulation,
- rendering,
- composting,
- motion tracking,
- video editing,
- game creation.

The simulated scanner for this research relied on Blender's modelling, animation and rendering capabilities. Blender is a powerful and feature rich package accompanied by a steep learning curve. A detailed, step-by-step, explanation here of how the virtual scanner was made using Blender would be impractical, and is considered to be beyond the scope of this document. Interested readers are encouraged to consult the multitude of online video tutorials showing how to use Blender. However, the next sub section will describe the strategy behind creating a laser light source in Blender, since such a light source does not exist at the time of writing, using Blender 2.69. Figure 4.4 shows the Blender software with the model simulated scanner.

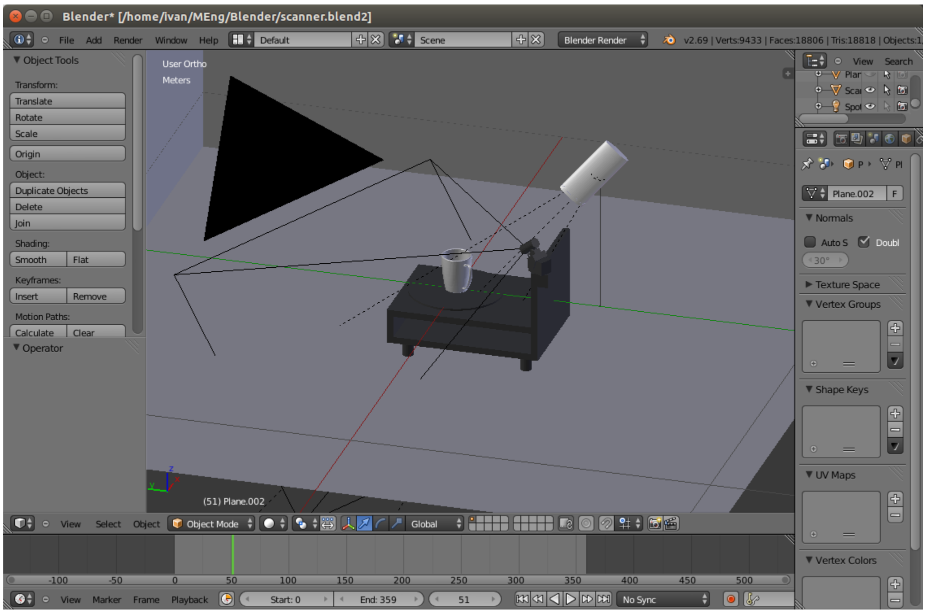


Figure 4.4: This figure shows the 3D model of the scanner modelled in Blender

Scanner Dimensions

The final dimensions for the virtual scanner is shown in figure 4.5 and also summarised in table 4.1. Note it is difficult to indicate clearly rotations about axes in the figure; hence the accompanying table.

Table 4.1: Scanner dimensions.

Dimension	Description	Value	Unit
t_{x_c}	x component of the distance between world- and camera coordinate systems	-115	mm
t_{y_c}	y component of the distance between world- and camera coordinate systems	-230	mm
t_{z_c}	z component of the distance between world- and camera coordinate systems	-200	mm
ψ_c	Rotation of the camera coordinate system about x_c	-128.2	$\circ$
ϕ_c	Rotation of the camera coordinate system about y_c	0	$\circ$
θ_c	Rotation of the camera coordinate system about z_c	-26.6	$\circ$
f_x	dimensionless focal length for x-axis, $\frac{focallength}{pixelwidth}$	600	–
f_x	dimensionless focal length for y-axis, $\frac{focallength}{pixelheight}$	600	–
c_x	half the image sensor width	400	pixels
c_y	half the image sensor height	300	pixels

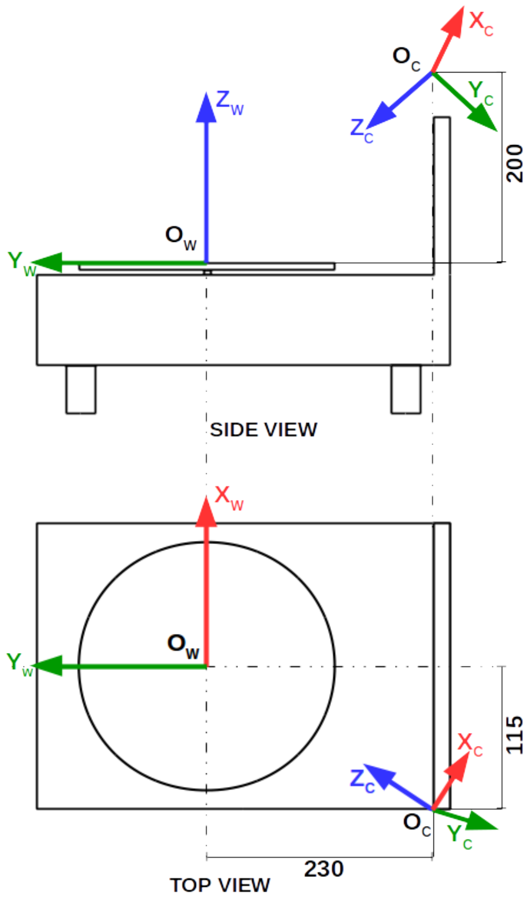


Figure 4.5: Scanner dimensions.

From these dimensions it is possible to find the camera matrix, $\boldsymbol{A}$, the rotation matrix, $\boldsymbol{R}$, and the translation vector, $\boldsymbol{t}$:

$$\boldsymbol{A} = \begin{bmatrix} 600 & 0 & 400 \\ 0 & 600 & 300 \\ 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{R} = \begin{bmatrix} 0.8944 & -0.4473 & 0.0000 \\ -0.2766 & -0.5531 & -0.7859 \\ 0.3515 & 0.7029 & -0.6184 \end{bmatrix} \quad \boldsymbol{t} = \begin{bmatrix} 0.00 \\ 0.00 \\ 325.77 \end{bmatrix} \qquad (4.1)$$

these parameters are required for performing laser triangulation as described in section 2.6.2.

Modelling a Laser

Table 4.2 shows the various types of default light sources available to the user. Each of these light sources are configurable and parameters such as light colour and intensity, amount of specular highlight and diffuse shading can be adjusted. The virtual laser scanner requires a laser light source. Unfortunately Blender does not provide a laser light source as a default option at this point in time. For that reason a laser had to be created with what tools are available in Blender.

Table 4.2: Blender light sources

Name	Description
Point	Omnidirectional point light source
Sun	Constant direction parallel ray light source
Spot	Directional cone light source
Hemi	180° constant light source
Area	Directional area light source

An acceptable approximation was achieved by enclosing an high intensity red spot light. The enclosure has a thin, extruded slit through which the red light is projected. This concept is depicted figure 4.6, and attempts to model the collimated nature of laser light. The result of the modelled laser can be seen figure 4.7a

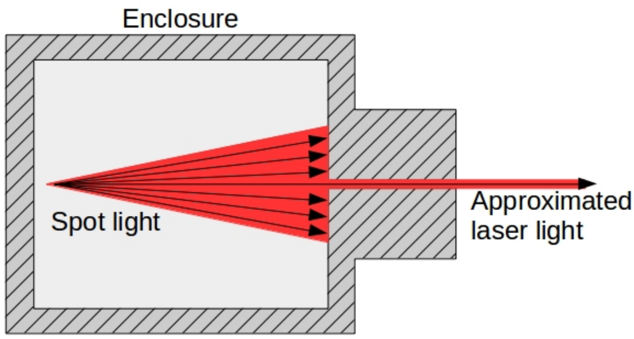


Figure 4.6: Laser model

4.2.2 Software

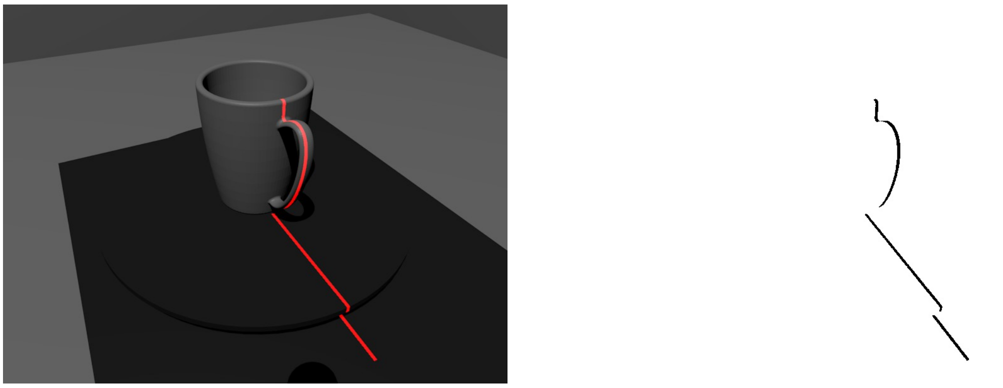
The previous section looked at the modelling, animation and rendering parts of creating a virtual scanner. In this section the final part of the simulation strategy is presented – the software that brings all the components together.

An Adaptive Binary Threshold Algorithm

In order to calculate 3D coordinates from the rendered images (and later the captured images) it necessary to isolate the laser line in each image. This is known as thresholding or segmentation; pixels with a light intensity below a certain value, called the threshold, is set to the minimum intensity value (0), while pixel with an intensity above the threshold are set to the maximum intensity value (254). Selecting an appropriate threshold value can be a difficult task due to reflections and changeable ambient light conditions. In their research on mapping underwater archaeological sites using laser scanning, [39], the authors calculated the threshold value as:

$$\tau = \mu + 2.5\sigma \tag{4.2}$$

where τ is the threshold value, μ is the mean intensity of the image, and σ the standard deviation. This approach served as a starting point for developing an adaptive threshold method shown in algorithm 4.1. This algorithm analyses the red channel of each RGB-image since the laser used in the simulation (and verification platform) projects red light. The algorithm is adaptive in the sense that the highest possible multiple of standard deviation is added to mean intensity to calculate the threshold value. This is done in an attempt to ensure that all that remains after the threshold operation is the laser line, even in the presence of changeable light conditions. The result of this algorithm can be seen in figure 4.7.



(a) Laser light modelled with Blender. (b) Same image after segmentation.

Figure 4.7: This figure shows the effect of the adaptive binary threshold algorithm.

A Simple Thinning Algorithm

The threshold operation described above is, on it's own, does not yet provide sufficient data required to calculate 3D coordinates from. An ideal laser would project a plane of light with an infinitesimal thickness. However, the light plane emitted by a real laser does have a real thickness, and the result is that the laser line, that is the intersection of the laser-light plane and the object surface, recovered by threshold operation can be several pixels thick. To improve the accuracy of the laser scanner it is necessary to determine where the *centre* of the laser light plane is. This is the research topic of several researchers . . .

The laser thinning approach used in this research is a compromise between accuracy an computational effort and is described in algorithm 4.2, the result of which can be seen in figure 4.8. Essentially, the algorithm traverses the image from top to bottom.

Algorithm 4.1 The binary threshold algorithm (Also see listing B.8).

Split image into RGB channels;
Find maximum intensity value, max_val, of the red channel;
Calculate the mean intensity, μ, of the red channel;
Calculate the standard deviation, σ, of the light intensity of the red channel;
Let the threshold value, $\tau = \mu + 4\sigma$;
if $\tau > max\_val$ **then**
  $\tau = \mu + 3.5\sigma$;
  if $\tau > max\_val$ **then**
    $\tau = \mu + 3\sigma$;
    if $\tau > max\_val$ **then**
      $\tau = \mu + 2.5\sigma$;
      if $\tau > max\_val$ **then**
        $\tau = \mu + 2\sigma$;
        if $\tau > max\_val$ **then**
          $\tau = \mu + 1.5\sigma$;
          if $\tau > max\_val$ **then**
            $\tau = \mu + \sigma$;
            if $\tau > max\_val$ **then**
              $\tau = \mu + 0.5\sigma$;
            end if
          end if
        end if
      end if
    end if
  end if
end if
Perform binary threshold on the red channel using τ and the *threshold()* function provided by OpenCV;

(a) Laser light after segmentation. (b) Same image after thinning.

Figure 4.8: This figure shows the effect of the thinning algorithm.

The left and right edges of the laser line is found, and the centre is calculated as the arithmetic mean of the edges of the laser line.

Point Cloud Estimation Algorithm

In the case of the simulated scanner, the point cloud estimation algorithm is responsible for: This section describes the software component of the scanner simulation.

1. reading the Blender generated images from directory where these images were stored,
2. performing the binary threshold operation, described in a previous subsection, on these images,
3. thinning laser lines within the segmented images and,
4. estimating 3D coordinates from pixel values using the equations discussed in section 2.6.

These operations are summarised in algorithm 4.3 and the source code can be found in listing B.12. Figure 4.9 shows the resulting point cloud for a simulated scan of a coffee mug.

4.3 Verification Platform

To confirm that the software algorithms (Algorithms 4.1, 4.2 and 4.3) developed for the scanner simulation could be applied in the real world, a physical scanner, or verification platform, was built. This section explains the hardware- and software architecture of this verification platform, and discusses the hardware components.

Algorithm 4.2 Algorithm for thinning the laser line after the threshold operation (Also see listing B.8).

Let *image_points* be an empty list of pixel values to describe the centre of the laser line;
Get input image;
for each row of the input image **do**
  for each column from the left of the input image **do**
    Get *intensity* of *pixel*;
    if $intensity > 0$ **then**
      $pixel_{left} = pixel$;
      Break out of for-loop;
    end if
  end for
  for each column from the right of the input image **do**
    Get *intensity* of *pixel*;
    if $intensity > 0$ **then**
      $pixel_{right} = pixel$;
      Break out of for-loop;
    end if
  end for
  $centre = 0.5(pixel_{right} + pixel_{left})$;
  Get *intensity* of *centre*;
  if $intensity > 0$ **then**
    Append *centre* to *image_points*;
  end if
end for

Algorithm 4.3 Point Cloud Estimation Algorithm (Simulated scanner)

Create empty point cloud, P;
Create memory buffer to hold images;
Load Blender generated images in directory into memory buffer;
for each image in memory buffer **do**
  Perform binary threshold;
  Perform thinning operation;
  Calculate object's z-axis rotation from previous image to current image, θ;
  for each image point **do**
    Calculate object point using equations given in section 2.6.2
  end for
  Rotate object points by θ;
  Append object points to point cloud, P;
end for
Write point cloud, P, to disk;

Figure 4.9: Point cloud of a coffee mug generated by the simulated scanner and related software.

4.3.1 Hardware Architecture

This section presents the hardware architecture of the physical scanner, or verification platform. This architecture, how all the components fit together, is shown in figure 4.10 and the main components are the PC, camera, laser, controller, and a stepper motor and driver. These hardware components are discussed in more detail in the following sub sections.

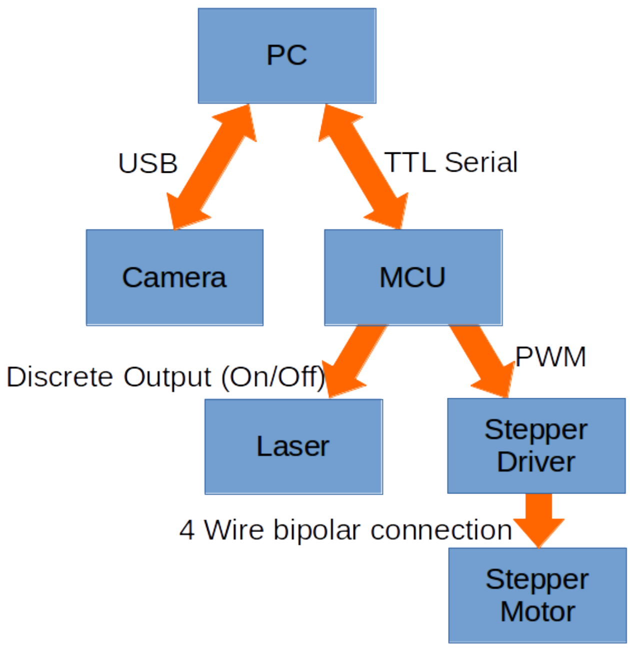


Figure 4.10: Hardware architecture.

PC

An ASUS N61Ja notebook was used in this research. The specification of this PC is summarised in table 4.3.

Table 4.3: PC specifications

Processor	4 x Intel Core i5 430M, 2.27 GHz
Operating System	Ubuntu 14.04 LTS 64 bit
Chipset	Intel HM55 Express Chipset
Memory	8 GB DDR3 1066 MHz SDRAM
Storage	Samsung 750 EVO 250 GB SSD
Graphics	ATI Mobility Radeon HD5730 1GB DDR3 VRAM

Camera

The camera used in this project is the Microsoft LifeCam Cinema (figure 4.11a). An OmniVision OV9712 CMOS imaging sensor is used with a pixel width and length of 3 μm. The camera is capable of capturing video at an resolution of 1280 x 720 at 30 frames per second depending on light conditions. For this project the camera was configured to capture image at resolution of 800 x 600 at the maximum frame rate possible. The LifeCam Cinema uses USB 2.0 to connect to the host PC. The camera is popular with amateur astronomers for lunar and planetary photography, and related internet forums provide valuable information not available in the official Microsoft documentation.

The following factors influenced the selection of this camera:

- the availability of data,
- the camera is designed and manufactured by a reputable company, Microsoft,
- the camera provides a good compromise between cost and quality.

Laser

A class III laser (figure 4.11b) that emits a red laser beam was used for 3D scanner. The laser module emits light with a wavelength of 650 nm. The laser module is connected to the Arduino and is activated by setting the digital output pin, pin 9, to high. Table 4.4 summarises the rest of the laser module specifications.

Table 4.4: Laser module specifications

Property	Min	Typical	Max	Units
Output power	2.5	3.0	5.0	mW
Working current	10	20	25	mA
Working voltage	2.3	4.5	8.0	VDC
Working temperature	-15	25	35	°C

Controller

A controller was needed to serve as an interface between the PC and the laser and the stepper motor. This controller turns the laser module on/off and generates the pulses for the stepper motor upon receiving a signal from the PC via TTL level serial communication. An Arduino Uno (figure 4.11c) was sourced for this purpose, and the specification for this controller is shown in table 4.5. The Uno was chosen for this project because it's ease to program.

Table 4.5: Arduino Uno specifications

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12 V
Input Voltage (limit)	6-20 V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Actuator and Accompanying Driver

Stepper motor To perform a scan it is necessary for the camera to view the entire surface of the artefact (excluding bottom/base view in this project). To achieve this, either the camera can be rotated about the artefact or, the artefact can be rotated in view of the camera. The later option was opted for since it was considered to be more practical. The object to be scanned is placed on a rotating platform. This platform is rotated by means of a bipolar NEMA 17 stepper (figure 4.11d). A stepper motor was selected because of the ability to control the rotational displacement and velocity accurately. Specifications for the NEMA 17 motor used in this project is shown in table 4.6.

Driver To control the stepper a drive circuit, or driver, was required. An EasyDriver V4.4 stepper driver (see figure 4.11e) was used to interface the Arduino Uno and NEMA 17 stepper motor. The EasyDriver is essentially a breakout board featuring an A3967 stepper driver. The driver is capable of microstepping, allowing step resolutions of full, 1/2, 1/4, and 1/8 steps. 1/8 microstepping is the default setting for the EasyDriver, requiring that 1600 pulses be generated by the Arduino Uno to rotate the 200 step/rotation stepper motor once. Table 4.7 provides a summary of the specifications of the EasyDriver V4.4.

Table 4.6: Specifications for NEMA 17 stepper motor

Step Angle	$1.8° \pm 5\%$ (200 steps/revolution)
Motor Length	34 mm
Rated Voltage	12 V
Rated Current	400 mA
Phase Resistance	30 Ω
Phase inductance	37 mH
Holding Torque	28 N.cm
Wires	4
Rotor Inertia	34 $g.cm^{-3}$
Detent Torque	1.6 N.cm
Weight	200 g

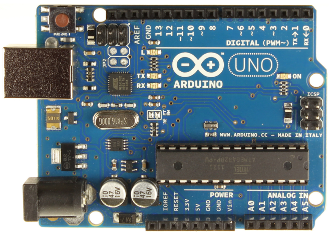
Table 4.7: Specifications EasyDriver V4.4

Driver chip	A3967
Microstepping	full, 1/2, 1/4, and 1/8 steps
Output current	150-750 mA/phase
Supply voltage	7-30 V

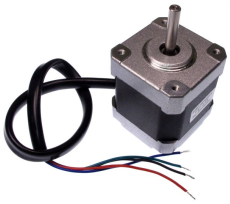
Figure 4.12 shows how the hardware components (Arduino Uno, laser module, stepper motor and stepper driver) where connected together. The completed scanner is shown in figure 4.13.

(a) Web camera.

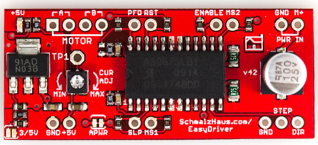
(b) Laser.



(c) Arduino Uno.



(d) NEMA 17 stepper motor.



(e) EasyDriver V4.4.

Figure 4.11: Hardware components for the 3D scanner.

Figure 4.13: The final 3D scanner.[1]

[1]Photographer: Nienke van Jaarsveld

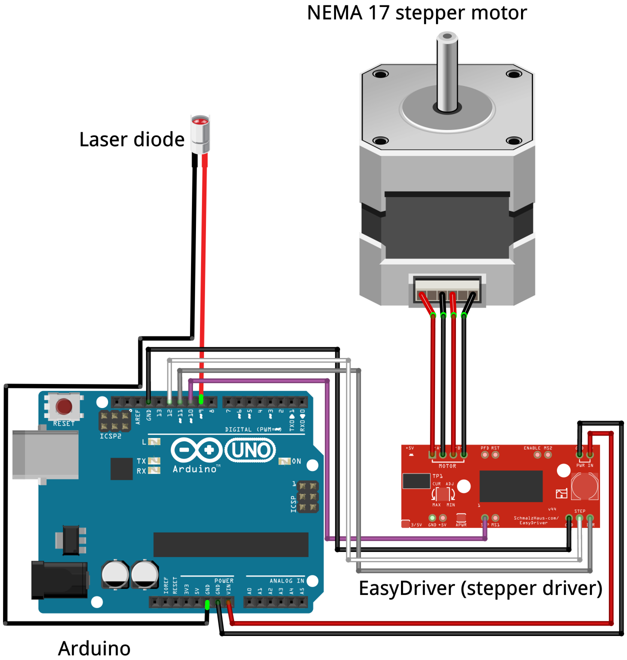


Figure 4.12: The electrical connection of the hardware components.

4.3.2 Software

The program flow for the verification platform is very similar to that of the simulated scanner. The main difference is that instead of reading rendered images from a directory on the PC hard drive, images are captured with a camera. This section describes the operation of the verification platform and how that was realised programmatically.

Scanner Operation

This section describes the sequence of actions performed by both the PC-side software and the micro-controller during a scan cycle.

On the PC: Once an operator or user has placed an object on the scanner and initiated a scan cycle, the PC performs the following:

1. Pings the camera. If the camera is not connected a scan will not be attempted.

2. Pings the micro-controller. If the micro-controller is not connected a scan will not be attempted.

3. At this point, if both camera and micro-controller are connected, the PC signals the micro-controller to turn on the laser and to rotate the stepper motor a full revolution.

4. Since the camera takes a few seconds to adjust to ambient light, the MCU waits 5 seconds before commencing with rotating the stepper. Upon completion of this 5 second waiting period, the PC is notified by the MCU that rotation has started. At this point the PC starts grabbing frames from the camera.

5. After the stepper motor has made a complete revolution the MCU signals the PC to stop grabbing more frames.

6. The PC software then proceeds to process the captured images. The images are segmented and thinned using algorithms 4.1 and 4.2 respectively.

7. Next the software calculates the point cloud of the object using the equations given in section 2.6.2

On the MCU: The interaction between the MCU and PC is as follows:

1. Replies to the ping received from PC.

2. Upon receiving the message from the PC to start stepper motor rotation, the MCU turns on the laser, wakes the stepper motor driver and waits 5 seconds. This waiting period is to allow the camera to settle and adjust to ambient lighting conditions.

3. After waiting 5 seconds the MCU signals the PC that it is about to start rotating the stepper motor. The MCU then starts rotating the stepper by sending 1600 pulses to the stepper driver. During this time the PC is capturing images from the camera.

4. Once the stepper motor has made a full rotation MCU turns the signals the PC to stop capturing images and then switches the laser off.

5. The MCU then waits for the next scan cycle.

The laser scanner software for the PC and MCU, described above, are available in appendix B.1. To help implement the software described above several third party libraries were used. These libraries are discussed in the subsection that follows.

Software Libraries

Boost Boost is a collection of C++ libraries that aims to provide functionality not yet part of the standard C++ libraries. Boost was used in this project to implement multi-threading, serial communication and file system operations. Boost 1.54 was employed in this project.

OpenCV OpenCV is a popular open source computer vision library. OpenCV contains a large number of computer vision algorithms and this library was employed to perform all of the image processing, and interaction with the camera, required in this research. The software in this project relies on OpenCV 2.4.8.

Point Cloud Library The Point Cloud Library is an open source library mainly used for 3D point cloud processing. For the laser scanner software the built-in visualisation and write-file functionality of Point Cloud Library was used. Point Cloud Library 1.7.2 was used in this research.

Object Oriented Approach

The laser scanner software was written in an object oriented way. This subsection provides brief description for each of the classes that make up the laser scanner software.

SerialPort This class contains methods needed for serial communication and depends the *boost::asio*, *boost::chrono*, and *boost::thread* libraries. This class provides serial (UART) communication between the PC and Arduino Uno. The source code of this class is given in appendix B.1.1.

Controller Methods needed for controlling the stepper motor and laser module are contained in this class. The class depends on the *SerialPort* class. Source code for this class is in appendix B.1.2.

Camera Methods required to access the camera, capture images and perform a camera calibration are contained in the *Camera* class. This class relies on a number OpenCV modules and the source code is documented in appendix B.1.3.

ImageProcessor This class contains all the required image processing methods, such as the adaptive binary threshold algorithm (Algorithm 4.1), the laser line thinning algorithm (Algorithm 4.2) and the method for removing distortions from captured images. The methods in this class depend on several OpenCV modules, specifically the *imgproc* module. The source code for the *ImageProcessor* class can be found in appendix B.1.4.

LaserScanner The *LaserScanner* class ties all of the above mentioned classes together, and provides the necessary methods to perform a 3D scan and visualise the acquired point cloud. The class also contains methods required to calibrate the scanner. In addition to the above mentioned classes,the *LaserScanner* class also depends on a few Point Cloud Library (PCL) modules for file I/O and point cloud visualisation. Appendix B.1.5 documents the class definition (*laserscanner.hpp*) and implementation (*laserscanner.cpp*) for the *LaserScanner* class.

4.3.3 Scanner Calibration

In the case of the simulated scanner, the geometry of the scanner is completely known since it is essentially a computer generated 3D model. However, this is not true for the verification platform. Inconsistencies are introduced during the construction of the scanner and the manufacturing of the camera. This means that the actual distances and angles must be determined in order to perform scans as accurately as possible. This is done by calibrating the scanner. This section will describe how the camera calibration was achieved.

Calibration in the case of the scanner can be defined as finding the:

- camera parameters contained in the camera matrix, $\boldsymbol{A}$,
- rotation transformation matrix, $\boldsymbol{R}$, of the camera coordinate system with respect to the world coordinate system and,
- translation transformation matrix, $\boldsymbol{t}$, of the camera coordinate system with respect to the world coordinate system.

The camera matrix can be determined using the camera calibration method described in section 2.4. Finding the rotation and translation matrices is equivalent to finding the pose of the world coordinate system with respect to the camera. This can be done by capturing an image of a number points with known positions, referred to as object points denoted by $P(X, Y, Z)$, and finding how those object points relate to the corresponding points in the image, refered to as image points and denoted as $p(u, v)$.

The dotted pattern in figure 4.14 provides the object points needed for calibrating the scanner. The pattern is placed on the scanner such that the pattern coincides with the plane if light projected by the laser. The calibration software finds the image points by calculating the centroids of the elements on the calibration pattern using OpenCV's *SimpleBlobDetector*. Figure 4.15 shows the centroids detected during a calibration.

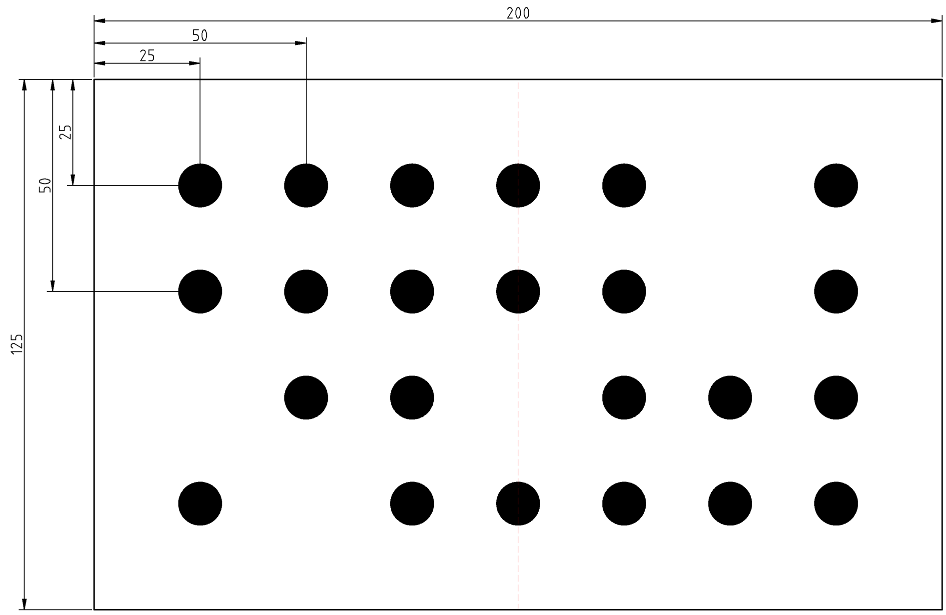


Figure 4.14: Scanner calibration pattern. (Not to scale)

Once both sets of points – the object points and corresponding image points – are available it possible to estimate $\boldsymbol{R}$ and $\boldsymbol{t}$ using the *solvePnP* function available in OpenCV. Table 4.8 summarises the object- and corresponding image points obtained during a calibration. The resulting rotation matrix and translation vector is shown in equations 4.3. Note that these values closely relate those of the simulated scanner in equations 4.1. The scanner calibration is implemented as part of the LaserScanner class, the source code of which can be found in appendix B.1.5.

$$\boldsymbol{R} = \begin{bmatrix} 0.9282 & -0.3338 & -0.1642 \\ -0.3333 & -0.5507 & -0.7652 \\ 0.1650 & 0.7650 & -0.6224 \end{bmatrix} \qquad \boldsymbol{t} = \begin{bmatrix} 15.31 \\ 3.67 \\ 321.90 \end{bmatrix} \tag{4.3}$$

4.4 Summary

This chapter addressed the second objective of this project as stated in section 2.6.2:

> *"Implement a 3D scanner to demonstrate an understanding of the underlying principles."*

In summary, the 3D scanner and related software developed in during the course of this research project were presented in this chapter. The chapter:

- introduced the concept for the scanner,

Figure 4.15: Finding centroids for calibration.

- discussed the need for, and implementation of the simulated scanner,

Table 4.8: Image– and object points used to estimate $\boldsymbol{R}$ and $\boldsymbol{t}$ for the verification platform.

Pattern Element	u	v	X (mm)	Y (mm)	Z (mm)
1	509.767	384.182	0	-75	25
2	472.376	344.466	0	-50	25
3	438.979	308.103	0	-75	50
4	409.726	276.741	0	-25	25
5	514.974	328.689	0	-50	50
6	474.725	289.566	0	0	25
7	520.285	265.159	0	-75	75
8	439.753	255.573	0	-25	50
9	383.857	248.787	0	25	25
10	339.139	200.501	0	75	25
11	381.531	198.798	0	25	50
12	439.99	196.448	0	-25	75
13	526.519	192.178	0	-75	100
14	407.41	168.351	0	50	50
15	357.208	174.972	0	0	75
16	330.774	102.078	0	25	75
17	378.895	143.672	0	-25	100
18	440.653	129.819	0	50	75
19	353.591	121.774	0	0	100
20	406.229	104.54	0	75	75
21	376.122	82.6321	0	25	100
22	349.279	61.3419	0	50	100
23	325.815	45.4718	0	75	100

- documented the adaptive binary threshold-, thinning-, and point cloud estimation algorithms used in the scanner software,
- presented the verification platform– a physical implementation of the simulated scanner,
- detailed the hardware components and software of the physical 3D scanner, and
- documented the scanner calibration procedure.

The following chapter documents the implementation of the *object recognition* part of this research.

Chapter 5

Object Recognition Implementation

5.1 Introduction

This chapter documents the work done in order to fulfil the third objective set out in section 1.1.2:

> *"Develop an ANN or ANNs to perform Object Recognition. This includes researching ways to reduce point cloud data to an input vector of manageable length for the proposed ANN(s)."*

5.2 Preprocessing

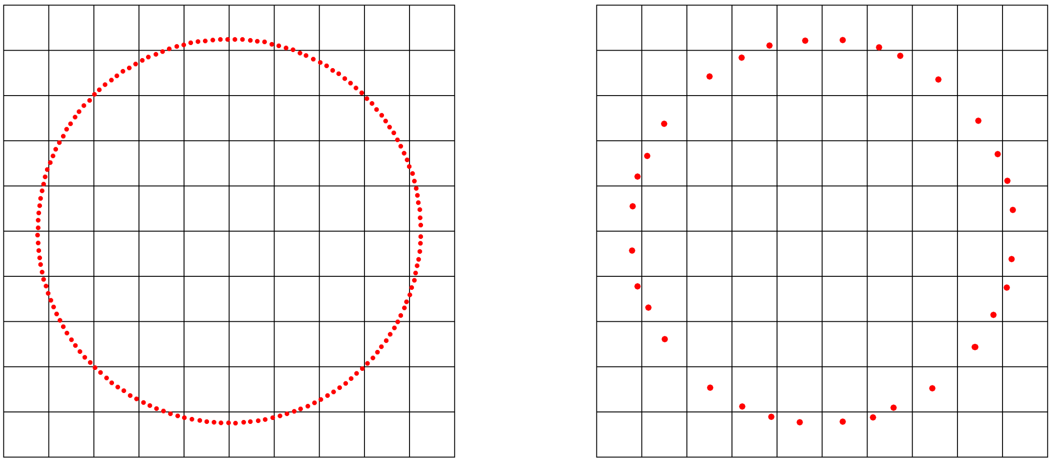
This section discusses the two preprocessing techniques that were applied to a point cloud before being presented to the ANN for recognition:

1. *Voxel grid filtering*, and
2. *global point feature histograms.*

5.2.1 Data reduction

Point clouds may contain many data points, many more than what may be necessary for the object recognition task. Reducing the number of points in a point cloud, in such a way that the overall surface information remains intact, will reduce the computational load during recognition. The this was achieved in this project was by filtering the point clouds with a voxel grid filter. The voxel grid filter is a standard filter available in the PCL. Following is an explanation how this filter works.

The voxel grid filter in PCL creates a 3D grid, such that, this grid encompasses the point cloud to be filtered. This grid is subdivided into cells known as voxels. Each voxel is then processed. Where a voxel contains any points, these points are replaced by a single point. This single point is calculated as the mean or centroid of the original set of points in that voxel. Figure 5.1 is a 2D illustration of this filtering process. In this illustration it is clear that even though the number of points are significantly less after filtering, the original shape – a circle – is still recognisable. An example of voxel grid filtering can be seen in figure 5.2.



(a) Before filtering.

(b) After filtering.

Figure 5.1: A 2D illustration of voxel grid filtering.

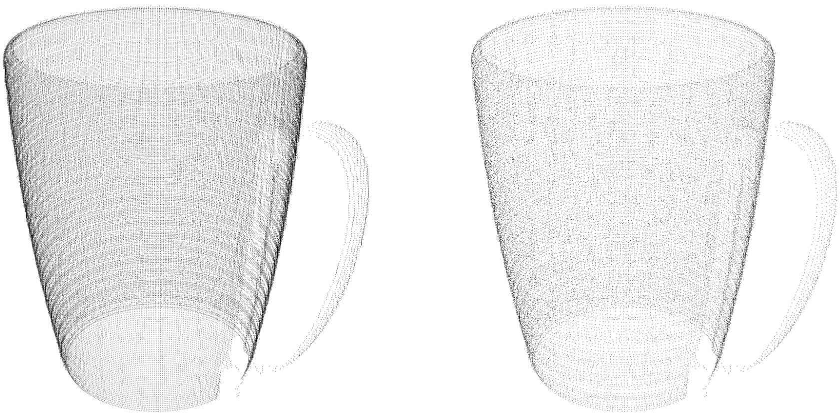


Figure 5.2: Point cloud of a coffee mug before (left) and after (right) voxel grid filtering.

5.2.2 The Global Point Feature Histogram

As seen in chapter 3, the classical Feed Forward Neural Network (FFNN) accepts an input vector, $\boldsymbol{z}$, containing a known- and fixed number of elements. This is not usually a property of point clouds. Even when taking several scans of the *same* object, those scans my differ in the number points they contain. These differences can occur because of changes in lighting conditions during the scan, or the object being moved between scan cycles. Objects with a larger surface area will also result in point clouds with more points. To combat this issue of varying pattern length, more preprocessing of some kind needed to be done. This further preprocessing incorporated part of an approach developed by [40]. Some of the authors of that paper developed several *descriptors* – histograms for point clouds. These descriptors can be thought of as an object's signature. The histograms proposed in [26], [41], [42] possess the following useful qualities:

1. Reduced number of data points compared to the original point cloud.
2. Fixed number of data points.
3. Attempts to retain geometric information of the original point cloud.

The Global Point Feature Histogram proved useful for this research. This histogram summarises the angles between all surface normals and the axes of a local reference frame placed at the centroid of the point cloud. The histogram consists of 135 bins – 45 bins for each of the 3 angles. What follows is a description of how global point feature histograms are calculated.

Consider the point cloud $\boldsymbol{P}$. To find the global point feature histogram, $\boldsymbol{H}_{GPF}$, of $\boldsymbol{P}$, we follow these steps:

1. Find the centroid, $\boldsymbol{p_c}$, of $\boldsymbol{P}$.
2. Place a unit vector, $\boldsymbol{n_c}$, at the centroid. $\boldsymbol{n_c}$ points in the z direction.
3. For each point, $\boldsymbol{p_i}$:
   (a) Find the surface normal, $\boldsymbol{n_i}$.
   (b) Place a local reference frame (u,v,w) at $\boldsymbol{p_c}$ using equations 5.1.
   (c) Calculate angles, α_i, ϕ_i and θ_i, using equations 5.2.
   (d) Update the bins of the histogram according to size of the angles calculated in the previous step.

$$\boldsymbol{u_i} = \boldsymbol{n_c}, \qquad \boldsymbol{v_i} = \frac{\boldsymbol{p_i} - \boldsymbol{p_c}}{\|\boldsymbol{p_i} - \boldsymbol{p_c}\|} \times \boldsymbol{u_i}, \qquad \boldsymbol{w_i} = \boldsymbol{u_i} \times \boldsymbol{v_i} \tag{5.1}$$

$$\alpha_i = \arccos\left(\boldsymbol{v_i} \cdot \boldsymbol{n_i}\right), \quad \phi_i = \arccos\left(\boldsymbol{u_i} \cdot \frac{\boldsymbol{p_i} - \boldsymbol{p_c}}{\|\boldsymbol{p_i} - \boldsymbol{p_c}\|}\right), \quad \theta_i = \arctan\left(\boldsymbol{w_i} \cdot \boldsymbol{n_i}, \boldsymbol{u_i} \cdot \boldsymbol{n_i}\right) \tag{5.2}$$

Figures 5.3 and 5.4 shows point clouds of 8 objects, each with their corresponding GPFH.

The global point feature histogram described above was implemented using the Viewpoint Feature Histogram class in PCL. This type of histogram contains 308 bins, and is actually a concatenation of several histograms. The first 135 bins is the GPFH as described above. This histogram contains information about the objects shape. The next 45 bins contains information about the objects scale, or size. This was not used in this research. The final 128 bins contains information about the viewpoint from which the object was viewed – not relevant in this research since the objects are scanned from all round.

5.3 ANN Implementation

This section will discuss how the ANNs were implemented in this research. For clarity the section is separated into two subsections. The first of these subsections discusses the training and testing data, while the second subsection will discuss the training and testing itself.

5.3.1 Training, Testing and Validation Datasets

Initially it was thought that training, testing and validation data would be obtained from *scans* performed by the simulated scanner. This soon proved to be problematic due to the significant time it takes to render the images for these simulated scans. Generating a sufficient amount of training, testing and validation samples in this way would take an impractically long time. A different approach was needed – it is presented below.

Point clouds of each of the eight objects were generated in two ways:

Method 1: Point clouds of the eight test objects are generated by means of the simulated scanner. This method is slow and impractical for generating large datasets.

Method 2: The PCL comes with command line utilities, *pcl_ply2pcd* and *pcl_obj2pcd*, that allows one to generate point clouds (*.pcd) from 3D models (*.ply and *.obj). In addition to the point clouds generated by means of the first method, point clouds were also generated using *pcl_ply2pcd*. This method has two main advantages:

1. Reduced generating time.
2. Can be easily automated with some scripting language like, Bash or Python.

These point clouds were then used as a starting point for generating the required training, testing and validation datasets. A program (see appendix B.2.1) was written to generate the data for these datasets according to algorithm 5.1. Essentially this program generates new point clouds by adding random noise to the X, Y and Z components of each point of the point clouds. The program also introduces random translation within the XY-plane, and random rotation about the Z-axis. Finally the program calculates point feature histograms (discussed in subsection 5.2.2) for each of the newly generated point clouds. These point feature histograms, along with their corresponding object identities, become the training, testing or validation dataset.

Algorithm 5.1 Algorithm for generating training, validation or testing data

Create memory, *clouds*, to hold point clouds;
Create memory, *output*, to hold dataset;
Read from disk point clouds into *clouds*;
for each *pointcloud* in *clouds* **do**
  for each *point* in *pointcloud* **do**
    $point.x = point.x +$ noise sampled from uniform distribution on $[-0.5, 0.5]$ in mm;
    $point.y = point.y +$ noise sampled from uniform distribution on $[-0.5, 0.5]$;
    $point.z = point.z +$ noise sampled from uniform distribution on $[-0.5, 0.5]$;
  end for
  Rotate *pointcloud* about z-axis an angle from uniform distribution on $[-\pi, \pi]$;
  Translate *pointcloud* along x-axis a distance from uniform distribution on $[-10, 10]$ in mm;
  Translate *pointcloud* along y-axis a distance from uniform distribution on $[-10, 10]$;
  Calculate GPFH for *pointcloud*;
  Append histogram and *pointcloud* ID to *output*;
end for
Write *output* to disk;

5.3.2 Training and Validation

Network Architecture

The networks developed for this project were simple three layer FFNNs. Each network contains:

- 135 input neurons (the number of elements in a point feature histogram),
- a hidden layer with the number of neuron to be determined, and
- 8 output neurons – a neuron for each object to be recognised.

80 networks were trained to establish the optimum number of neurons required for the hidden layer. These networks can be divided into 8 groups, each group containing 10 examples. The groups are summarised in table 5.1.

Table 5.1: Summary of training groups

Group	Hidden neurons	Networks in group	Results
1	2	10	Appendix A.1
2	5	10	Appendix A.2
3	10	10	Appendix A.3
4	15	10	Appendix A.4
5	20	10	Appendix A.5
6	30	10	Appendix A.6
7	40	10	Appendix A.7
8	60	10	Appendix A.8

Training Strategy

The ANNs implemented in this research were trained using a batch back-propagation learning algorithm based on algorithm 3.1. This means that the network's weights are updated only once all training patterns have been presented. The source code of the implementation can be found in appendix B.2.2.

Over-fitting occurs when a neural network starts memorising each of the patterns in the training set. The neural network loses the ability to generalise and correctly classify previously unseen examples – this is not desirable. To prevent over-fitting from occurring the mean square error (MSE) of the training set was compared to that of a validation dataset. Over-fitting is observed when $MSE_{validation}$ increases while $MSE_{training}$ decreases. In this project over-fitting was deemed to be occurring when an increase in $MSE_{validation}$ was observed for 20 consecutive epochs.

The learning rate, η, was kept fixed at 0.3 for each network trained. Similarly, the momentum, α, was fixed at 0.5.

To implement the ANNs, the Fast Artificial Neural Network library was used. This is an open source C/C++ library developed by [cite work of FANN dev].

The stopping conditions for training the neural network were the following: The neural network training was stopped when any of the following conditions were true:

1. Over-fitting observed.
2. Maximum number of epochs reached ($epochs_{max} = 50000$).
3. Minimum required $MSE_{training}$ ($MSE_{training} \leq 0.0001$).

Once all 80 networks were trained, they were then evaluated using a previously unseen test dataset. This evaluation, and the results thereof, will be presented in the next chapter.

5.3.3 Object Recognition Software Operation

An object recognition program (see appendix B.2.3) was developed that utilises what is considered the best neural network found during training. This command-line program can be called as follows:

```
$ ./3d_recognition input.pcd
```

Figure 5.5 shows the output of the object recognition program. The object was in this case correctly identified as Object 1.

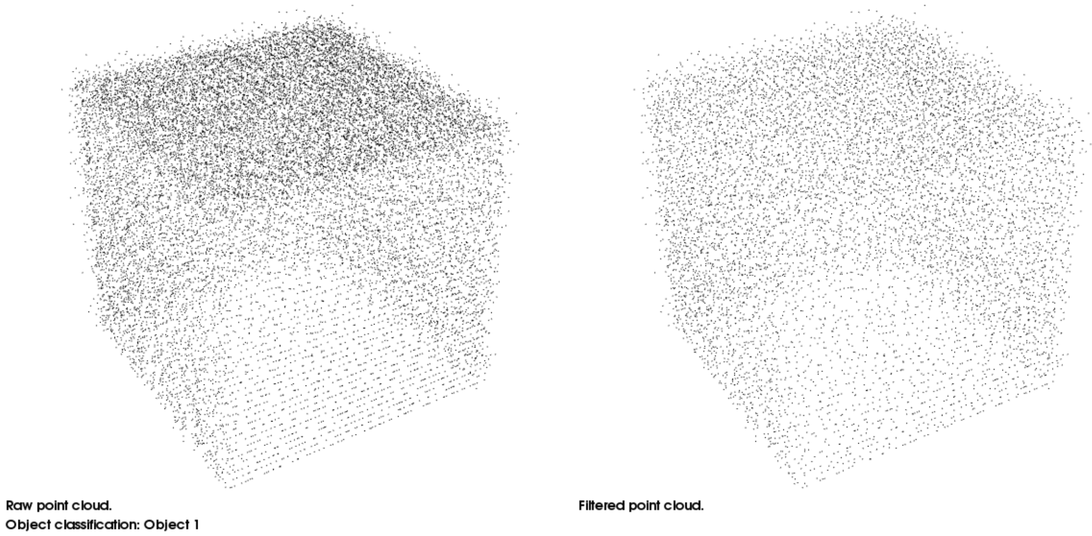
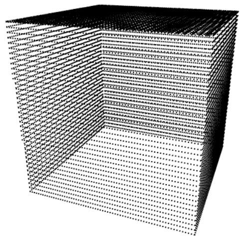


Figure 5.5: Output of the object recognition software.

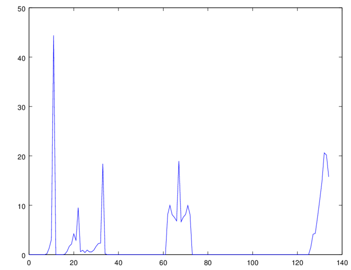
5.4 Summary

This chapter detailed the implementation of the object recognition part of this study. Data preprocessing, using voxel grid filtering and global point feature histograms were presented. Dataset generation was discussed, as well as the training and validation of the artificial neural networks.

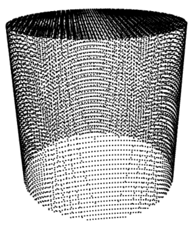
The next chapter documents the results obtained during this study.



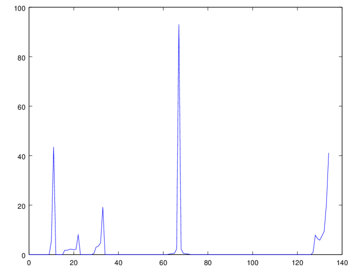
(a) Object 1: Cube



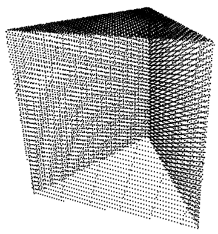
(b) Histogram: Object 1



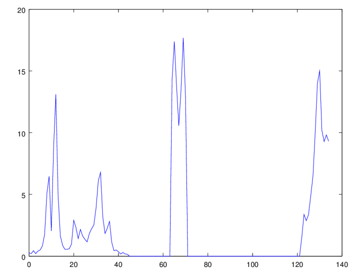
(c) Object 2: Cylinder



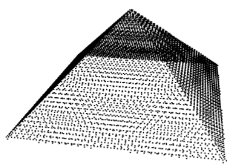
(d) Histogram: Object 2



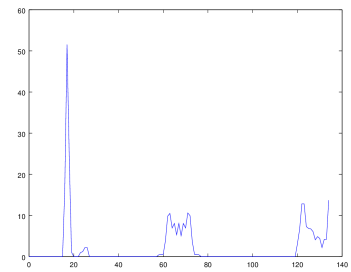
(e) Object 3: Triangular prism



(f) Histogram: Object 3

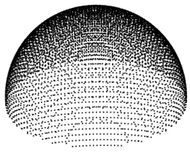


(g) Object 4: Square base pyramid

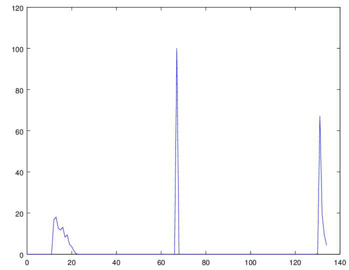


(h) Histogram: Object 4

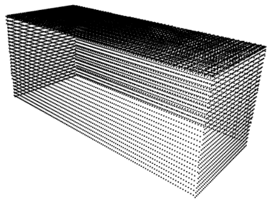
Figure 5.3: An illustration of point clouds with their corresponding global point feature histograms. Continued in figure 5.4



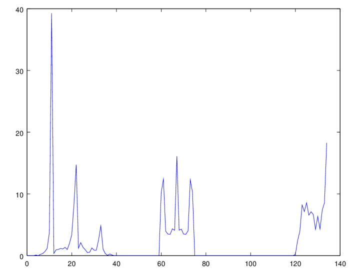
(a) Object 5: Hemisphere



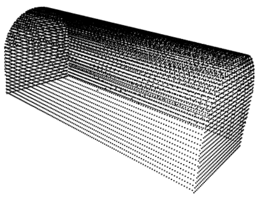
(b) Histogram: Object 5



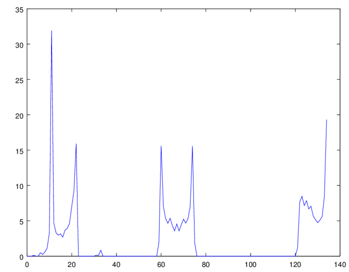
(c) Object 6: Rectangular prism



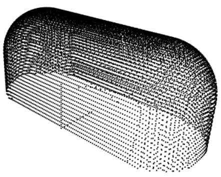
(d) Histogram: Object 6



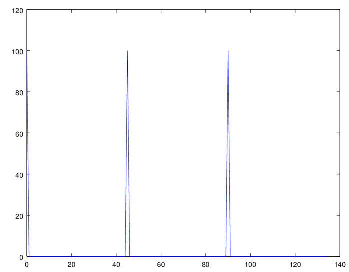
(e) Object 7: Variation of object 6



(f) Histogram: Object 7



(g) Object 8: Variation of object 7



(h) Histogram: Object 8

Figure 5.4: Continued from figure 5.3

Part III

Validation

Chapter 6

Results and Discussion

This chapter presents the results of this research, and provides some explanation and discussion where deemed necessary. Following is a short explanation of the layout of this chapter. Section 6.1 documents the:

- Calibration results of the scanner's camera. This is done in subsection 6.1.1.
- Image processing results. In the context of image processing, a comparison between the simulated scanner and verification platform is shown in subsection 6.1.2.
- 3D reconstruction results. Subsection 6.1.3 shows point clouds that were constructed from images, using the software developed during this research project. Point clouds generated using the simulated scanner, as well as the actual scanner are shown.

The second part of this chapter, section 6.2, documents the result of the object recognition objective of this study. This section is divided into two subsections:

- The first part of this section, subsection 6.2.1, discusses the theory behind the performance measures used to evaluate the object recognition results in this research.
- Next, subsection 6.2.2, summarises the object recognition results.

6.1 Scanner Results

6.1.1 Camera Calibration

The camera of the 3D scanner was calibrated using the method described in chapter 2, section 2.4. Figure 6.1 shows some of the images captured while performing the calibration. The calibration pattern used, was a 10 × 7 checker board pattern with the sides of each square being 23.5 millimetres in length.

The calibration yielded a camera matrix, $\boldsymbol{A}_{verification}$, shown in equation 6.1. This camera matrix compares well to that of the simulated scanner (see equation 6.2).

$$\boldsymbol{A}_{verification} = \begin{bmatrix} 727.09 & 0 & 397.45 \\ 0 & 730.56 & 309.88 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

$$\boldsymbol{A}_{simulation} = \begin{bmatrix} 600 & 0 & 400 \\ 0 & 600 & 300 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.2}$$

The radial and tangential distortion coefficients obtained by the calibration are shown in equations 6.3 and 6.4 respectively.

$$k_1 = -3.42 \times 10^{-3} \qquad k_2 = -1.44 \times 10^{-1} \qquad k_3 = 1.48 \times 10^{-1} \tag{6.3}$$

$$p_1 = 6.03 \times 10^{-3} \qquad p_2 = -4.35 \times 10^{-3} \tag{6.4}$$

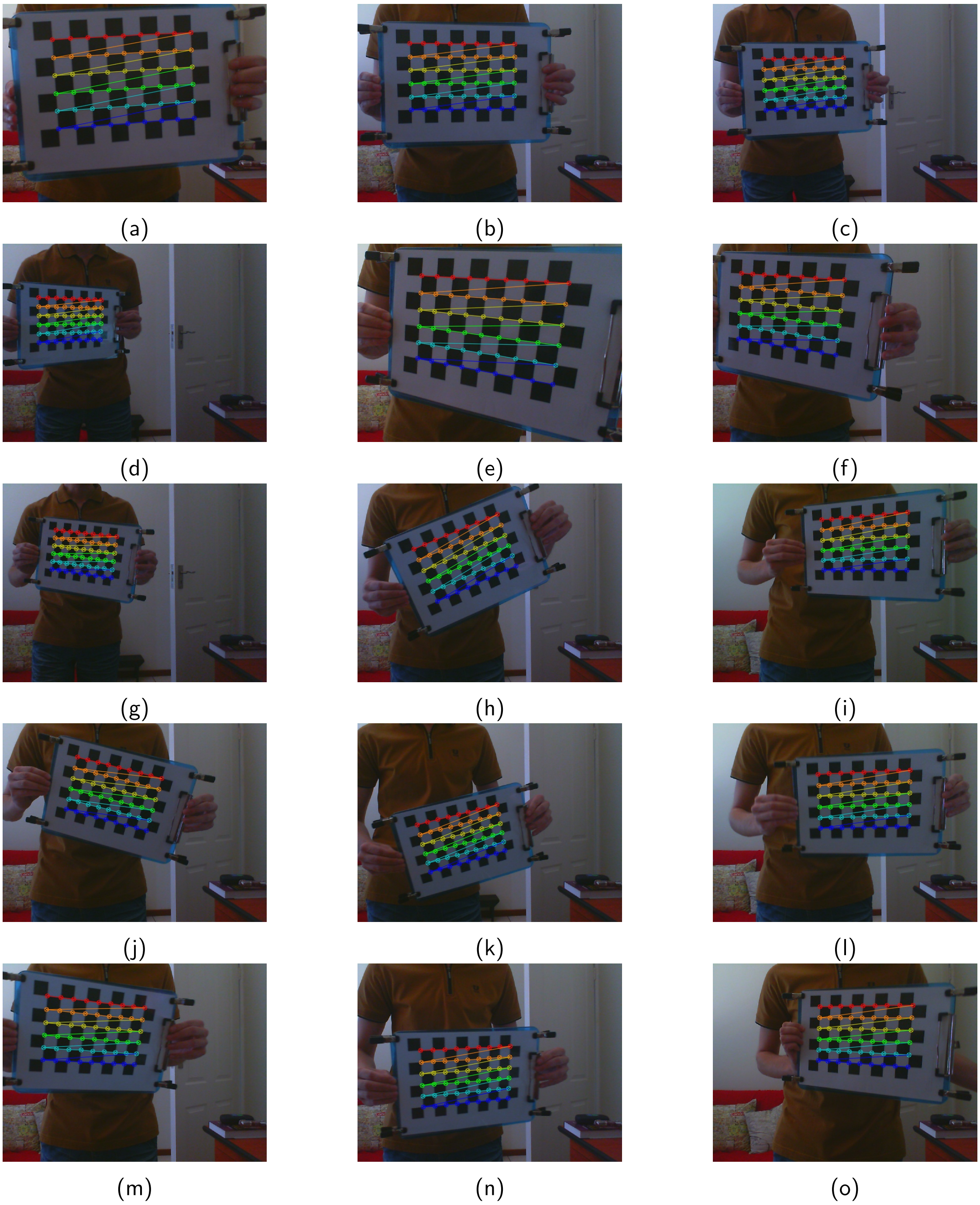
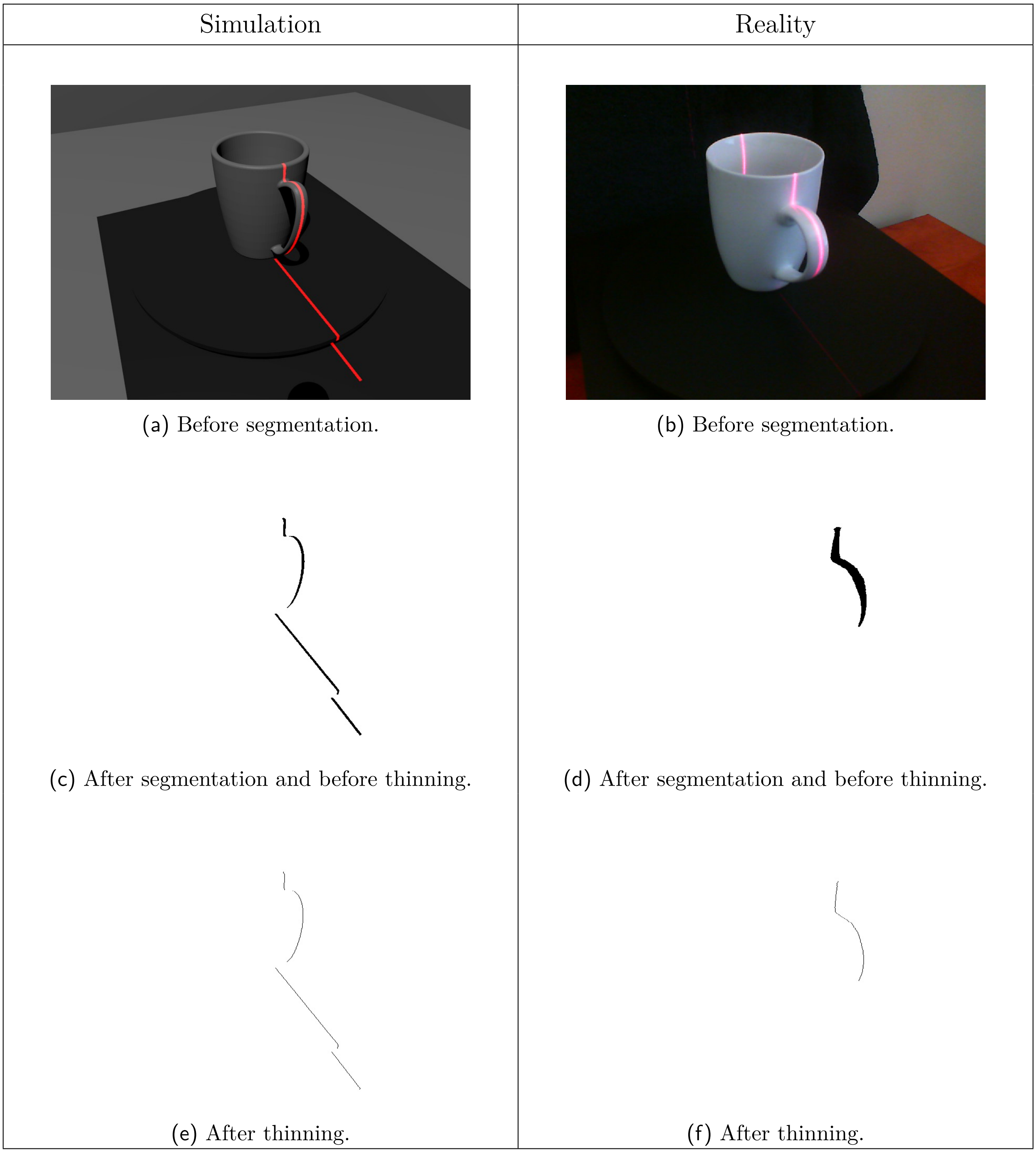


Figure 6.1: A selection of the images captured during the camera calibration procedure.

6.1.2 Image Processing

This subsection briefly shows the results of the image processing algorithms discussed in chapter 4, section 4.2.2. Figure 6.2 is a comparison showing how the image processing algorithms perform on both synthetic and real images.



Simulation | Reality

(a) Before segmentation.

(b) Before segmentation.

(c) After segmentation and before thinning.

(d) After segmentation and before thinning.

(e) After thinning.

(f) After thinning.

Figure 6.2: Image processing comparison: Simulation versus reality.

6.1.3 3D Reconstruction

This subsection presents the results of point cloud estimation algorithm discussed in chapter 4, section 4.2.2. See figure 6.3 below.

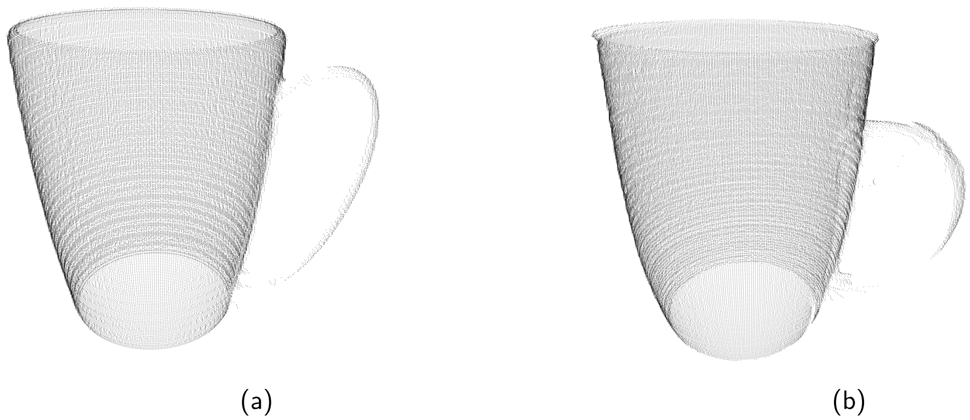


Figure 6.3: A comparison of a mug point cloud produced with the scanner simulation (a), and with the actual scanner (b).

6.2 ANN Training and Validation Results

6.2.1 Performance Measures for Classification

This subsection presents the theory behind the performance measures that were used to evaluate the performance of the artificial neural networks trained in this study. [43] analyses performance measures for classification tasks, and that work influenced the approach taken in this research.

The performance evaluation of the artificial neural networks was based on the following performance measures:

- Accuracy
- Precision
- Negative Predictive Value
- Sensitivity
- Specificity

These performance measures are discussed below.

Accuracy

The author of [43] describes accuracy as being the classifier's overall effectiveness. The classifiers being evaluated in this study being artificial neural networks. Equation 6.5 calculates the accuracy.

$$ACC = \frac{TP + TN}{TP + FN + FP + TN} \tag{6.5}$$

Precision (Positive Predictive Value)

This performance measure quantifies how well a classifier can make positive identifications. A positive identification, for example, is the neural network determining that a given object **is** in fact object 1. The following equation shows how to calculate the positive predictive value.

$$PPV = \frac{TP}{TP + FP} \tag{6.6}$$

Negative Predictive Value

This performance measure quantifies how well the neural network can make negative identifications. A negative identification, for example, is the neural network determining that a given object is **not** object 1. The following equation shows how to calculate the negative predictive value.

$$NPV = \frac{TN}{TN + FN} \tag{6.7}$$

Sensitivity (True Positive Rate)

Sensitivity is the measure of how effective the neural network can make positive identifications. The sensitivity is calculated using the following equation.

$$TPR = \frac{TP}{TP + FN} \tag{6.8}$$

Specificity (True Negative Rate)

Specificity is the measure of how effective the neural network can make negative identifications. The specificity of the network is calculated using equation 6.9.

$$SPC = \frac{TN}{TN + FP} \tag{6.9}$$

6.2.2 Object Recognition Results

Training

This section presents a summary of the training performed for the 80 artificial neural networks. Note that due to the large quantity of data generated by training the 80 neural networks, it is not practical to present all the results here. The complete results are documented in appendix 6.2.

The time taken to train a neural network can be described by two variables:

1. The number of times the network evaluated the training set during the training. This is known as the epochs.

2. The actual time spent in seconds to perform the training.

Table 6.1 and figure 6.4 summarises the training times of the various artificial neural networks that were developed in this study. As can be seen, the variations in training times are quite significant, nevertheless, there are some trends that emerged.

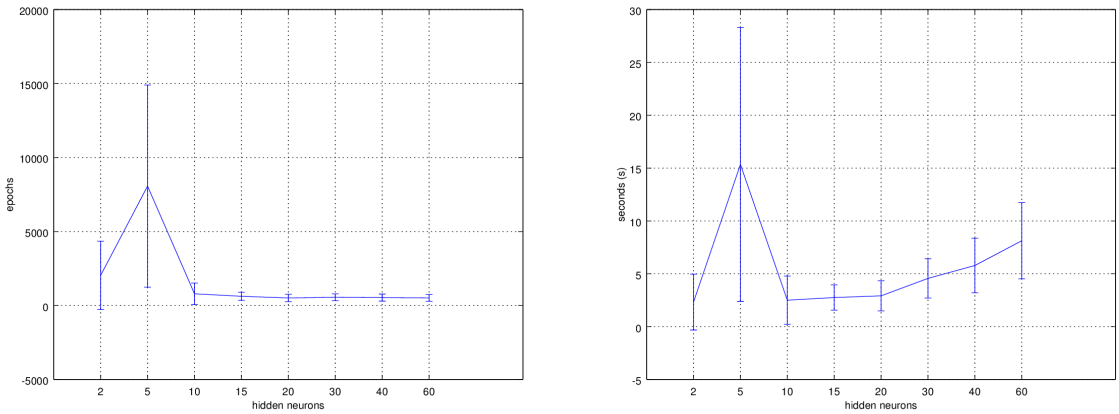
The neural networks with 5 hidden neurons took a significant amount of epochs and time to train. This is likely because 5 neurons were only just not enough neurons to learn the object recognition task.

Neural networks with 2 hidden neurons were even more ill equipped for the particular task, but training times were significantly less than in the case of 5 hidden neurons. The reason for this is over-fitting. The 2 hidden neurons is just about enough to enable the network to learn to identify 1 of the 8 objects within a reasonable amount of time and further training leads to over-fitting.

The remaining neural networks took a similar number of epochs to learn the recognition task, while a steady increase in actual training time was observed. This increase in training time was expected, since more hidden neurons will result in more computations that must be performed during training.

Table 6.1: A summary of the training of the different ANNs

Neurons	Epochs	Training Time (s)
2	2036.10 ± 2313.42	2.33 ± 2.64
5	8072.10 ± 6831.11	15.35 ± 12.97
10	793.80 ± 726.38	2.51 ± 2.28
15	626.40 ± 271.39	2.76 ± 1.19
20	513.80 ± 250.71	2.92 ± 1.43
30	558.90 ± 227.78	4.57 ± 1.86
40	538.70 ± 239.01	5.79 ± 2.57
60	518.40 ± 230.56	8.13 ± 3.61

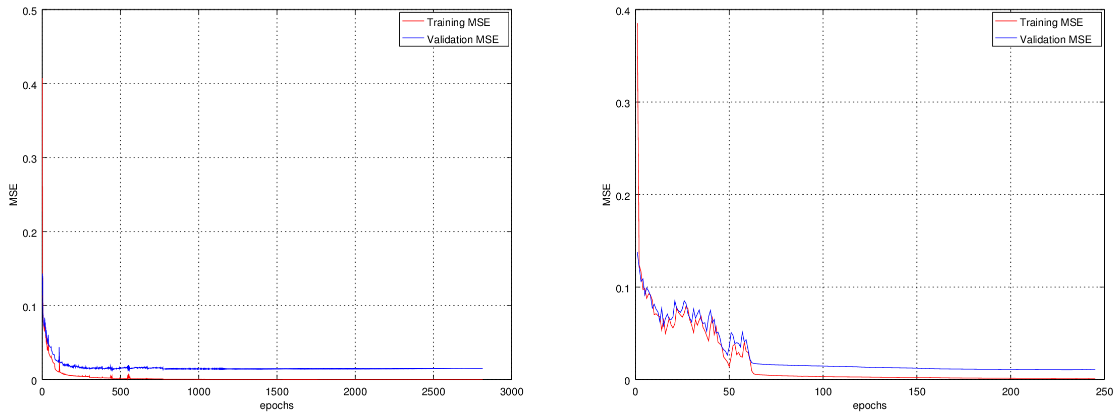


(a) Mean epochs for all tests

(b) Mean training time for all tests

Figure 6.4: Training time

Figure 6.5 highlights how the mean square error reduced during training.



(a) 10 hidden neurons in hidden layer, test 2

(b) 40 neurons in hidden layer, test 7

Figure 6.5: Mean Squared Error during training

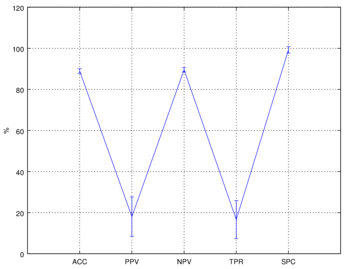
Performance Evaluation

This section documents the results of the performance evaluation of the 80 neural networks that were trained as part of this research project. Again, it is not practical to present all the results here, and the complete results can be found in appendix 6.2.

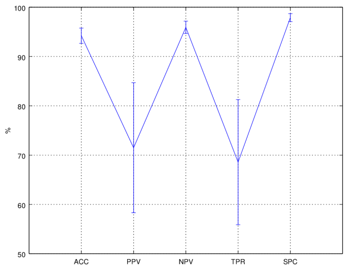
Table 6.2 is a summary showing how each group of neural network performed on average in the performance evaluation. Figures 6.6 is a visual representation of this table. A comparison of the evaluation is illustrated in figure 6.7. As expected, an increase in performance can be observed as the hidden neurons in the networks are increased. Note that from 30 hidden neurons onwards, the performance increase becomes minimal.

Table 6.2: A summary of the performance of the different ANNs

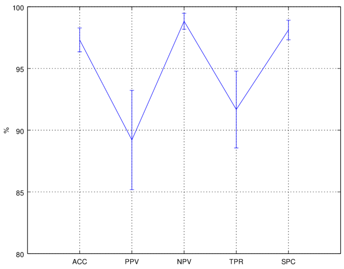
Neurons	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)
2	88.91 ± 1.16	18.07 ± 9.59	89.54 ± 1.06	16.62 ± 9.26	99.23 ± 1.57
5	94.22 ± 1.55	71.49 ± 13.18	95.91 ± 1.26	68.58 ± 12.69	97.89 ± 0.81
10	97.31 ± 0.97	89.21 ± 4.02	98.83 ± 0.65	91.67 ± 3.11	98.12 ± 0.79
15	97.65 ± 0.68	91.29 ± 3.62	98.69 ± 0.67	90.66 ± 3.20	98.65 ± 0.36
20	97.93 ± 0.74	92.87 ± 2.47	98.74 ± 0.81	91.03 ± 4.81	98.91 ± 0.18
30	98.06 ± 0.38	94.19 ± 2.16	98.66 ± 0.54	90.38 ± 2.10	99.15 ± 0.15
40	98.01 ± 0.62	95.05 ± 1.90	98.47 ± 0.77	88.97 ± 4.44	99.30 ± 0.12
60	98.17 ± 0.43	95.14 ± 0.77	98.63 ± 0.73	90.12 ± 3.92	99.32 ± 0.01



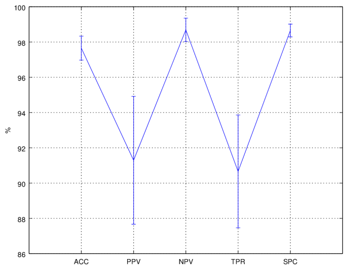
(a) 2 neurons in hidden layer



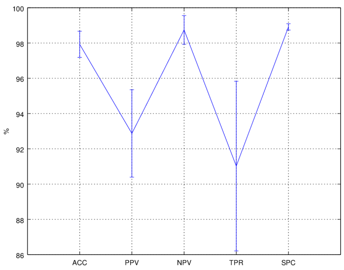
(b) 5 neurons in hidden layer



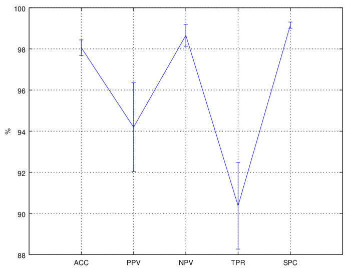
(c) 10 neurons in hidden layer



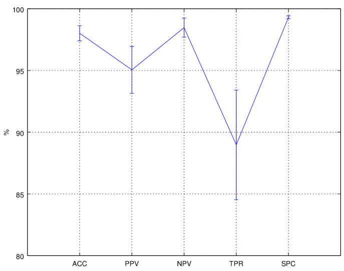
(d) 15 neurons in hidden layer



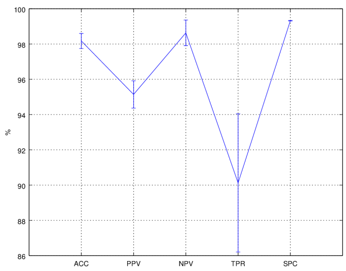
(e) 20 neurons in hidden layer



(f) 30 neurons in hidden layer



(g) 40 neurons in hidden layer



(h) 60 neurons in hidden layer

Figure 6.6: Means of performance measures for each group of neural networks.

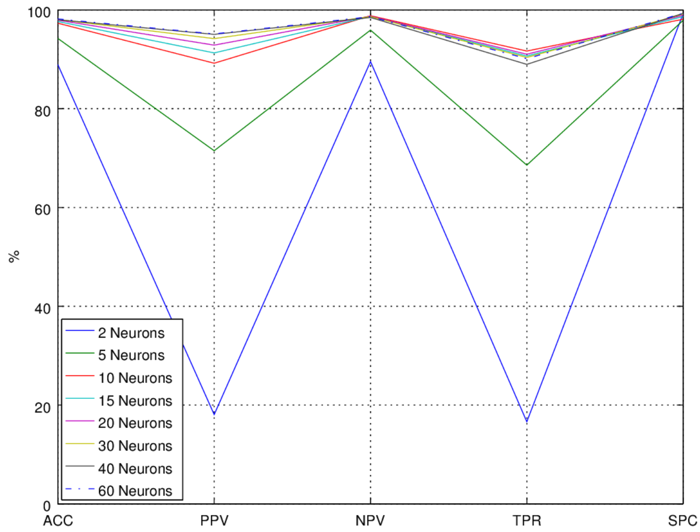


Figure 6.7: Mean performance comparison

Table 6.3 shows the evaluation results for 2 of the 80 test cases. Specifically test 2 from the 10 hidden neuron group, and test 7 from the 40 hidden neuron group of networks. Here one can see that these specific networks are better at identifying some objects than others. The final line of each sub-table shows the average of the performance measures for that that particular test case. For the complete results see appendix 6.2.

Table 6.3: Performance Measures

(a) 10 neurons in hidden layer, test 2

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)
1	100	685	15	0	98.12	86.96	100	100	97.86
2	100	693	7	0	99.12	93.46	100	100	99
3	91	679	21	9	96.25	81.25	98.69	91	97
4	87	680	20	13	95.88	81.31	98.12	87	97.14
5	100	689	11	0	98.62	90.09	100	100	98.43
6	88	680	20	12	96	81.48	98.27	88	97.14
7	95	671	29	5	95.75	76.61	99.26	95	95.86
8	86	696	4	14	97.75	95.56	98.03	86	99.43
Total:	747	5473	127	53	97.19	85.84	99.05	93.38	97.73

(b) 40 neurons in hidden layer, test 7

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)
1	87	692	8	13	97.38	91.58	98.16	87	98.86
2	100	669	31	0	96.12	76.34	100	100	95.57
3	88	700	0	12	98.5	100	98.31	88	100
4	88	696	4	12	98	95.65	98.31	88	99.43
5	100	700	0	0	100	100	100	100	100
6	75	700	0	25	96.88	100	96.55	75	100
7	81	692	8	19	96.62	91.01	97.33	81	98.86
8	84	660	40	16	93	67.74	97.63	84	94.29
Total:	703	5509	91	97	97.06	90.29	98.29	87.88	98.37

That then concludes, this, the second to last chapter of this document.

Part IV

Conclusion

Chapter 7

Conclusion

In chapter 1 the hypothesis:

> **It is possible, using ANNs, to classify, or recognise, 3D objects represented by point cloud data.**

was formulated.

In an attempt to prove or disprove whether it is possible to perform 3D object recognition with artificial neural networks the following was done during the course of this research:

Relevant literature was reviewed which suggested that neural networks are particularly well suited for pattern recognition type applications. An understanding of basic machine vision (camera theory) was gained. This knowledge was applied in order to develop a laser triangulation 3D scanner and object recognition software that relies on artificial neural networks. The use of global point feature histograms to reduce point clouds to a form more manageable for neural networks proved important.

Studying the validation results of the trained neural networks, it is clear that networks with 30 or more hidden neuron perform the task of object recognition very well. Once again the roll of the global point feature histogram needs to be stressed here. The overall impression of the results is that it is that the hypothesis is at the very least partly confirmed – Artificial neural networks can indeed be used to perform 3D object recognition, but the preprocessing of the data plays a major part in how successful these networks will be.

7.1 Pitfalls Encountered

7.1.1 Generating Training- and Testing Data

Images obtained from the simulated scanner took a long time to render. This was not a practical solution, but was a valuable exercise towards understanding laser scanning, and is still useful as an engineering tool for design and analysis of machine vision applications and testing of vision algorithms. PCL's built in virtual scanner was used instead of generating training and testing data from the Blender simulation. The PCL virtual scanner program has the advantage that it can generate point cloud data using the 3D CAD model as input, thus eliminating the time consuming rendering process.

7.2 Research Contribution

One contribution by this research is the use of 3D modelling and animation software to study, develop and test machine vision configurations and the required software without having to build potentially expensive prototypes. In this research Blender was used to do just that. A verification platform, an actual laser scanner, was then built and performed relatively well considering the low cost component that were used.

7.3 Future Work

1. The research in this dissertation only considered FFNNs for classifying non-complex 3D objects. Future research may consider using other types of classifiers, such as k-Nearest Neighbours, Support Vector Machines, more complex ANN architectures, and comparing the performance of these classifying techniques. Furthermore, the object recognition task could be tested on more complex geometries that are more representative of what might be encountered in a manufacturing environment. This would provide more definitive proof whether or not ANNs are applicable and reliable classifiers.

2. 3D scanners are becoming increasingly important within quality inspection systems. Existing metrology and inspection software, like GOM Inspect, provide the functionality of comparing 3D scan data to corresponding CAD data. Any dimensional deviations from the specified tolerances are reported, often as a colour plot. These reports require interpretation by a human expert in order to identify

the possible causes of the deviations. Building on the research presented in this dissertation, future work can be done to make metrology software more intelligent. Through the use of ANNs, inspection software might be made capable of classifying detected deviations into different error classes. These error classes could represent geometrical characteristics, like:

- Straightness
- Flatness
- Roundness
- Cylindricity
- Parallelism
- Perpendicularity
- Angularity
- Location or position
- Run-out

Bibliography

[1] H. Ford and S. Crowther. *My Life and Work*. Library of American civilization. Doubleday, Page & Company, 1922.

[2] Florian Viol. "An in-line measuring technique with tool error detection in an automotive production line". Masters dissertation. Port Elizabeth: Nelson Mandela Metropolitan University, Dec. 2010.

[3] J Padayachee, J Davrajh, and G Bright. "The development of reconfigurable manufacturing equipment for product mass customization". In: *International Conference on Competitive Manufacturing*. Ed. by Dimitri Dimitrov. Feb. 2010, pp. 291–296.

[4] Shaniel Davrajh and Glen Bright. "An automated apparatus for dynamic inspection of mass-produced custom parts". In: *Assembly Automation* 30.1 (2010), pp. 47–55.

[5] Tian Chen, Xiaoming Du, Ming Jia, et al. "Application of optical inspection and metrology in quality control for aircraft components". In: *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*. Vol. 5. IEEE. 2010, pp. V5–294.

[6] Maien Hamed. "Vision-guided tracking of complex three-dimensional seams for robotic gas metal arc welding". Masters dissertation. Port Elizabeth: Nelson Mandela Metropolitan University, Jan. 2011.

[7] Stefan Buys. "Genetic algorithm for artificial neural network training for the purpose of automated part recognition". Masters dissertation. Port Elizabeth: Nelson Mandela Metropolitan University, 2012.

[8] Alberto Tellaeche and Beatriz Robles. "3D machine vision and artificial neural networks for quality inspection in mass production pieces". In: *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*. IEEE. 2010, pp. 1–4.

[9] João L Vilaça, Jaime C Fonseca, and António M Pinho. "Non-contact 3D acquisition system based on stereo vision and laser triangulation". In: *Machine vision and applications* 21.3 (2010), pp. 341–350.

[10] Edmond Wai Yan So, Stefano Michieletto, and Emanuele Menegatti. "Calibration of a dual-laser triangulation system for assembly line completeness inspection". In: *Robotic and Sensors Environments (ROSE), 2012 IEEE International Symposium on*. IEEE. 2012, pp. 138–143.

[11] Jigar Senjalia, Parinda Pandya, and Harsh Kapadia. "Measurement of wheel alignment using Camera Calibration and Laser Triangulation". In: *Engineering (NUiCONE), 2013 Nirma University International Conference on*. IEEE. 2013, pp. 1–5.

[12] Paolo Bellandi, Franco Docchio, and Giovanna Sansoni. "Roboscan: a combined 2D and 3D vision system for improved speed and flexibility in pick-and-place operation". In: *The International Journal of Advanced Manufacturing Technology* 69.5-8 (2013), pp. 1873–1886.

[13] Leszek Jarzebowicz and Slawomir Judek. "3D machine vision system for inspection of contact strips in railway vehicle current collectors". In: *Applied Electronics (AE), 2014 International Conference on*. IEEE. 2014, pp. 139–144.

[14] Ming J Tsai and Nai-Jun Ann. "An automatic golf head robotic welding system using 3D machine vision system". In: *Advanced robotics and Its Social Impacts, 2008. ARSO 2008. IEEE Workshop on*. IEEE. 2008, pp. 1–6.

[15] Wei Huang and Radovan Kovacevic. "Development of a real-time laser-based machine vision system to monitor and control welding processes". In: *The International Journal of Advanced Manufacturing Technology* 63.1-4 (2012), pp. 235–248.

[16] Yu Huang, Yangliu Xiao, Pingjiang Wang, et al. "A seam-tracking laser welding platform with 3D and 2D visual information fusion vision sensor system". In: *The International Journal of Advanced Manufacturing Technology* 67.1-4 (2013), pp. 415–426.

[17] OpenCV. *Camera Calibration and 3D Reconstruction*. URL: `http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html` (visited on 12/16/2016).

[18] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008. ISBN: 9780596554040.

[19] Zhengyou Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on pattern analysis and machine intelligence* 22.11 (2000), pp. 1330–1334.

[20] Duane C Brown. "Close-range camera calibration". In: *Photogrammetric Engineering* 37.8 (1971), pp. 855–866.

[21] Vaibhav Bajpai and Vladislav Perelman. "A Cross-Platform Open Source 3D Object Reconstruction System using a Laser Line Projector." In: *IEEE German Student Conference 2012*. 2012.

[22] Jincheng Yu, Kaijian Weng, Guoyuan Liang, et al. "A vision-based robotic grasping system using deep learning for 3D object recognition and pose estimation". In: *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*. IEEE. 2013, pp. 1175–1180.

[23] S Hamidreza Kasaei, Miguel Oliveira, Gi Hyun Lim, et al. "An interactive open-ended learning approach for 3d object recognition". In: *Autonomous Robot Systems and Competitions (ICARSC), 2014 IEEE International Conference on*. IEEE. 2014, pp. 47–52.

[24] Dong Liang, Kaijian Weng, Can Wang, et al. "A 3D object recognition and pose estimation system using deep learning method". In: *Information Science and Technology (ICIST), 2014 4th IEEE International Conference on*. IEEE. 2014, pp. 401–404.

[25] Ren C Luo, Chia-Wen Kuo, and Yi-Ting Chung. "Model-based 3D object recognition and fetching by a 7-DoF robot with online obstacle avoidance for factory automation". In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 2647–2652.

[26] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, et al. "Fast 3d recognition and pose using the viewpoint feature histogram". In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE. 2010, pp. 2155–2162.

[27] Aitor Aldoma, Markus Vincze, Nico Blodow, et al. "CAD-model recognition and 6DOF pose estimation using 3D cues". In: *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE. 2011, pp. 585–592.

[28] Aitor Aldoma, Federico Tombari, Radu Bogdan Rusu, et al. "OUR-CVFH–oriented, unique and repeatable clustered viewpoint feature histogram for object recognition and 6DOF pose estimation". In: *Joint DAGM (German Association for Pattern Recognition) and OAGM Symposium*. Springer. 2012, pp. 113–122.

[29] Walter Wohlkinger and Markus Vincze. "Ensemble of shape functions for 3d object classification". In: *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2987–2992.

[30] Guan Pang and Ulrich Neumann. "Fast and Robust Multi-view 3D Object Recognition in Point Clouds". In: *3D Vision (3DV), 2015 International Conference on*. IEEE. 2015, pp. 171–179.

[31] Jianhua Wang, Jinjin Lu, Weihai Chen, et al. "Convolutional neural network for 3D object recognition based on RGB-D dataset". In: *Industrial Electronics and Applications (ICIEA), 2015 IEEE 10th Conference on*. IEEE. 2015, pp. 34–39.

[32] Alberto Garcia-Garcia, Sergio Orts-Escolano, Jose Garcia-Rodriguez, et al. "Interactive 3D object recognition pipeline on mobile GPGPU computing plat-

forms using low-cost RGB-D sensors”. In: *Journal of Real-Time Image Processing* (2016), pp. 1–20.

[33] Andrea Frome, Daniel Huber, Ravi Kolluri, et al. “Recognizing objects in range data using regional point descriptors”. In: *European conference on computer vision*. Springer. 2004, pp. 224–237.

[34] Federico Tombari, Samuele Salti, and Luigi Di Stefano. “Unique shape context for 3D data description”. In: *Proceedings of the ACM workshop on 3D object retrieval*. ACM. 2010, pp. 57–62.

[35] Federico Tombari, Samuele Salti, and Luigi Di Stefano. “A combined texture-shape descriptor for enhanced 3D feature matching”. In: *Image Processing (ICIP), 2011 18th IEEE International Conference on*. IEEE. 2011, pp. 809–812.

[36] Yulan Guo, Ferdous Sohel, Mohammed Bennamoun, et al. “Rotational projection statistics for 3D local surface description and object recognition”. In: *International journal of computer vision* 105.1 (2013), pp. 63–86.

[37] Andries P Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[38] Blender Foundation. *About - blender.org - Home of the Blender project - Free and Open 3D Creation Software*. URL: `https://www.blender.org/about/` (visited on 12/16/2016).

[39] Chris Roman, Gabrielle Inglis, and James Rutter. “Application of structured light imaging for high resolution mapping of underwater archaeological sites”. In: *OCEANS 2010 IEEE-Sydney*. IEEE. 2010, pp. 1–9.

[40] Marius Muja, Radu Bogdan Rusu, Gary Bradski, et al. “Rein-a fast, robust, scalable recognition infrastructure”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2939–2946.

[41] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. “Fast point feature histograms (FPFH) for 3D registration”. In: *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on*. IEEE. 2009, pp. 3212–3217.

[42] Radu Bogdan Rusu, Jan Bandouch, Franziska Meier, et al. “Human action recognition using global point feature histograms and action shapes”. In: *Advanced Robotics* 23.14 (2009), pp. 1873–1908.

[43] Marina Sokolova and Guy Lapalme. “A systematic analysis of performance measures for classification tasks”. In: *Information Processing & Management* 45.4 (2009), pp. 427–437.

Appendices

Appendix A

Results: ANN Training and Validation

A.1 2 Hidden Neurons

Table A.1: Summary of training and validation results for each test case.

(a) Test 1.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	0	700	0	100	87.5	0	87.5	0	100	-	-	-
2	0	700	0	100	87.5	0	87.5	0	100	-	-	-
3	0	700	0	100	87.5	0	87.5	0	100	-	-	-
4	0	700	0	100	87.5	0	87.5	0	100	-	-	-
5	100	618	82	0	89.75	54.95	100	100	88.29	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	0	700	0	100	87.5	0	87.5	0	100	-	-	-
8	0	700	0	100	87.5	0	87.5	0	100	-	-	-
Total:	100	5518	82	700	87.78	6.868	89.06	12.5	98.54	0.09627	56	0.06559

(b) Test 2.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	0	700	0	100	87.5	0	87.5	0	100	-	-	-
2	78	664	36	22	92.75	68.42	96.79	78	94.86	-	-	-
3	0	700	0	100	87.5	0	87.5	0	100	-	-	-
4	95	688	12	5	97.88	88.79	99.28	95	98.29	-	-	-
5	91	700	0	9	98.88	100	98.73	91	100	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	0	700	0	100	87.5	0	87.5	0	100	-	-	-
8	0	700	0	100	87.5	0	87.5	0	100	-	-	-
Total:	264	5552	48	536	90.88	32.15	91.54	33	99.14	0.06815	68	0.08639

(c) Test 3.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	0	700	0	100	87.5	0	87.5	0	100	-	-	-
2	0	700	0	100	87.5	0	87.5	0	100	-	-	-
3	1	700	0	99	87.62	100	87.61	1	100	-	-	-
4	0	700	0	100	87.5	0	87.5	0	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	0	700	0	100	87.5	0	87.5	0	100	-	-	-
8	0	700	0	100	87.5	0	87.5	0	100	-	-	-
Total:	101	5600	0	699	89.08	25	89.08	12.62	100	0.08885	2708	3.069

(d) Test 4.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	0	700	0	100	87.5	0	87.5	0	100	-	-	-
2	0	700	0	100	87.5	0	87.5	0	100	-	-	-
3	0	700	0	100	87.5	0	87.5	0	100	-	-	-
4	59	700	0	41	94.88	100	94.47	59	100	-	-	-
5	0	700	0	100	87.5	0	87.5	0	100	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	0	700	0	100	87.5	0	87.5	0	100	-	-	-
8	0	700	0	100	87.5	0	87.5	0	100	-	-	-
Total:	59	5600	0	741	88.42	12.5	88.37	7.375	100	0.08146	194	0.2349

(e) Test 5.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	0	700	0	100	87.5	0	87.5	0	100	-	-	-
2	0	700	0	100	87.5	0	87.5	0	100	-	-	-
3	0	700	0	100	87.5	0	87.5	0	100	-	-	-
4	100	674	26	0	96.75	79.37	100	100	96.29	-	-	-
5	0	700	0	100	87.5	0	87.5	0	100	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	0	700	0	100	87.5	0	87.5	0	100	-	-	-
8	0	700	0	100	87.5	0	87.5	0	100	-	-	-
Total:	100	5574	26	700	88.66	9.921	89.06	12.5	99.54	0.08088	7606	8.691

Continued on next page

Table A.1: – continued from previous page

(f) Test 6.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	0	700	0	100	87.5	0	87.5	0	100	-	-	-
2	0	700	0	100	87.5	0	87.5	0	100	-	-	-
3	84	683	17	16	95.88	83.17	97.71	84	97.57	-	-	-
4	0	700	0	100	87.5	0	87.5	0	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	0	700	0	100	87.5	0	87.5	0	100	-	-	-
8	0	700	0	100	87.5	0	87.5	0	100	-	-	-
Total:	184	5583	17	616	90.11	22.9	90.34	23	99.7	0.08106	2240	2.575

(g) Test 7.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	0	700	0	100	87.5	0	87.5	0	100	-	-	-
2	85	683	17	15	96	83.33	97.85	85	97.57	-	-	-
3	0	700	0	100	87.5	0	87.5	0	100	-	-	-
4	0	700	0	100	87.5	0	87.5	0	100	-	-	-
5	89	700	0	11	98.62	100	98.45	89	100	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	0	700	0	100	87.5	0	87.5	0	100	-	-	-
8	0	700	0	100	87.5	0	87.5	0	100	-	-	-
Total:	174	5583	17	626	89.95	22.92	90.16	21.75	99.7	0.07968	4480	5.109

(h) Test 8.

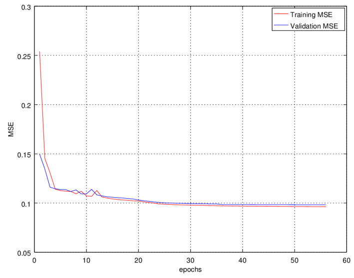
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	0	700	0	100	87.5	0	87.5	0	100	-	-	-
2	87	569	131	13	82	39.91	97.77	87	81.29	-	-	-
3	0	700	0	100	87.5	0	87.5	0	100	-	-	-
4	30	700	0	70	91.25	100	90.91	30	100	-	-	-
5	100	591	109	0	86.38	47.85	100	100	84.43	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	0	700	0	100	87.5	0	87.5	0	100	-	-	-
8	0	700	0	100	87.5	0	87.5	0	100	-	-	-
Total:	217	5360	240	583	87.14	23.47	90.77	27.12	95.71	0.0832	1119	1.282

(i) Test 9.

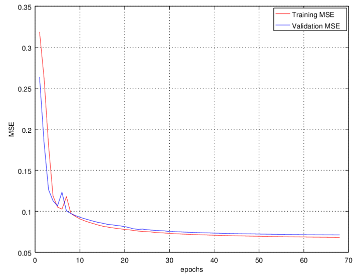
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	0	700	0	100	87.5	0	87.5	0	100	-	-	-
2	0	700	0	100	87.5	0	87.5	0	100	-	-	-
3	0	700	0	100	87.5	0	87.5	0	100	-	-	-
4	100	700	0	0	100	100	100	100	100	-	-	-
5	0	700	0	100	87.5	0	87.5	0	100	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	31	700	0	69	91.38	100	91.03	31	100	-	-	-
8	0	700	0	100	87.5	0	87.5	0	100	-	-	-
Total:	131	5600	0	669	89.55	25	89.5	16.38	100	0.08226	1845	2.11

(j) Test 10.

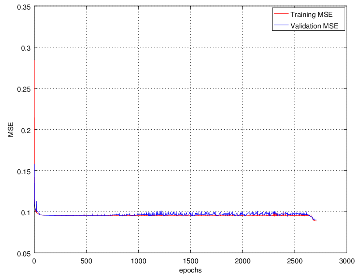
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	0	700	0	100	87.5	0	87.5	0	100	-	-	-
2	0	700	0	100	87.5	0	87.5	0	100	-	-	-
3	0	700	0	100	87.5	0	87.5	0	100	-	-	-
4	0	700	0	100	87.5	0	87.5	0	100	-	-	-
5	0	700	0	100	87.5	0	87.5	0	100	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	0	700	0	100	87.5	0	87.5	0	100	-	-	-
8	0	700	0	100	87.5	0	87.5	0	100	-	-	-
Total:	0	5600	0	800	87.5	0	87.5	0	100	0.09571	45	0.05353



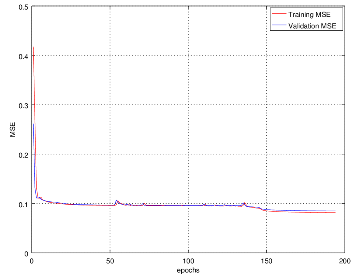
(a) Test 1.



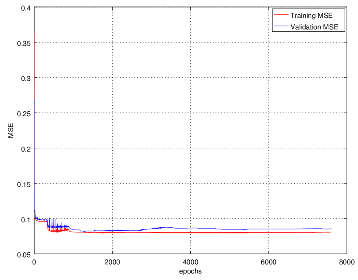
(b) Test 2.



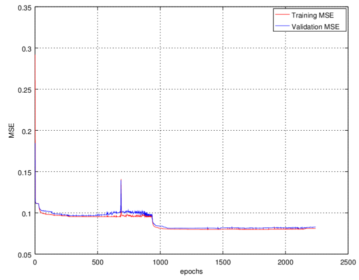
(c) Test 3.



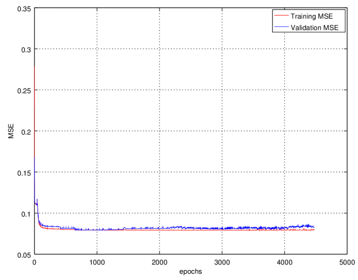
(d) Test 4.



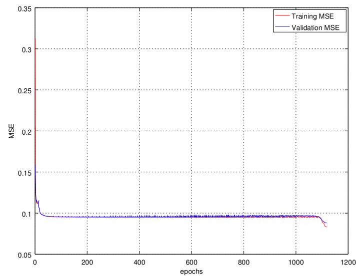
(e) Test 5.



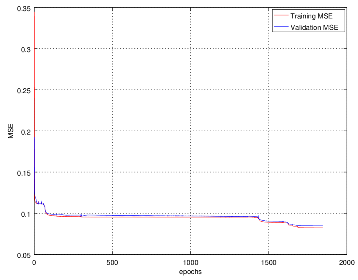
(f) Test 6.



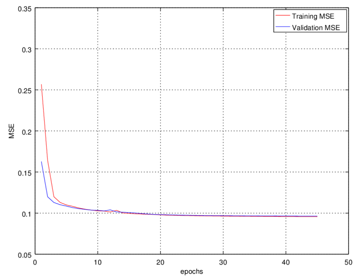
(g) Test 7.



(h) Test 8.

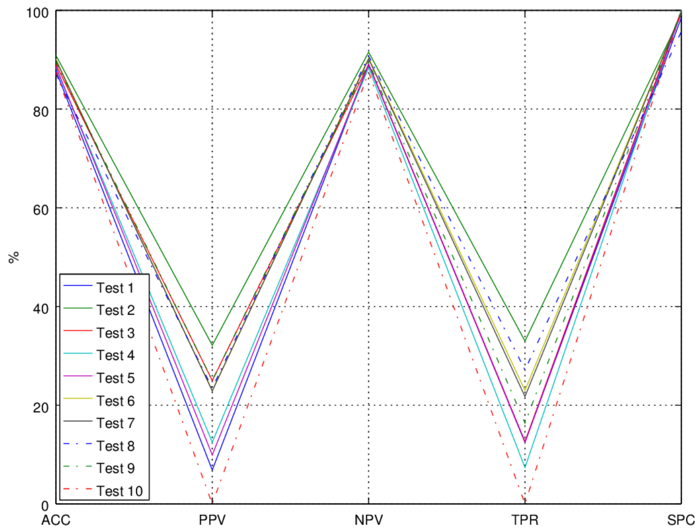


(i) Test 9.

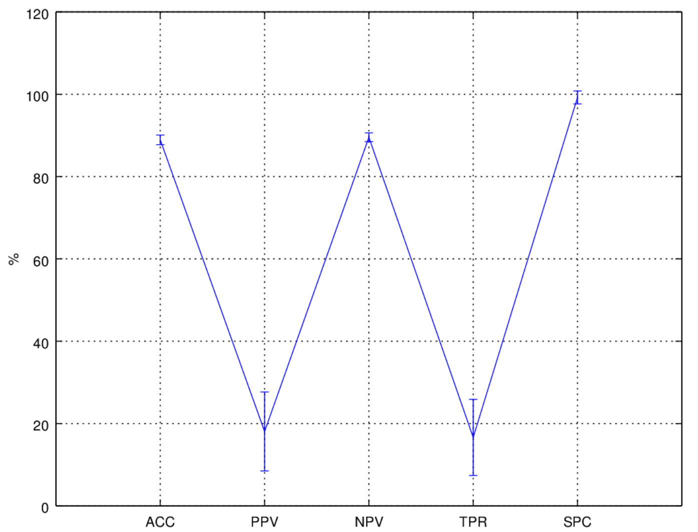


(j) Test 10.

Figure A.1: MSE vs Epochs: Mean Squared Error during the training of each test case.



(a) Comparative graph: Performance for each test case.



(b) Mean performance with error bars indicating standard deviation.

Figure A.2: Visualisation of recognition performance of ANNs with 2 hidden neurons.

Table A.2: Summary of the mean performance.

ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	Epochs	Training Time (s)
88.91 ± 1.16	18.07 ± 9.59	89.54 ± 1.06	16.62 ± 9.26	99.23 ± 1.57	2036.10 ± 2313.42	2.3277 ± 2.6396

A.2 5 Hidden Neurons

Table A.3: Summary of training and validation results for each test case.

(a) Test 1.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	99	691	9	1	98.75	91.67	99.86	99	98.71	-	-	-
2	100	690	10	0	98.75	90.91	100	100	98.57	-	-	-
3	32	686	14	68	89.75	69.57	90.98	32	98	-	-	-
4	94	700	0	6	99.25	100	99.15	94	100	-	-	-
5	91	682	18	9	96.62	83.49	98.7	91	97.43	-	-	-
6	95	693	7	5	98.5	93.14	99.28	95	99	-	-	-
7	42	699	1	58	92.62	97.67	92.34	42	99.86	-	-	-
8	100	577	123	0	84.62	44.84	100	100	82.43	-	-	-
Total:	653	5418	182	147	94.86	83.91	97.54	81.62	96.75	0.00612	6739	12.88

(b) Test 2.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	0	688	12	100	86	0	87.31	0	98.29	-	-	-
2	99	699	1	1	99.75	99	99.86	99	99.86	-	-	-
3	12	700	0	88	89	100	88.83	12	100	-	-	-
4	79	700	0	21	97.38	100	97.09	79	100	-	-	-
5	99	679	21	1	97.25	82.5	99.85	99	97	-	-	-
6	91	684	16	9	96.88	85.05	98.7	91	97.71	-	-	-
7	87	660	40	13	93.38	68.5	98.07	87	94.29	-	-	-
8	78	700	0	22	97.25	100	96.95	78	100	-	-	-
Total:	545	5510	90	255	94.61	79.38	95.83	68.12	98.39	0.02078	1248	2.422

(c) Test 3.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	6	696	4	94	87.75	60	88.1	6	99.43	-	-	-
2	100	665	35	0	95.62	74.07	100	100	95	-	-	-
3	79	684	16	21	95.38	83.16	97.02	79	97.71	-	-	-
4	87	697	3	13	98	96.67	98.17	87	99.57	-	-	-
5	100	688	12	0	98.5	89.29	100	100	98.29	-	-	-
6	0	697	3	100	87.12	0	87.45	0	99.57	-	-	-
7	3	677	23	97	85	11.54	87.47	3	96.71	-	-	-
8	48	663	37	52	88.88	56.47	92.73	48	94.71	-	-	-
Total:	423	5467	133	377	92.03	58.9	93.87	52.88	97.62	0.03521	774	1.494

(d) Test 4.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	59	690	10	41	93.62	85.51	94.39	59	98.57	-	-	-
2	59	696	4	41	94.38	93.65	94.44	59	99.43	-	-	-
3	90	651	49	10	92.62	64.75	98.49	90	93	-	-	-
4	96	689	11	4	98.12	89.72	99.42	96	98.43	-	-	-
5	94	700	0	6	99.25	100	99.15	94	100	-	-	-
6	0	698	2	100	87.25	0	87.47	0	99.71	-	-	-
7	0	698	2	100	87.25	0	87.47	0	99.71	-	-	-
8	92	691	9	8	97.88	91.09	98.86	92	98.71	-	-	-
Total:	490	5513	87	310	93.8	65.59	94.96	61.25	98.45	0.02263	1.998e+04	38.25

(e) Test 5.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	59	700	0	41	94.88	100	94.47	59	100	-	-	-
2	100	688	12	0	98.5	89.29	100	100	98.29	-	-	-
3	99	656	44	1	94.38	69.23	99.85	99	93.71	-	-	-
4	100	696	4	0	99.5	96.15	100	100	99.43	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	0	700	0	100	87.5	0	87.5	0	100	-	-	-
8	93	700	0	7	99.12	100	99.01	93	100	-	-	-
Total:	551	5540	60	249	95.17	69.33	96.04	68.88	98.93	0.0163	7530	14.41

Continued on next page

Table A.3: – continued from previous page

(f) Test 6.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	58	697	3	42	94.38	95.08	94.32	58	99.57	-	-	-
2	85	698	2	15	97.88	97.7	97.9	85	99.71	-	-	-
3	95	688	12	5	97.88	88.79	99.28	95	98.29	-	-	-
4	82	694	6	18	97	93.18	97.47	82	99.14	-	-	-
5	97	666	34	3	95.38	74.05	99.55	97	95.14	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	0	700	0	100	87.5	0	87.5	0	100	-	-	-
8	46	700	0	54	93.25	100	92.84	46	100	-	-	-
Total:	463	5543	57	337	93.84	68.6	94.54	57.88	98.98	0.02487	1.394e+04	26.29

(g) Test 7.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	51	700	0	49	93.88	100	93.46	51	100	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	48	691	9	52	92.38	84.21	93	48	98.71	-	-	-
4	94	699	1	6	99.12	98.95	99.15	94	99.86	-	-	-
5	100	664	36	0	95.5	73.53	100	100	94.86	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	99	558	142	1	82.12	41.08	99.82	99	79.71	-	-	-
8	94	671	29	6	95.62	76.42	99.11	94	95.86	-	-	-
Total:	586	5383	217	214	93.27	71.77	96.51	73.25	96.12	0.01205	1.764e+04	33.26

(h) Test 8.

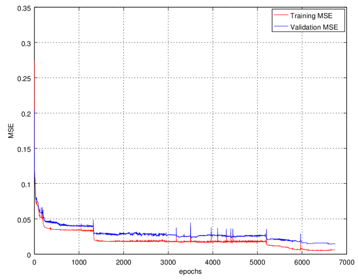
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	46	698	2	54	93	95.83	92.82	46	99.71	-	-	-
2	100	690	10	0	98.75	90.91	100	100	98.57	-	-	-
3	97	682	18	3	97.38	84.35	99.56	97	97.43	-	-	-
4	89	688	12	11	97.12	88.12	98.43	89	98.29	-	-	-
5	100	656	44	0	94.5	69.44	100	100	93.71	-	-	-
6	84	698	2	16	97.75	97.67	97.76	84	99.71	-	-	-
7	58	681	19	42	92.38	75.32	94.19	58	97.29	-	-	-
8	91	651	49	9	92.75	65	98.64	91	93	-	-	-
Total:	665	5444	156	135	95.45	83.33	97.67	83.12	97.21	0.00429	1600	3.05

(i) Test 9.

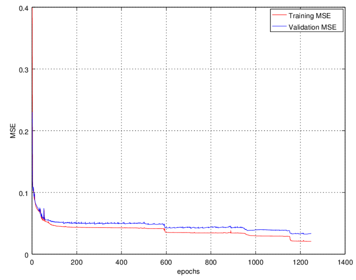
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	0	700	0	100	87.5	0	87.5	0	100	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	0	700	0	100	87.5	0	87.5	0	100	-	-	-
4	99	700	0	1	99.88	100	99.86	99	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	0	700	0	100	87.5	0	87.5	0	100	-	-	-
7	97	586	114	3	85.38	45.97	99.49	97	83.71	-	-	-
8	0	700	0	100	87.5	0	87.5	0	100	-	-	-
Total:	396	5486	114	404	91.91	43.25	93.67	49.5	97.96	0.04803	982	1.894

(j) Test 10.

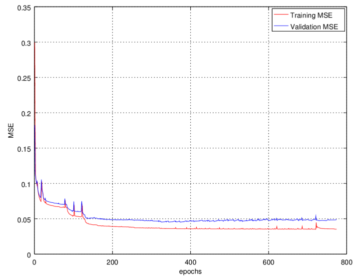
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	52	700	0	48	94	100	93.58	52	100	-	-	-
2	78	654	46	22	91.5	62.9	96.75	78	93.43	-	-	-
3	96	697	3	4	99.12	96.97	99.43	96	99.57	-	-	-
4	97	678	22	3	96.88	81.51	99.56	97	96.86	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	92	696	4	8	98.5	95.83	98.86	92	99.43	-	-	-
7	99	692	8	1	98.88	92.52	99.86	99	98.86	-	-	-
8	100	697	3	0	99.62	97.09	100	100	99.57	-	-	-
Total:	714	5514	86	86	97.31	90.85	98.5	89.25	98.46	0.005115	1.029e+04	19.53



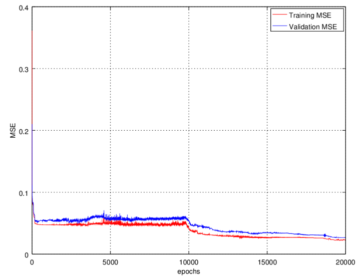
(a) Test 1.



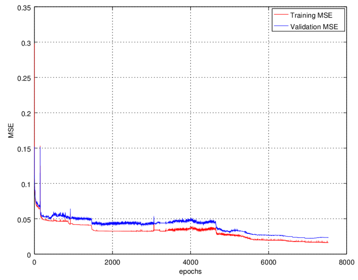
(b) Test 2.



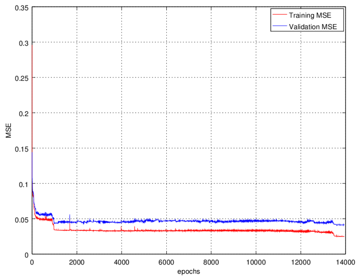
(c) Test 3.



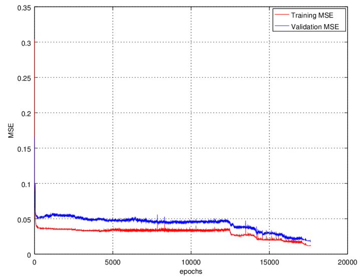
(d) Test 4.



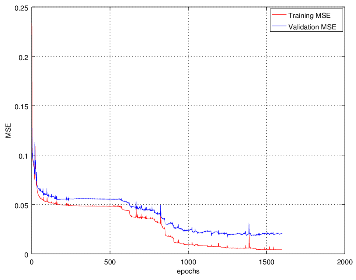
(e) Test 5.



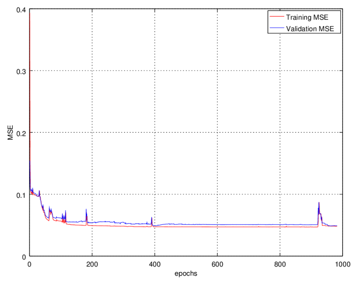
(f) Test 6.



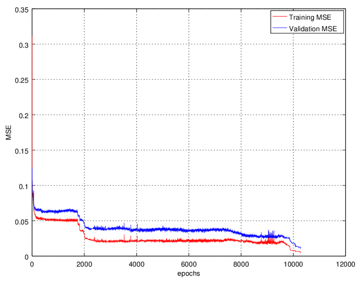
(g) Test 7.



(h) Test 8.

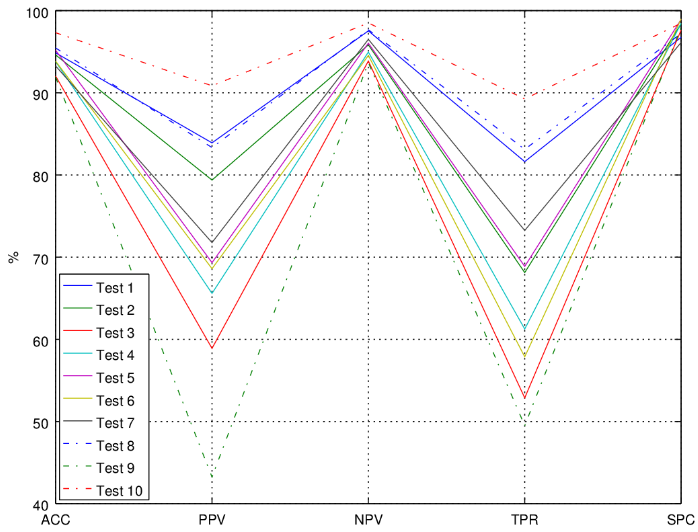


(i) Test 9.

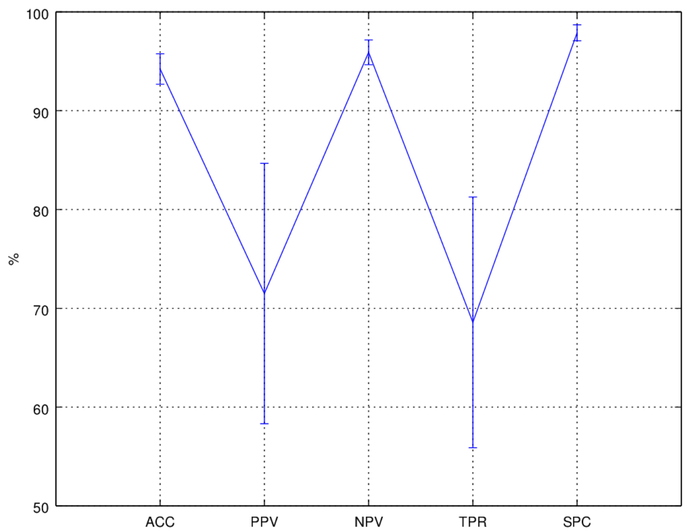


(j) Test 10.

Figure A.3: MSE vs Epochs: Mean Squared Error during the training of each test case.



(a) Comparative graph: Performance for each test case.



(b) Mean performance with error bars indicating standard deviation.

Figure A.4: Visualisation of recognition performance of ANNs with 5 hidden neurons.

Table A.4: Summary of the mean performance.

ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	Epochs	Training Time (s)
94.22 ± 1.55	71.49 ± 13.18	95.91 ± 1.26	68.58 ± 12.69	97.89 ± 0.81	8072.10 ± 6831.11	15.3485 ± 12.9652

A.3 10 Hidden Neurons

Table A.5: Summary of training and validation results for each test case.

(a) Test 1.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	92	689	11	8	97.62	89.32	98.85	92	98.43	-	-	-
2	100	658	42	0	94.75	70.42	100	100	94	-	-	-
3	87	694	6	13	97.62	93.55	98.16	87	99.14	-	-	-
4	98	694	6	2	99	94.23	99.71	98	99.14	-	-	-
5	100	697	3	0	99.62	97.09	100	100	99.57	-	-	-
6	84	688	12	16	96.5	87.5	97.73	84	98.29	-	-	-
7	95	686	14	5	97.62	87.16	99.28	95	98	-	-	-
8	91	700	0	9	98.88	100	98.73	91	100	-	-	-
Total:	747	5506	94	53	97.7	89.91	99.06	93.38	98.32	0.002071	241	0.7685

(b) Test 2.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	100	685	15	0	98.12	86.96	100	100	97.86	-	-	-
2	100	693	7	0	99.12	93.46	100	100	99	-	-	-
3	91	679	21	9	96.25	81.25	98.69	91	97	-	-	-
4	87	680	20	13	95.88	81.31	98.12	87	97.14	-	-	-
5	100	689	11	0	98.62	90.09	100	100	98.43	-	-	-
6	88	680	20	12	96	81.48	98.27	88	97.14	-	-	-
7	95	671	29	5	95.75	76.61	99.26	95	95.86	-	-	-
8	86	696	4	14	97.75	95.56	98.03	86	99.43	-	-	-
Total:	747	5473	127	53	97.19	85.84	99.05	93.38	97.73	9.231e-05	2814	8.839

(c) Test 3.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	93	690	10	7	97.88	90.29	99	93	98.57	-	-	-
2	80	679	21	20	94.88	79.21	97.14	80	97	-	-	-
3	98	698	2	2	99.5	98	99.71	98	99.71	-	-	-
4	100	698	2	0	99.75	98.04	100	100	99.71	-	-	-
5	89	700	0	11	98.62	100	98.45	89	100	-	-	-
6	93	690	10	7	97.88	90.29	99	93	98.57	-	-	-
7	53	700	0	47	94.12	100	93.71	53	100	-	-	-
8	91	693	7	9	98	92.86	98.72	91	99	-	-	-
Total:	697	5548	52	103	97.58	93.59	98.22	87.12	99.07	0.00193	328	1.044

(d) Test 4.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	46	696	4	54	92.75	92	92.8	46	99.43	-	-	-
2	81	700	0	19	97.62	100	97.36	81	100	-	-	-
3	99	678	22	1	97.12	81.82	99.85	99	96.86	-	-	-
4	95	700	0	5	99.38	100	99.29	95	100	-	-	-
5	93	690	10	7	97.88	90.29	99	93	98.57	-	-	-
6	89	639	61	11	91	59.33	98.31	89	91.29	-	-	-
7	88	694	6	12	97.75	93.62	98.3	88	99.14	-	-	-
8	100	669	31	0	96.12	76.34	100	100	95.57	-	-	-
Total:	691	5466	134	109	96.2	86.67	98.11	86.38	97.61	0.00221	540	1.706

(e) Test 5.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	90	692	8	10	97.75	91.84	98.58	90	98.86	-	-	-
2	100	676	24	0	97	80.65	100	100	96.57	-	-	-
3	99	698	2	1	99.62	98.02	99.86	99	99.71	-	-	-
4	100	686	14	0	98.25	87.72	100	100	98	-	-	-
5	100	693	7	0	99.12	93.46	100	100	99	-	-	-
6	80	699	1	20	97.38	98.77	97.22	80	99.86	-	-	-
7	95	681	19	5	97	83.33	99.27	95	97.29	-	-	-
8	95	699	1	5	99.25	98.96	99.29	95	99.86	-	-	-
Total:	759	5524	76	41	98.17	91.59	99.28	94.88	98.64	0.001471	764	2.411

Continued on next page

Table A.5: – continued from previous page

(f) Test 6.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	84	700	0	16	98	100	97.77	84	100	-	-	-
2	100	695	5	0	99.38	95.24	100	100	99.29	-	-	-
3	98	682	18	2	97.5	84.48	99.71	98	97.43	-	-	-
4	99	695	5	1	99.25	95.19	99.86	99	99.29	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	94	693	7	6	98.38	93.07	99.14	94	99	-	-	-
7	95	689	11	5	98	89.62	99.28	95	98.43	-	-	-
8	99	685	15	1	98	86.84	99.85	99	97.86	-	-	-
Total:	769	5539	61	31	98.56	93.06	99.45	96.12	98.91	0.0004649	1190	3.782

(g) Test 7.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	77	695	5	23	96.5	93.9	96.8	77	99.29	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	93	668	32	7	95.12	74.4	98.96	93	95.43	-	-	-
4	94	700	0	6	99.25	100	99.15	94	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	92	688	12	8	97.5	88.46	98.85	92	98.29	-	-	-
7	88	693	7	12	97.62	92.63	98.3	88	99	-	-	-
8	97	699	1	3	99.5	98.98	99.57	97	99.86	-	-	-
Total:	741	5543	57	59	98.19	93.55	98.95	92.62	98.98	0.004332	196	0.629

(h) Test 8.

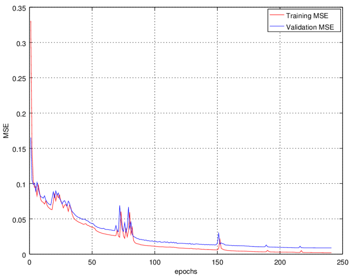
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	53	700	0	47	94.12	100	93.71	53	100	-	-	-
2	100	641	59	0	92.62	62.89	100	100	91.57	-	-	-
3	99	655	45	1	94.25	68.75	99.85	99	93.57	-	-	-
4	100	700	0	0	100	100	100	100	100	-	-	-
5	100	693	7	0	99.12	93.46	100	100	99	-	-	-
6	68	694	6	32	95.25	91.89	95.59	68	99.14	-	-	-
7	93	689	11	7	97.75	89.42	98.99	93	98.43	-	-	-
8	99	699	1	1	99.75	99	99.86	99	99.86	-	-	-
Total:	712	5471	129	88	96.61	88.18	98.5	89	97.7	0.001717	578	1.826

(i) Test 9.

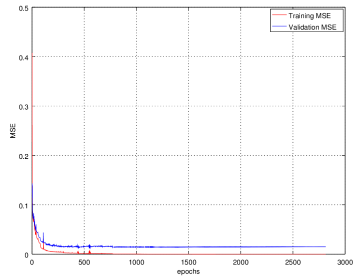
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	97	658	42	3	94.38	69.78	99.55	97	94	-	-	-
2	100	699	1	0	99.88	99.01	100	100	99.86	-	-	-
3	83	679	21	17	95.25	79.81	97.56	83	97	-	-	-
4	100	696	4	0	99.5	96.15	100	100	99.43	-	-	-
5	91	699	1	9	98.75	98.91	98.73	91	99.86	-	-	-
6	96	683	17	4	97.38	84.96	99.42	96	97.57	-	-	-
7	84	696	4	16	97.5	95.45	97.75	84	99.43	-	-	-
8	96	693	7	4	98.62	93.2	99.43	96	99	-	-	-
Total:	747	5503	97	53	97.66	89.66	99.05	93.38	98.27	0.0004399	667	2.106

(j) Test 10.

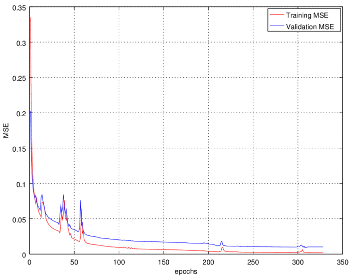
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	81	653	47	19	91.75	63.28	97.17	81	93.29	-	-	-
2	78	676	24	22	94.25	76.47	96.85	78	96.57	-	-	-
3	92	595	105	8	85.88	46.7	98.67	92	85	-	-	-
4	100	697	3	0	99.62	97.09	100	100	99.57	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	93	673	27	7	95.75	77.5	98.97	93	96.14	-	-	-
7	92	694	6	8	98.25	93.88	98.86	92	99.14	-	-	-
8	88	685	15	12	96.62	85.44	98.28	88	97.86	-	-	-
Total:	724	5373	227	76	95.27	80.04	98.6	90.5	95.95	0.002927	620	1.961



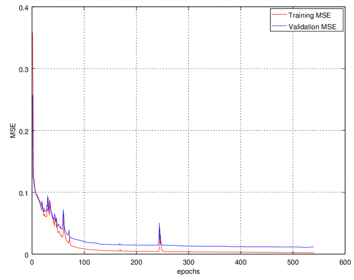
(a) Test 1.



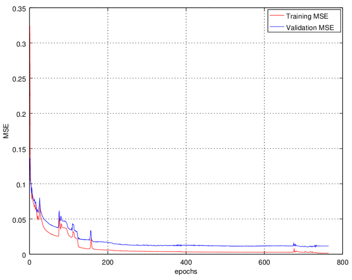
(b) Test 2.



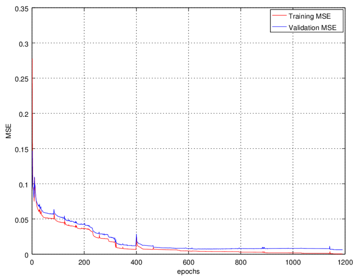
(c) Test 3.



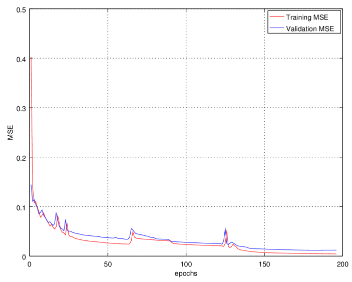
(d) Test 4.



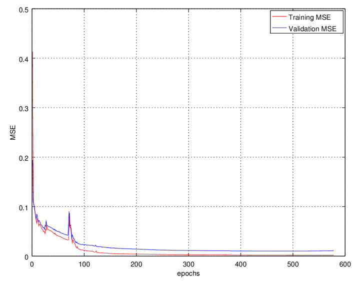
(e) Test 5.



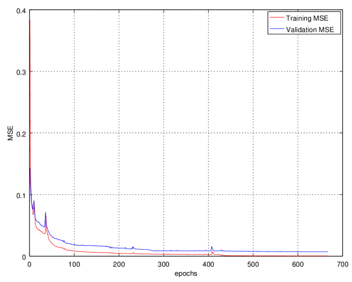
(f) Test 6.



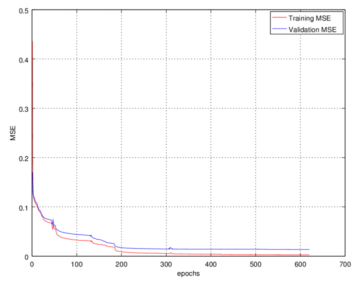
(g) Test 7.



(h) Test 8.

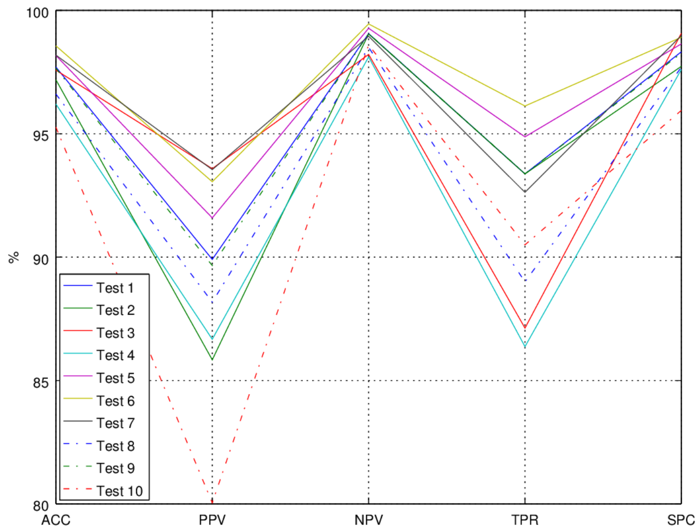


(i) Test 9.

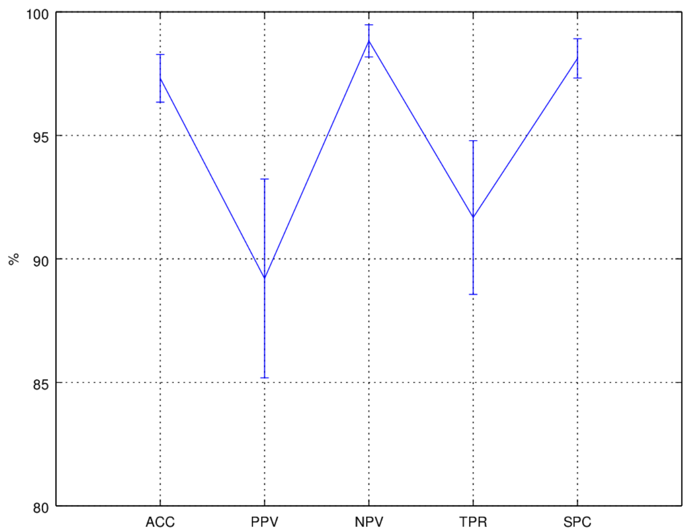


(j) Test 10.

Figure A.5: MSE vs Epochs: Mean Squared Error during the training of each test case.



(a) Comparative graph: Performance for each test case.



(b) Mean performance with error bars indicating standard deviation.

Figure A.6: Visualisation of recognition performance of ANNs with 10 hidden neurons.

Table A.6: Summary of the mean performance.

ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	Epochs	Training Time (s)
97.31 ± 0.97	89.21 ± 4.02	98.83 ± 0.65	91.67 ± 3.11	98.12 ± 0.79	793.80 ± 726.38	2.5072 ± 2.2801

A.4 15 Hidden Neurons

Table A.7: Summary of training and validation results for each test case.

(a) Test 1.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	95	669	31	5	95.5	75.4	99.26	95	95.57	-	-	-
2	100	658	42	0	94.75	70.42	100	100	94	-	-	-
3	70	683	17	30	94.12	80.46	95.79	70	97.57	-	-	-
4	99	698	2	1	99.62	98.02	99.86	99	99.71	-	-	-
5	99	694	6	1	99.12	94.29	99.86	99	99.14	-	-	-
6	94	685	15	6	97.38	86.24	99.13	94	97.86	-	-	-
7	84	695	5	16	97.38	94.38	97.75	84	99.29	-	-	-
8	97	684	16	3	97.62	85.84	99.56	97	97.71	-	-	-
Total:	738	5466	134	62	96.94	85.63	98.9	92.25	97.61	0.0003499	817	3.604

(b) Test 2.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	100	666	34	0	95.75	74.63	100	100	95.14	-	-	-
2	100	683	17	0	97.88	85.47	100	100	97.57	-	-	-
3	68	689	11	32	94.62	86.08	95.56	68	98.43	-	-	-
4	99	700	0	1	99.88	100	99.86	99	100	-	-	-
5	100	683	17	0	97.88	85.47	100	100	97.57	-	-	-
6	93	687	13	7	97.5	87.74	98.99	93	98.14	-	-	-
7	88	693	7	12	97.62	92.63	98.3	88	99	-	-	-
8	98	688	12	2	98.25	89.09	99.71	98	98.29	-	-	-
Total:	746	5489	111	54	97.42	87.64	99.05	93.25	98.02	0.0002995	859	3.777

(c) Test 3.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	58	682	18	42	92.5	76.32	94.2	58	97.43	-	-	-
2	100	661	39	0	95.12	71.94	100	100	94.43	-	-	-
3	77	691	9	23	96	89.53	96.78	77	98.71	-	-	-
4	100	695	5	0	99.38	95.24	100	100	99.29	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	89	695	5	11	98	94.68	98.44	89	99.29	-	-	-
7	81	682	18	19	95.38	81.82	97.29	81	97.43	-	-	-
8	91	684	16	9	96.88	85.05	98.7	91	97.71	-	-	-
Total:	696	5490	110	104	96.66	86.82	98.18	87	98.04	0.0002576	1276	5.609

(d) Test 4.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	98	697	3	2	99.38	97.03	99.71	98	99.57	-	-	-
2	100	695	5	0	99.38	95.24	100	100	99.29	-	-	-
3	80	697	3	20	97.12	96.39	97.21	80	99.57	-	-	-
4	99	700	0	1	99.88	100	99.86	99	100	-	-	-
5	100	695	5	0	99.38	95.24	100	100	99.29	-	-	-
6	85	697	3	15	97.75	96.59	97.89	85	99.57	-	-	-
7	94	691	9	6	98.12	91.26	99.14	94	98.71	-	-	-
8	90	700	0	10	98.75	100	98.59	90	100	-	-	-
Total:	746	5572	28	54	98.72	96.47	99.05	93.25	99.5	0.0007948	406	1.795

(e) Test 5.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	98	686	14	2	98	87.5	99.71	98	98	-	-	-
2	91	692	8	9	97.88	91.92	98.72	91	98.86	-	-	-
3	94	700	0	6	99.25	100	99.15	94	100	-	-	-
4	97	700	0	3	99.62	100	99.57	97	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	87	689	11	13	97	88.78	98.15	87	98.43	-	-	-
7	67	692	8	33	94.88	89.33	95.45	67	98.86	-	-	-
8	92	668	32	8	95	74.19	98.82	92	95.43	-	-	-
Total:	726	5527	73	74	97.7	91.47	98.7	90.75	98.7	0.0006047	530	2.342

Continued on next page

Table A.7: – continued from previous page

(f) Test 6.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	100	684	16	0	98	86.21	100	100	97.71	-	-	-
2	100	694	6	0	99.25	94.34	100	100	99.14	-	-	-
3	90	697	3	10	98.38	96.77	98.59	90	99.57	-	-	-
4	95	700	0	5	99.38	100	99.29	95	100	-	-	-
5	100	699	1	0	99.88	99.01	100	100	99.86	-	-	-
6	91	695	5	9	98.25	94.79	98.72	91	99.29	-	-	-
7	89	699	1	11	98.5	98.89	98.45	89	99.86	-	-	-
8	95	696	4	5	98.88	95.96	99.29	95	99.43	-	-	-
Total:	760	5564	36	40	98.81	95.75	99.29	95	99.36	0.0003922	471	2.074

(g) Test 7.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	100	697	3	0	99.62	97.09	100	100	99.57	-	-	-
2	94	700	0	6	99.25	100	99.15	94	100	-	-	-
3	99	652	48	1	93.88	67.35	99.85	99	93.14	-	-	-
4	88	700	0	12	98.5	100	98.31	88	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	98	675	25	2	96.62	79.67	99.7	98	96.43	-	-	-
7	83	700	0	17	97.88	100	97.63	83	100	-	-	-
8	92	699	1	8	98.88	98.92	98.87	92	99.86	-	-	-
Total:	754	5523	77	46	98.08	92.88	99.19	94.25	98.62	0.0003856	519	2.286

(h) Test 8.

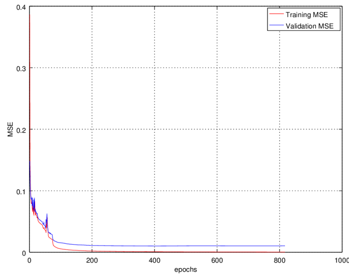
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	54	700	0	46	94.25	100	93.83	54	100	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	87	694	6	13	97.62	93.55	98.16	87	99.14	-	-	-
4	88	698	2	12	98.25	97.78	98.31	88	99.71	-	-	-
5	93	700	0	7	99.12	100	99.01	93	100	-	-	-
6	95	696	4	5	98.88	95.96	99.29	95	99.43	-	-	-
7	85	690	10	15	96.88	89.47	97.87	85	98.57	-	-	-
8	91	683	17	9	96.75	84.26	98.7	91	97.57	-	-	-
Total:	693	5561	39	107	97.72	95.13	98.15	86.62	99.3	0.001501	351	1.55

(i) Test 9.

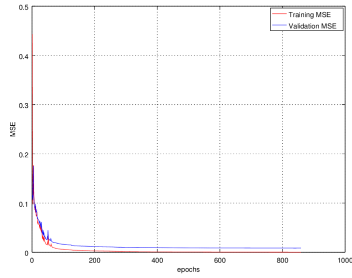
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	55	673	27	45	91	67.07	93.73	55	96.14	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	74	686	14	26	95	84.09	96.35	74	98	-	-	-
4	89	700	0	11	98.62	100	98.45	89	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	93	690	10	7	97.88	90.29	99	93	98.57	-	-	-
7	86	697	3	14	97.88	96.63	98.03	86	99.57	-	-	-
8	99	692	8	1	98.88	92.52	99.86	99	98.86	-	-	-
Total:	696	5538	62	104	97.41	91.33	98.18	87	98.89	0.002047	393	1.734

(j) Test 10.

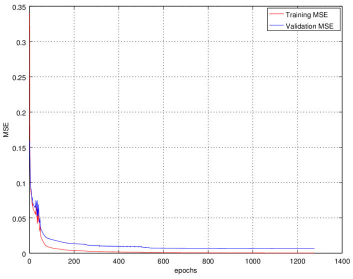
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	61	691	9	39	94	87.14	94.66	61	98.71	-	-	-
2	85	698	2	15	97.88	97.7	97.9	85	99.71	-	-	-
3	86	695	5	14	97.62	94.51	98.03	86	99.29	-	-	-
4	95	695	5	5	98.75	95	99.29	95	99.29	-	-	-
5	100	698	2	0	99.75	98.04	100	100	99.71	-	-	-
6	85	677	23	15	95.25	78.7	97.83	85	96.71	-	-	-
7	91	693	7	9	98	92.86	98.72	91	99	-	-	-
8	95	668	32	5	95.38	74.8	99.26	95	95.43	-	-	-
Total:	698	5515	85	102	97.08	89.84	98.21	87.25	98.48	0.0004005	642	2.826



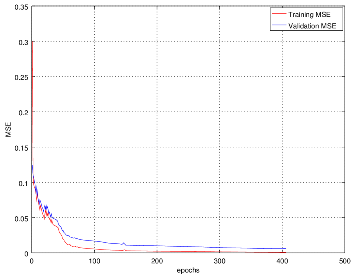
(a) Test 1.



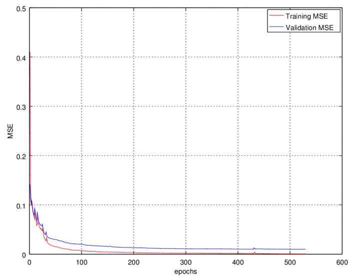
(b) Test 2.



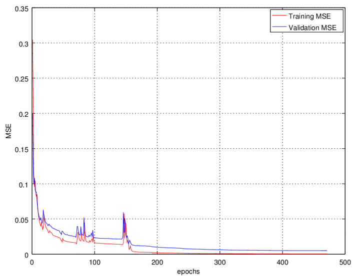
(c) Test 3.



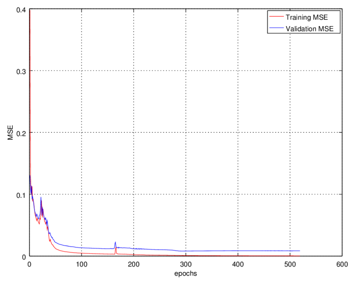
(d) Test 4.



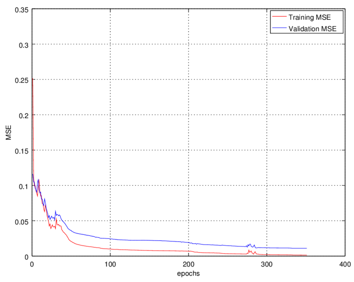
(e) Test 5.



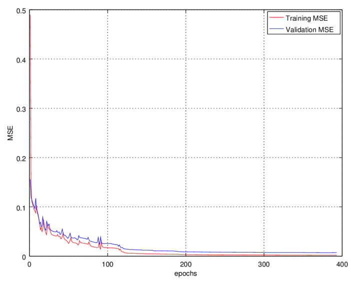
(f) Test 6.



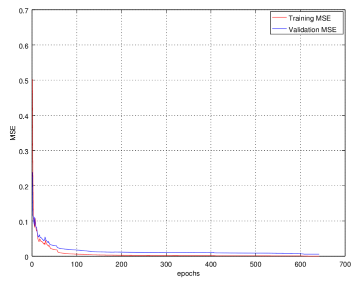
(g) Test 7.



(h) Test 8.

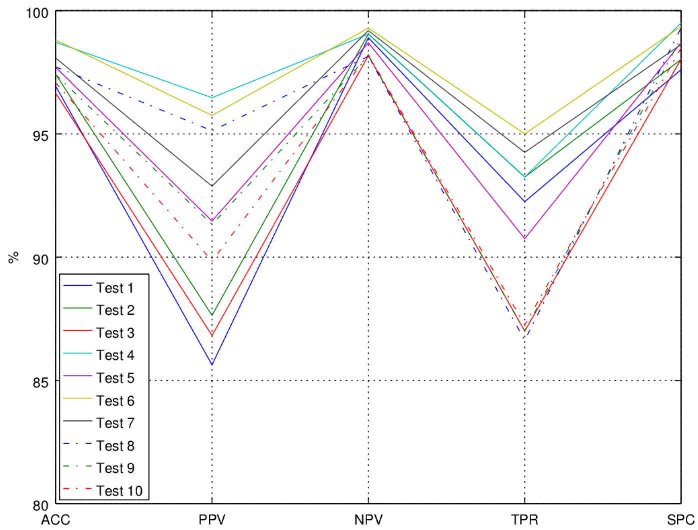


(i) Test 9.

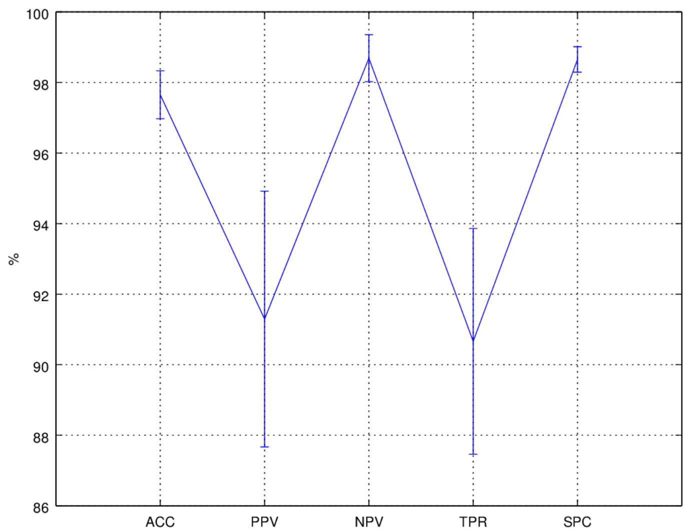


(j) Test 10.

Figure A.7: MSE vs Epochs: Mean Squared Error during the training of each test case.



(a) Comparative graph: Performance for each test case.



(b) Mean performance with error bars indicating standard deviation.

Figure A.8: Visualisation of recognition performance of ANNs with 15 hidden neurons.

Table A.8: Summary of the mean performance.

ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	Epochs	Training Time (s)
97.65 ± 0.68	91.29 ± 3.62	98.69 ± 0.67	90.66 ± 3.20	98.65 ± 0.36	626.40 ± 271.39	2.7596 ± 1.1911

A.5 20 Hidden Neurons

Table A.9: Summary of training and validation results for each test case.

(a) Test 1.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	39	700	0	61	92.38	100	91.98	39	100	-	-	-
2	89	700	0	11	98.62	100	98.45	89	100	-	-	-
3	93	694	6	7	98.38	93.94	99	93	99.14	-	-	-
4	97	699	1	3	99.5	98.98	99.57	97	99.86	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	69	661	39	31	91.25	63.89	95.52	69	94.43	-	-	-
7	93	677	23	7	96.25	80.17	98.98	93	96.71	-	-	-
8	90	699	1	10	98.62	98.9	98.59	90	99.86	-	-	-
Total:	670	5530	70	130	96.88	91.99	97.76	83.75	98.75	0.006212	128	0.7304

(b) Test 2.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	78	700	0	22	97.25	100	96.95	78	100	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	85	691	9	15	97	90.43	97.88	85	98.71	-	-	-
4	95	686	14	5	97.62	87.16	99.28	95	98	-	-	-
5	100	697	3	0	99.62	97.09	100	100	99.57	-	-	-
6	88	692	8	12	97.5	91.67	98.3	88	98.86	-	-	-
7	91	675	25	9	95.75	78.45	98.68	91	96.43	-	-	-
8	94	691	9	6	98.12	91.26	99.14	94	98.71	-	-	-
Total:	731	5532	68	69	97.86	92.01	98.78	91.38	98.79	0.003752	134	0.766

(c) Test 3.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	60	684	16	40	93	78.95	94.48	60	97.71	-	-	-
2	61	700	0	39	95.12	100	94.72	61	100	-	-	-
3	100	698	2	0	99.75	98.04	100	100	99.71	-	-	-
4	100	700	0	0	100	100	100	100	100	-	-	-
5	99	700	0	1	99.88	100	99.86	99	100	-	-	-
6	92	652	48	8	93	65.71	98.79	92	93.14	-	-	-
7	66	679	21	34	93.12	75.86	95.23	66	97	-	-	-
8	94	685	15	6	97.38	86.24	99.13	94	97.86	-	-	-
Total:	672	5498	102	128	96.41	88.1	97.78	84	98.18	0.0002697	607	3.445

(d) Test 4.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	94	691	9	6	98.12	91.26	99.14	94	98.71	-	-	-
2	100	689	11	0	98.62	90.09	100	100	98.43	-	-	-
3	91	664	36	9	94.38	71.65	98.66	91	94.86	-	-	-
4	95	694	6	5	98.62	94.06	99.28	95	99.14	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	100	699	1	0	99.88	99.01	100	100	99.86	-	-	-
7	89	680	20	11	96.12	81.65	98.41	89	97.14	-	-	-
8	98	696	4	2	99.25	96.08	99.71	98	99.43	-	-	-
Total:	767	5513	87	33	98.12	90.48	99.4	95.88	98.45	0.0001961	891	5.094

(e) Test 5.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	98	674	26	2	96.5	79.03	99.7	98	96.29	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	95	691	9	5	98.25	91.35	99.28	95	98.71	-	-	-
4	100	699	1	0	99.88	99.01	100	100	99.86	-	-	-
5	100	696	4	0	99.5	96.15	100	100	99.43	-	-	-
6	96	696	4	4	99	96	99.43	96	99.43	-	-	-
7	94	674	26	6	96	78.33	99.12	94	96.29	-	-	-
8	94	689	11	6	97.88	89.52	99.14	94	98.43	-	-	-
Total:	777	5519	81	23	98.38	91.17	99.58	97.12	98.55	0.0002685	708	4.014

Continued on next page

Table A.9: – continued from previous page

(f) Test 6.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	97	699	1	3	99.5	98.98	99.57	97	99.86	-	-	-
2	100	695	5	0	99.38	95.24	100	100	99.29	-	-	-
3	96	700	0	4	99.5	100	99.43	96	100	-	-	-
4	100	700	0	0	100	100	100	100	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	88	693	7	12	97.62	92.63	98.3	88	99	-	-	-
7	97	681	19	3	97.25	83.62	99.56	97	97.29	-	-	-
8	95	700	0	5	99.38	100	99.29	95	100	-	-	-
Total:	773	5568	32	27	99.08	96.31	99.52	96.62	99.43	0.0002235	673	3.817

(g) Test 7.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	94	673	27	6	95.88	77.69	99.12	94	96.14	-	-	-
2	100	696	4	0	99.5	96.15	100	100	99.43	-	-	-
3	73	693	7	27	95.75	91.25	96.25	73	99	-	-	-
4	99	700	0	1	99.88	100	99.86	99	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	91	686	14	9	97.12	86.67	98.71	91	98	-	-	-
7	86	696	4	14	97.75	95.56	98.03	86	99.43	-	-	-
8	91	698	2	9	98.62	97.85	98.73	91	99.71	-	-	-
Total:	734	5542	58	66	98.06	93.15	98.84	91.75	98.96	0.0003019	696	3.947

(h) Test 8.

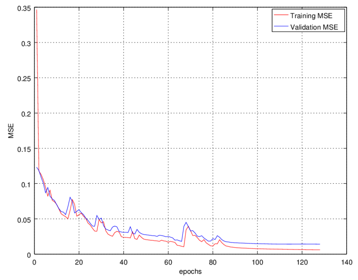
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	52	697	3	48	93.62	94.55	93.56	52	99.57	-	-	-
2	92	698	2	8	98.75	97.87	98.87	92	99.71	-	-	-
3	80	690	10	20	96.25	88.89	97.18	80	98.57	-	-	-
4	100	700	0	0	100	100	100	100	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	90	698	2	10	98.5	97.83	98.59	90	99.71	-	-	-
7	78	694	6	22	96.5	92.86	96.93	78	99.14	-	-	-
8	90	694	6	10	98	93.75	98.58	90	99.14	-	-	-
Total:	682	5571	29	118	97.7	95.72	97.96	85.25	99.48	0.001659	253	1.437

(i) Test 9.

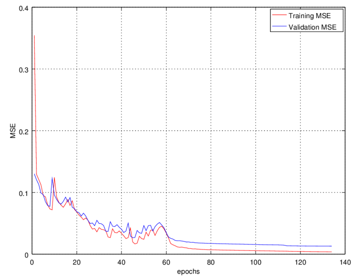
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	96	681	19	4	97.12	83.48	99.42	96	97.29	-	-	-
2	94	700	0	6	99.25	100	99.15	94	100	-	-	-
3	81	700	0	19	97.62	100	97.36	81	100	-	-	-
4	100	700	0	0	100	100	100	100	100	-	-	-
5	100	699	1	0	99.88	99.01	100	100	99.86	-	-	-
6	82	689	11	18	96.38	88.17	97.45	82	98.43	-	-	-
7	96	688	12	4	98	88.89	99.42	96	98.29	-	-	-
8	86	700	0	14	98.25	100	98.04	86	100	-	-	-
Total:	735	5557	43	65	98.31	94.94	98.85	91.88	99.23	0.0007126	424	2.409

(j) Test 10.

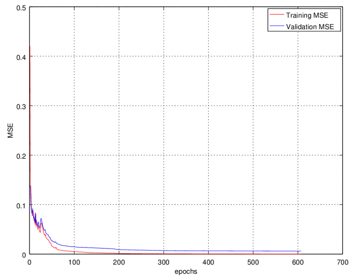
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	97	700	0	3	99.62	100	99.57	97	100	-	-	-
2	100	699	1	0	99.88	99.01	100	100	99.86	-	-	-
3	98	696	4	2	99.25	96.08	99.71	98	99.43	-	-	-
4	96	697	3	4	99.12	96.97	99.43	96	99.57	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	72	682	18	28	94.25	80	96.06	72	97.43	-	-	-
7	86	687	13	14	96.62	86.87	98	86	98.14	-	-	-
8	92	700	0	8	99	100	98.87	92	100	-	-	-
Total:	741	5561	39	59	98.47	94.87	98.96	92.62	99.3	0.0002227	624	3.543



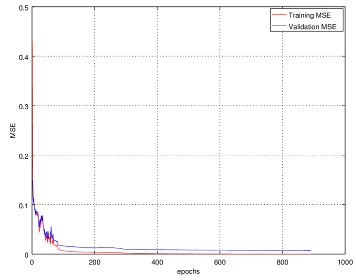
(a) Test 1.



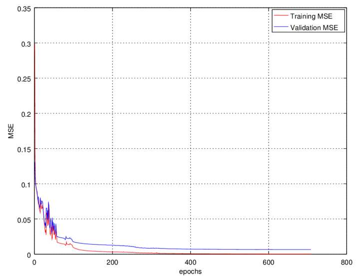
(b) Test 2.



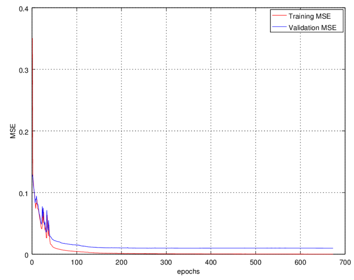
(c) Test 3.



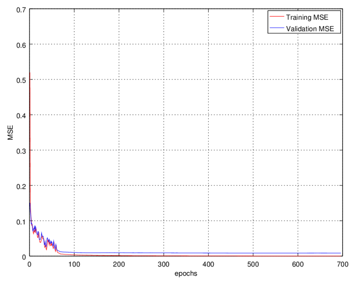
(d) Test 4.



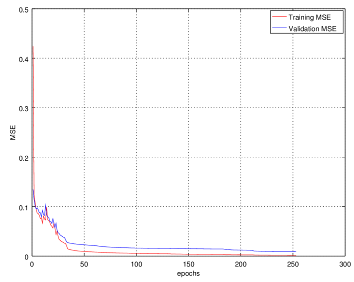
(e) Test 5.



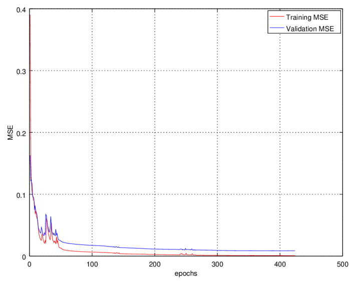
(f) Test 6.



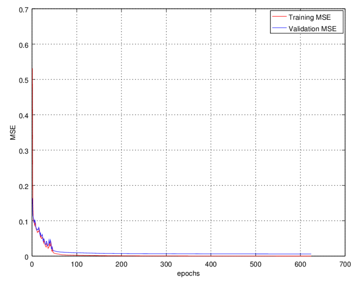
(g) Test 7.



(h) Test 8.

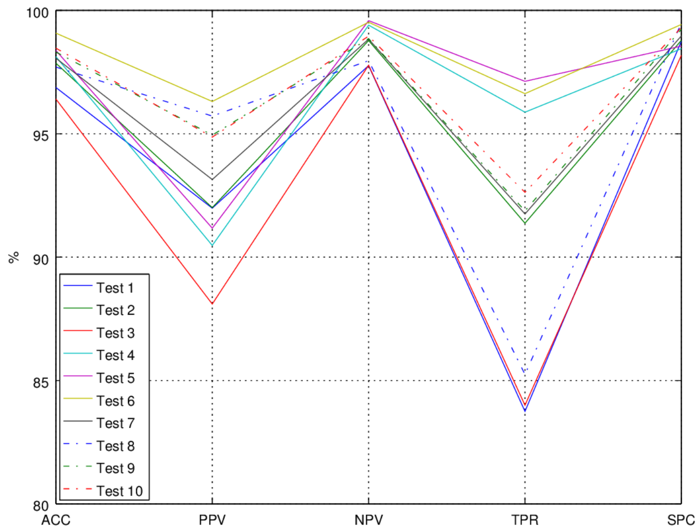


(i) Test 9.

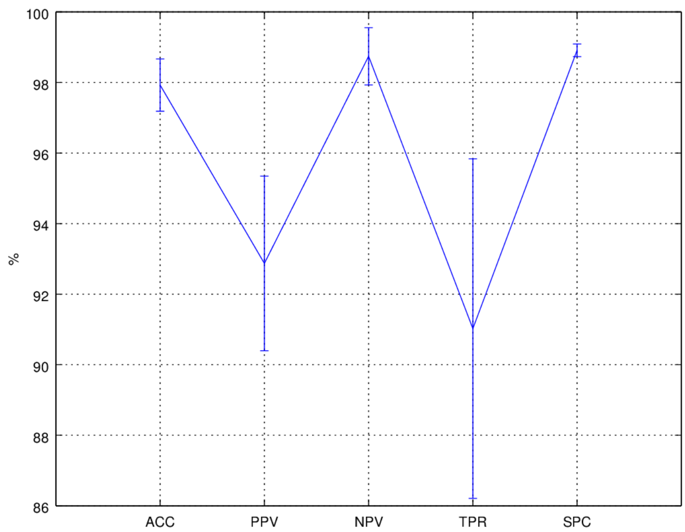


(j) Test 10.

Figure A.9: MSE vs Epochs: Mean Squared Error during the training of each test case.



(a) Comparative graph: Performance for each test case.



(b) Mean performance with error bars indicating standard deviation.

Figure A.10: Visualisation of recognition performance of ANNs with 20 hidden neurons.

Table A.10: Summary of the mean performance.

ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	Epochs	Training Time (s)
97.93 ± 0.74	92.87 ± 2.47	98.74 ± 0.81	91.03 ± 4.81	98.91 ± 0.18	513.80 ± 250.71	2.9202 ± 1.4261

A.6 30 Hidden Neurons

Table A.11: Summary of training and validation results for each test case.

(a) Test 1.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	78	699	1	22	97.12	98.73	96.95	78	99.86	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	97	695	5	3	99	95.1	99.57	97	99.29	-	-	-
4	96	698	2	4	99.25	97.96	99.43	96	99.71	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	96	681	19	4	97.12	83.48	99.42	96	97.29	-	-	-
7	89	696	4	11	98.12	95.7	98.44	89	99.43	-	-	-
8	95	697	3	5	99	96.94	99.29	95	99.57	-	-	-
Total:	751	5566	34	49	98.7	95.99	99.14	93.88	99.39	0.001181	285	2.337

(b) Test 2.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	78	684	16	22	95.25	82.98	96.88	78	97.71	-	-	-
2	100	698	2	0	99.75	98.04	100	100	99.71	-	-	-
3	94	695	5	6	98.62	94.95	99.14	94	99.29	-	-	-
4	89	699	1	11	98.5	98.89	98.45	89	99.86	-	-	-
5	99	700	0	1	99.88	100	99.86	99	100	-	-	-
6	83	697	3	17	97.5	96.51	97.62	83	99.57	-	-	-
7	78	695	5	22	96.62	93.98	96.93	78	99.29	-	-	-
8	87	699	1	13	98.25	98.86	98.17	87	99.86	-	-	-
Total:	708	5567	33	92	98.05	95.53	98.38	88.5	99.41	0.002863	92	0.7729

(c) Test 3.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	97	695	5	3	99	95.1	99.57	97	99.29	-	-	-
2	89	696	4	11	98.12	95.7	98.44	89	99.43	-	-	-
3	87	698	2	13	98.12	97.75	98.17	87	99.71	-	-	-
4	97	700	0	3	99.62	100	99.57	97	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	80	698	2	20	97.25	97.56	97.21	80	99.71	-	-	-
7	52	694	6	48	93.25	89.66	93.53	52	99.14	-	-	-
8	92	697	3	8	98.62	96.84	98.87	92	99.57	-	-	-
Total:	694	5578	22	106	98	96.58	98.17	86.75	99.61	0.000163	567	4.635

(d) Test 4.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	61	695	5	39	94.5	92.42	94.69	61	99.29	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	100	699	1	0	99.88	99.01	100	100	99.86	-	-	-
4	100	697	3	0	99.62	97.09	100	100	99.57	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	92	698	2	8	98.75	97.87	98.87	92	99.71	-	-	-
7	82	692	8	18	96.75	91.11	97.46	82	98.86	-	-	-
8	89	694	6	11	97.88	93.68	98.44	89	99.14	-	-	-
Total:	724	5575	25	76	98.42	96.4	98.68	90.5	99.55	0.0003258	388	3.177

(e) Test 5.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	100	697	3	0	99.62	97.09	100	100	99.57	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	90	695	5	10	98.12	94.74	98.58	90	99.29	-	-	-
4	96	700	0	4	99.5	100	99.43	96	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	94	678	22	6	96.5	81.03	99.12	94	96.86	-	-	-
7	52	685	15	48	92.12	77.61	93.45	52	97.86	-	-	-
8	95	669	31	5	95.5	75.4	99.26	95	95.57	-	-	-
Total:	727	5524	76	73	97.67	90.73	98.73	90.88	98.64	0.0001664	708	5.775

Continued on next page

Table A.11: – continued from previous page

(f) Test 6.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	94	694	6	6	98.5	94	99.14	94	99.14	-	-	-
2	95	700	0	5	99.38	100	99.29	95	100	-	-	-
3	96	689	11	4	98.12	89.72	99.42	96	98.43	-	-	-
4	100	700	0	0	100	100	100	100	100	-	-	-
5	99	700	0	1	99.88	100	99.86	99	100	-	-	-
6	82	657	43	18	92.38	65.6	97.33	82	93.86	-	-	-
7	46	696	4	54	92.75	92	92.8	46	99.43	-	-	-
8	94	684	16	6	97.25	85.45	99.13	94	97.71	-	-	-
Total:	706	5520	80	94	97.28	90.85	98.37	88.25	98.57	0.0001525	629	5.126

(g) Test 7.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	99	687	13	1	98.25	88.39	99.85	99	98.14	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	94	694	6	6	98.5	94	99.14	94	99.14	-	-	-
4	100	699	1	0	99.88	99.01	100	100	99.86	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	48	696	4	52	93	92.31	93.05	48	99.43	-	-	-
7	94	692	8	6	98.25	92.16	99.14	94	98.86	-	-	-
8	95	700	0	5	99.38	100	99.29	95	100	-	-	-
Total:	730	5568	32	70	98.41	95.73	98.81	91.25	99.43	9.955e-05	778	6.329

(h) Test 8.

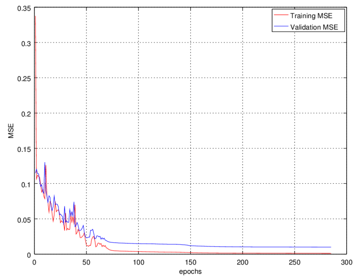
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	79	669	31	21	93.5	71.82	96.96	79	95.57	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	79	694	6	21	96.62	92.94	97.06	79	99.14	-	-	-
4	96	700	0	4	99.5	100	99.43	96	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	90	695	5	10	98.12	94.74	98.58	90	99.29	-	-	-
7	88	690	10	12	97.25	89.8	98.29	88	98.57	-	-	-
8	99	696	4	1	99.38	96.12	99.86	99	99.43	-	-	-
Total:	731	5544	56	69	98.05	93.18	98.77	91.38	99	0.0001223	786	6.393

(i) Test 9.

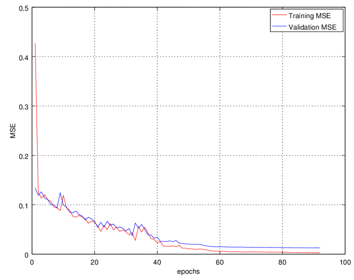
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	57	695	5	43	94	91.94	94.17	57	99.29	-	-	-
2	97	700	0	3	99.62	100	99.57	97	100	-	-	-
3	98	693	7	2	98.88	93.33	99.71	98	99	-	-	-
4	97	700	0	3	99.62	100	99.57	97	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	93	690	10	7	97.88	90.29	99	93	98.57	-	-	-
7	84	684	16	16	96	84	97.71	84	97.71	-	-	-
8	88	699	1	12	98.38	98.88	98.31	88	99.86	-	-	-
Total:	714	5561	39	86	98.05	94.8	98.51	89.25	99.3	0.0002577	534	4.347

(j) Test 10.

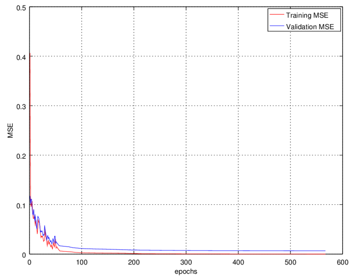
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	98	653	47	2	93.88	67.59	99.69	98	93.29	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	86	700	0	14	98.25	100	98.04	86	100	-	-	-
4	100	697	3	0	99.62	97.09	100	100	99.57	-	-	-
5	100	696	4	0	99.5	96.15	100	100	99.43	-	-	-
6	81	689	11	19	96.25	88.04	97.32	81	98.43	-	-	-
7	82	698	2	18	97.5	97.62	97.49	82	99.71	-	-	-
8	98	690	10	2	98.5	90.74	99.71	98	98.57	-	-	-
Total:	745	5523	77	55	97.94	92.15	99.03	93.12	98.62	0.0001764	822	6.791



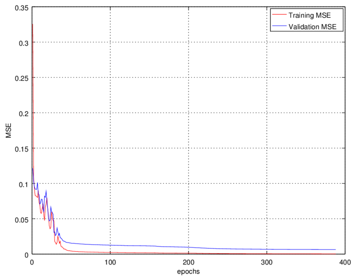
(a) Test 1.



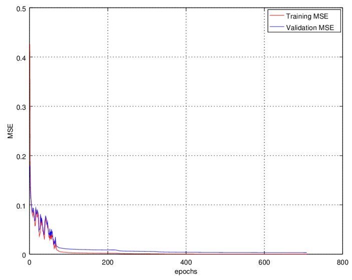
(b) Test 2.



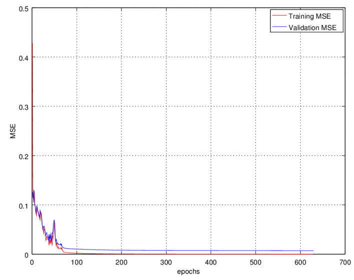
(c) Test 3.



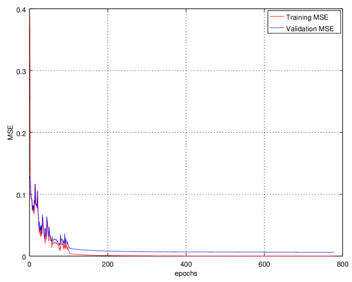
(d) Test 4.



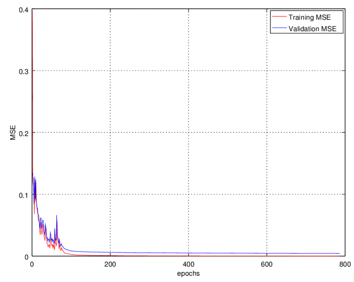
(e) Test 5.



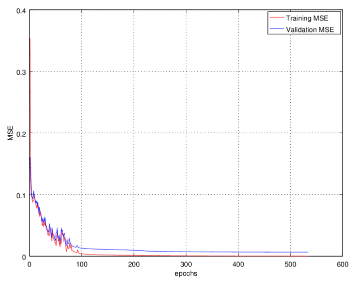
(f) Test 6.



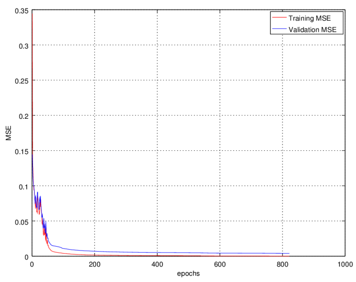
(g) Test 7.



(h) Test 8.

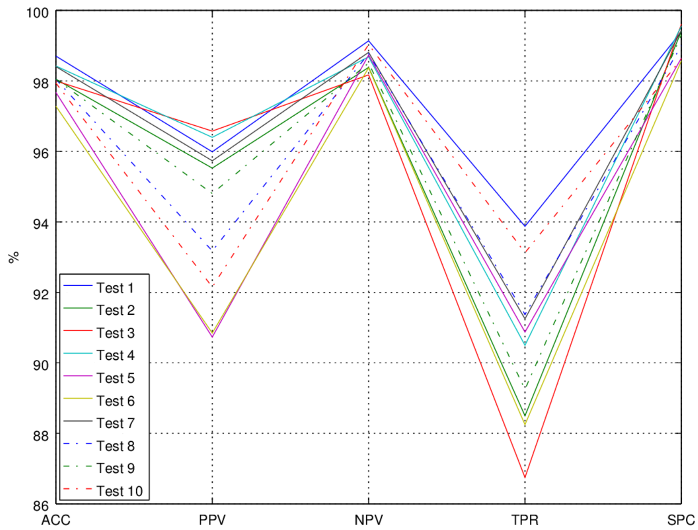


(i) Test 9.

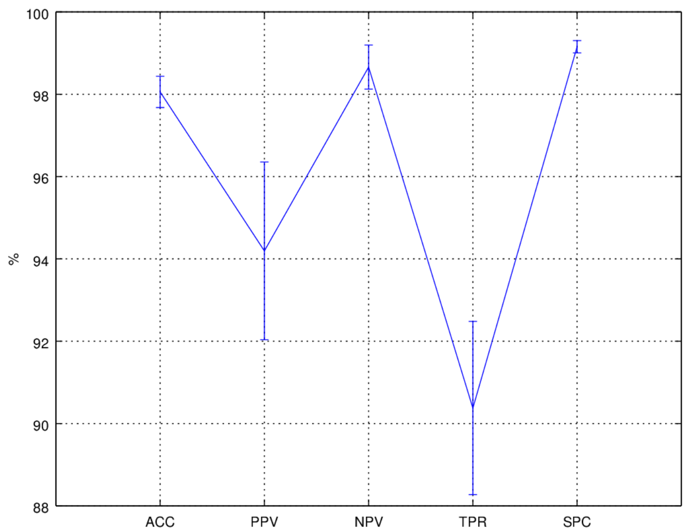


(j) Test 10.

Figure A.11: MSE vs Epochs: Mean Squared Error during the training of each test case.



(a) Comparative graph: Performance for each test case.



(b) Mean performance with error bars indicating standard deviation.

Figure A.12: Visualisation of recognition performance of ANNs with 30 hidden neurons.

Table A.12: Summary of the mean performance.

ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	Epochs	Training Time (s)
98.06 ± 0.38	94.19 ± 2.16	98.66 ± 0.54	90.38 ± 2.10	99.15 ± 0.15	558.90 ± 227.78	4.5683 ± 1.8578

A.7 40 Hidden Neurons

Table A.13: Summary of training and validation results for each test case.

(a) Test 1.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	42	700	0	58	92.75	100	92.35	42	100	-	-	-
2	78	700	0	22	97.25	100	96.95	78	100	-	-	-
3	96	689	11	4	98.12	89.72	99.42	96	98.43	-	-	-
4	89	700	0	11	98.62	100	98.45	89	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	81	684	16	19	95.62	83.51	97.3	81	97.71	-	-	-
7	87	691	9	13	97.25	90.62	98.15	87	98.71	-	-	-
8	91	698	2	9	98.62	97.85	98.73	91	99.71	-	-	-
Total:	664	5562	38	136	97.28	95.21	97.67	83	99.32	0.002521	161	1.743

(b) Test 2.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	67	697	3	33	95.5	95.71	95.48	67	99.57	-	-	-
2	100	699	1	0	99.88	99.01	100	100	99.86	-	-	-
3	94	700	0	6	99.25	100	99.15	94	100	-	-	-
4	100	700	0	0	100	100	100	100	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	93	688	12	7	97.62	88.57	98.99	93	98.29	-	-	-
7	83	699	1	17	97.75	98.81	97.63	83	99.86	-	-	-
8	92	699	1	8	98.88	98.92	98.87	92	99.86	-	-	-
Total:	729	5582	18	71	98.61	97.63	98.76	91.12	99.68	9.521e-05	744	7.977

(c) Test 3.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	65	698	2	35	95.38	97.01	95.23	65	99.71	-	-	-
2	91	699	1	9	98.75	98.91	98.73	91	99.86	-	-	-
3	86	697	3	14	97.88	96.63	98.03	86	99.57	-	-	-
4	100	700	0	0	100	100	100	100	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	85	696	4	15	97.62	95.51	97.89	85	99.43	-	-	-
7	96	687	13	4	97.88	88.07	99.42	96	98.14	-	-	-
8	93	695	5	7	98.5	94.9	99	93	99.29	-	-	-
Total:	716	5572	28	84	98.25	96.38	98.54	89.5	99.5	9.291e-05	808	8.817

(d) Test 4.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	97	694	6	3	98.88	94.17	99.57	97	99.14	-	-	-
2	100	692	8	0	99	92.59	100	100	98.86	-	-	-
3	97	698	2	3	99.38	97.98	99.57	97	99.71	-	-	-
4	100	700	0	0	100	100	100	100	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	94	691	9	6	98.12	91.26	99.14	94	98.71	-	-	-
7	82	690	10	18	96.5	89.13	97.46	82	98.57	-	-	-
8	94	697	3	6	98.88	96.91	99.15	94	99.57	-	-	-
Total:	764	5562	38	36	98.84	95.26	99.36	95.5	99.32	0.0001259	564	6.072

(e) Test 5.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	53	700	0	47	94.12	100	93.71	53	100	-	-	-
2	100	698	2	0	99.75	98.04	100	100	99.71	-	-	-
3	86	695	5	14	97.62	94.51	98.03	86	99.29	-	-	-
4	100	700	0	0	100	100	100	100	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	56	695	5	44	93.88	91.8	94.05	56	99.29	-	-	-
7	49	691	9	51	92.5	84.48	93.13	49	98.71	-	-	-
8	99	699	1	1	99.75	99	99.86	99	99.86	-	-	-
Total:	643	5578	22	157	97.2	95.98	97.35	80.38	99.61	0.0001142	594	6.334

Continued on next page

Table A.13: – continued from previous page

(f) Test 6.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	55	681	19	45	92	74.32	93.8	55	97.29	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	97	697	3	3	99.25	97	99.57	97	99.57	-	-	-
4	99	700	0	1	99.88	100	99.86	99	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	98	685	15	2	97.88	86.73	99.71	98	97.86	-	-	-
7	82	691	9	18	96.62	90.11	97.46	82	98.71	-	-	-
8	90	698	2	10	98.5	97.83	98.59	90	99.71	-	-	-
Total:	721	5552	48	79	98.02	93.25	98.62	90.12	99.14	0.0001097	808	8.676

(g) Test 7.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	87	692	8	13	97.38	91.58	98.16	87	98.86	-	-	-
2	100	669	31	0	96.12	76.34	100	100	95.57	-	-	-
3	88	700	0	12	98.5	100	98.31	88	100	-	-	-
4	88	696	4	12	98	95.65	98.31	88	99.43	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	75	700	0	25	96.88	100	96.55	75	100	-	-	-
7	81	692	8	19	96.62	91.01	97.33	81	98.86	-	-	-
8	84	660	40	16	93	67.74	97.63	84	94.29	-	-	-
Total:	703	5509	91	97	97.06	90.29	98.29	87.88	98.37	0.0008384	245	2.66

(h) Test 8.

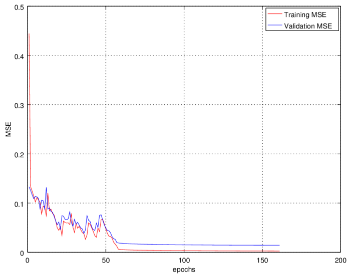
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	100	698	2	0	99.75	98.04	100	100	99.71	-	-	-
2	89	700	0	11	98.62	100	98.45	89	100	-	-	-
3	92	696	4	8	98.5	95.83	98.86	92	99.43	-	-	-
4	100	698	2	0	99.75	98.04	100	100	99.71	-	-	-
5	100	697	3	0	99.62	97.09	100	100	99.57	-	-	-
6	92	692	8	8	98	92	98.86	92	98.86	-	-	-
7	86	693	7	14	97.38	92.47	98.02	86	99	-	-	-
8	89	690	10	11	97.38	89.9	98.43	89	98.57	-	-	-
Total:	748	5564	36	52	98.62	95.42	99.08	93.5	99.36	0.0001221	670	7.155

(i) Test 9.

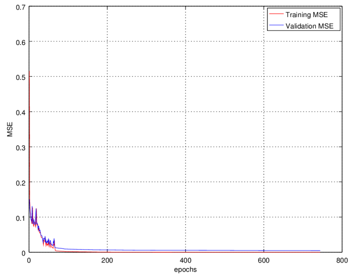
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	82	681	19	18	95.38	81.19	97.42	82	97.29	-	-	-
2	99	700	0	1	99.88	100	99.86	99	100	-	-	-
3	96	699	1	4	99.38	98.97	99.43	96	99.86	-	-	-
4	100	700	0	0	100	100	100	100	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	91	692	8	9	97.88	91.92	98.72	91	98.86	-	-	-
7	78	700	0	22	97.25	100	96.95	78	100	-	-	-
8	92	690	10	8	97.75	90.2	98.85	92	98.57	-	-	-
Total:	738	5562	38	62	98.44	95.28	98.9	92.25	99.32	0.001206	180	1.923

(j) Test 10.

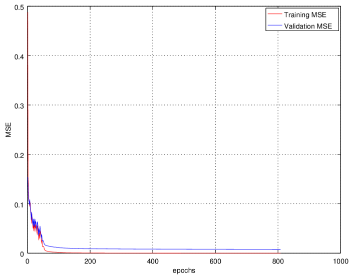
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	54	700	0	46	94.25	100	93.83	54	100	-	-	-
2	96	700	0	4	99.5	100	99.43	96	100	-	-	-
3	82	694	6	18	97	93.18	97.47	82	99.14	-	-	-
4	99	700	0	1	99.88	100	99.86	99	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	88	683	17	12	96.38	83.81	98.27	88	97.57	-	-	-
7	84	690	10	16	96.75	89.36	97.73	84	98.57	-	-	-
8	89	700	0	11	98.62	100	98.45	89	100	-	-	-
Total:	692	5567	33	108	97.8	95.79	98.13	86.5	99.41	6.033e-05	613	6.538



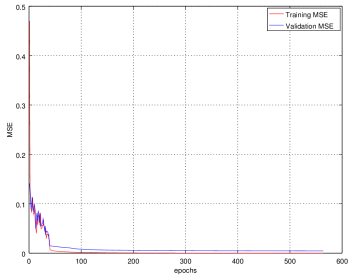
(a) Test 1.



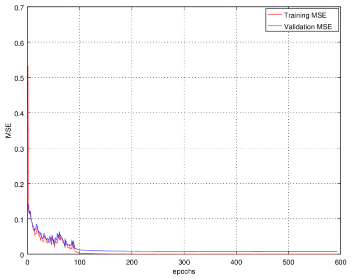
(b) Test 2.



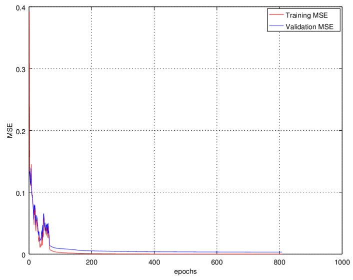
(c) Test 3.



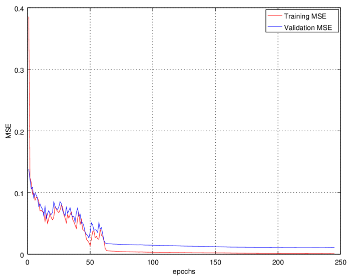
(d) Test 4.



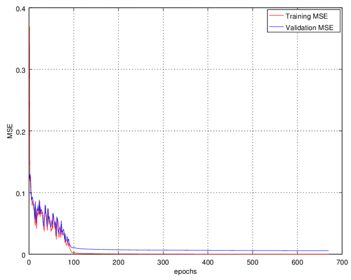
(e) Test 5.



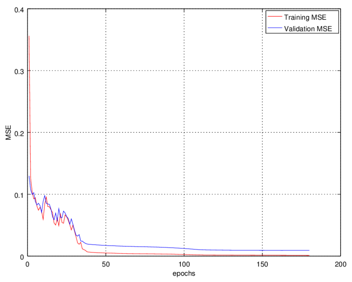
(f) Test 6.



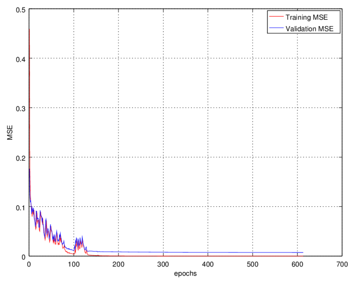
(g) Test 7.



(h) Test 8.

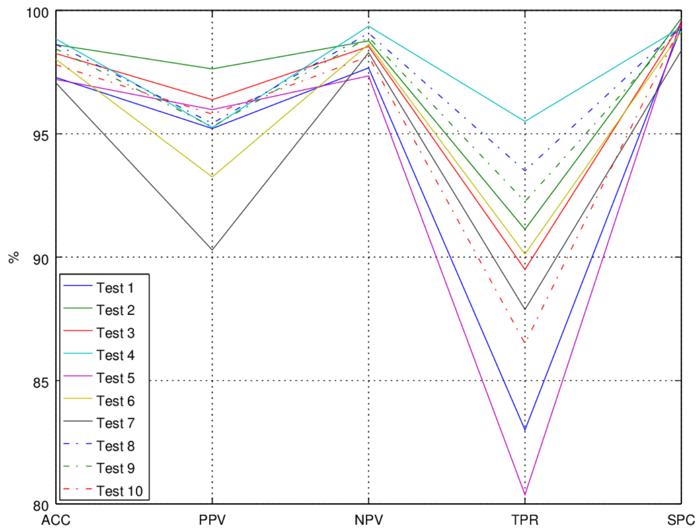


(i) Test 9.

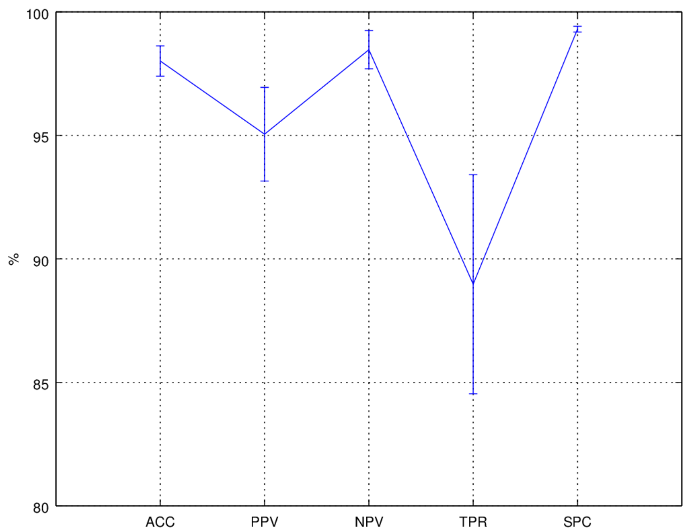


(j) Test 10.

Figure A.13: MSE vs Epochs: Mean Squared Error during the training of each test case.



(a) Comparative graph: Performance for each test case.



(b) Mean performance with error bars indicating standard deviation.

Figure A.14: Visualisation of recognition performance of ANNs with 40 hidden neurons.

Table A.14: Summary of the mean performance.

ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	Epochs	Training Time (s)
98.01 ± 0.62	95.05 ± 1.90	98.47 ± 0.77	88.97 ± 4.44	99.30 ± 0.12	538.70 ± 239.01	5.7895 ± 2.5729

A.8 60 Hidden Neurons

Table A.15: Summary of training and validation results for each test case.

(a) Test 1.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	98	683	17	2	97.62	85.22	99.71	98	97.57	-	-	-
2	100	699	1	0	99.88	99.01	100	100	99.86	-	-	-
3	96	694	6	4	98.75	94.12	99.43	96	99.14	-	-	-
4	98	697	3	2	99.38	97.03	99.71	98	99.57	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	80	689	11	20	96.12	87.91	97.18	80	98.43	-	-	-
7	54	697	3	46	93.88	94.74	93.81	54	99.57	-	-	-
8	89	696	4	11	98.12	95.7	98.44	89	99.43	-	-	-
Total:	715	5555	45	85	97.97	94.22	98.54	89.38	99.2	3.409e-05	925	14.47

(b) Test 2.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	100	698	2	0	99.75	98.04	100	100	99.71	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	86	695	5	14	97.62	94.51	98.03	86	99.29	-	-	-
4	96	696	4	4	99	96	99.43	96	99.43	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	89	685	15	11	96.75	85.58	98.42	89	97.86	-	-	-
7	92	684	16	8	97	85.19	98.84	92	97.71	-	-	-
8	89	699	1	11	98.5	98.89	98.45	89	99.86	-	-	-
Total:	752	5557	43	48	98.58	94.77	99.15	94	99.23	0.0003386	163	2.571

(c) Test 3.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	100	687	13	0	98.38	88.5	100	100	98.14	-	-	-
2	100	687	13	0	98.38	88.5	100	100	98.14	-	-	-
3	88	700	0	12	98.5	100	98.31	88	100	-	-	-
4	97	699	1	3	99.5	98.98	99.57	97	99.86	-	-	-
5	99	700	0	1	99.88	100	99.86	99	100	-	-	-
6	88	700	0	12	98.5	100	98.31	88	100	-	-	-
7	88	693	7	12	97.62	92.63	98.3	88	99	-	-	-
8	92	699	1	8	98.88	98.92	98.87	92	99.86	-	-	-
Total:	752	5565	35	48	98.7	95.94	99.15	94	99.37	0.000106	446	7.015

(d) Test 4.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	68	700	0	32	96	100	95.63	68	100	-	-	-
2	89	699	1	11	98.5	98.89	98.45	89	99.86	-	-	-
3	96	697	3	4	99.12	96.97	99.43	96	99.57	-	-	-
4	97	696	4	3	99.12	96.04	99.57	97	99.43	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	93	699	1	7	99	98.94	99.01	93	99.86	-	-	-
7	44	699	1	56	92.88	97.78	92.58	44	99.86	-	-	-
8	87	684	16	13	96.38	84.47	98.13	87	97.71	-	-	-
Total:	674	5574	26	126	97.62	96.63	97.85	84.25	99.54	9.475e-05	605	9.479

(e) Test 5.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	84	694	6	16	97.25	93.33	97.75	84	99.14	-	-	-
2	85	696	4	15	97.62	95.51	97.89	85	99.43	-	-	-
3	94	697	3	6	98.88	96.91	99.15	94	99.57	-	-	-
4	97	699	1	3	99.5	98.98	99.57	97	99.86	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	79	694	6	21	96.62	92.94	97.06	79	99.14	-	-	-
7	47	692	8	53	92.38	85.45	92.89	47	98.86	-	-	-
8	85	698	2	15	97.88	97.7	97.9	85	99.71	-	-	-
Total:	671	5570	30	129	97.52	95.1	97.78	83.88	99.46	3.922e-05	813	12.74

Continued on next page

Table A.15: – continued from previous page

(f) Test 6.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	100	688	12	0	98.5	89.29	100	100	98.29	-	-	-
2	96	697	3	4	99.12	96.97	99.43	96	99.57	-	-	-
3	97	697	3	3	99.25	97	99.57	97	99.57	-	-	-
4	100	698	2	0	99.75	98.04	100	100	99.71	-	-	-
5	100	699	1	0	99.88	99.01	100	100	99.86	-	-	-
6	88	687	13	12	96.88	87.13	98.28	88	98.14	-	-	-
7	44	697	3	56	92.62	93.62	92.56	44	99.57	-	-	-
8	100	697	3	0	99.62	97.09	100	100	99.57	-	-	-
Total:	725	5560	40	75	98.2	94.77	98.73	90.62	99.29	0.0001768	254	3.994

(g) Test 7.

Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	74	700	0	26	96.75	100	96.42	74	100	-	-	-
2	94	700	0	6	99.25	100	99.15	94	100	-	-	-
3	95	695	5	5	98.75	95	99.29	95	99.29	-	-	-
4	95	690	10	5	98.12	90.48	99.28	95	98.57	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	74	698	2	26	96.5	97.37	96.41	74	99.71	-	-	-
7	84	684	16	16	96	84	97.71	84	97.71	-	-	-
8	100	700	0	0	100	100	100	100	100	-	-	-
Total:	716	5567	33	84	98.17	95.86	98.53	89.5	99.41	0.0001267	373	5.853

(h) Test 8.

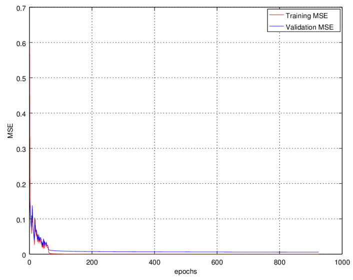
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	66	700	0	34	95.75	100	95.37	66	100	-	-	-
2	99	697	3	1	99.5	97.06	99.86	99	99.57	-	-	-
3	95	698	2	5	99.12	97.94	99.29	95	99.71	-	-	-
4	95	699	1	5	99.25	98.96	99.29	95	99.86	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	55	692	8	45	93.38	87.3	93.89	55	98.86	-	-	-
7	89	680	20	11	96.12	81.65	98.41	89	97.14	-	-	-
8	95	695	5	5	98.75	95	99.29	95	99.29	-	-	-
Total:	694	5561	39	106	97.73	94.74	98.17	86.75	99.3	7.479e-05	430	6.744

(i) Test 9.

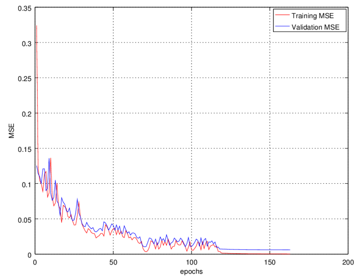
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	89	697	3	11	98.25	96.74	98.45	89	99.57	-	-	-
2	97	699	1	3	99.5	98.98	99.57	97	99.86	-	-	-
3	93	695	5	7	98.5	94.9	99	93	99.29	-	-	-
4	97	700	0	3	99.62	100	99.57	97	100	-	-	-
5	100	699	1	0	99.88	99.01	100	100	99.86	-	-	-
6	94	696	4	6	98.75	95.92	99.15	94	99.43	-	-	-
7	88	677	23	12	95.62	79.28	98.26	88	96.71	-	-	-
8	98	698	2	2	99.5	98	99.71	98	99.71	-	-	-
Total:	756	5561	39	44	98.7	95.35	99.21	94.5	99.3	6.708e-05	461	7.221

(j) Test 10.

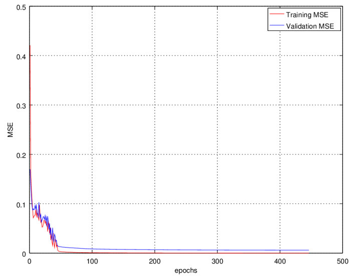
Object	TP	TN	FP	FN	ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	MSE	Epochs	Training Time (s)
1	100	681	19	0	97.62	84.03	100	100	97.29	-	-	-
2	100	700	0	0	100	100	100	100	100	-	-	-
3	95	697	3	5	99	96.94	99.29	95	99.57	-	-	-
4	96	700	0	4	99.5	100	99.43	96	100	-	-	-
5	100	700	0	0	100	100	100	100	100	-	-	-
6	89	690	10	11	97.38	89.9	98.43	89	98.57	-	-	-
7	84	691	9	16	96.88	90.32	97.74	84	98.71	-	-	-
8	91	691	9	9	97.75	91	98.71	91	98.71	-	-	-
Total:	755	5550	50	45	98.52	94.02	99.2	94.38	99.11	4.921e-05	714	11.2



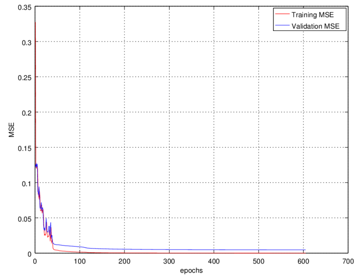
(a) Test 1.



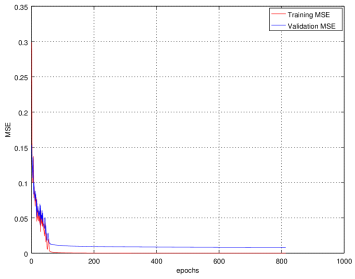
(b) Test 2.



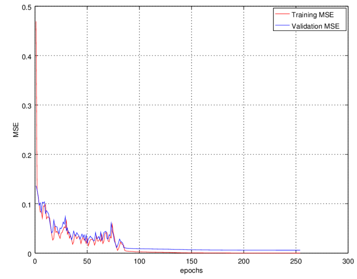
(c) Test 3.



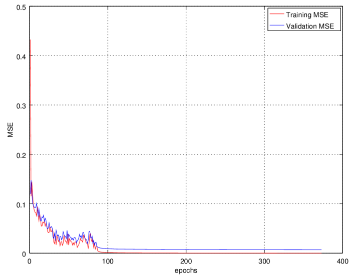
(d) Test 4.



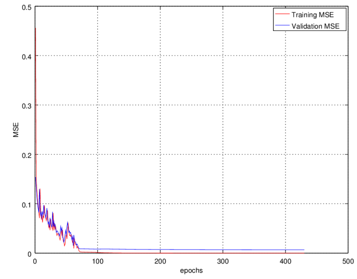
(e) Test 5.



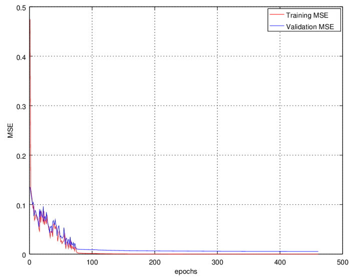
(f) Test 6.



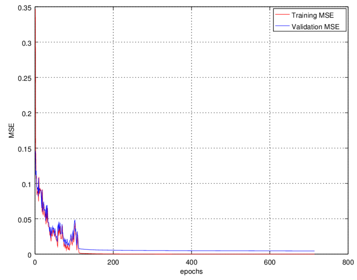
(g) Test 7.



(h) Test 8.

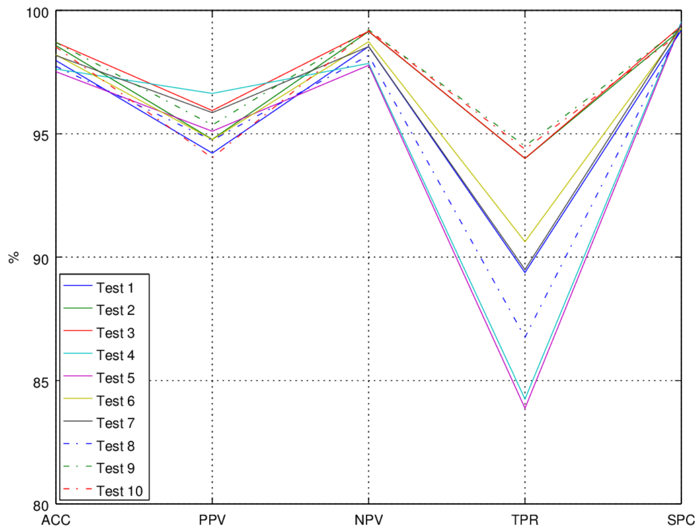


(i) Test 9.

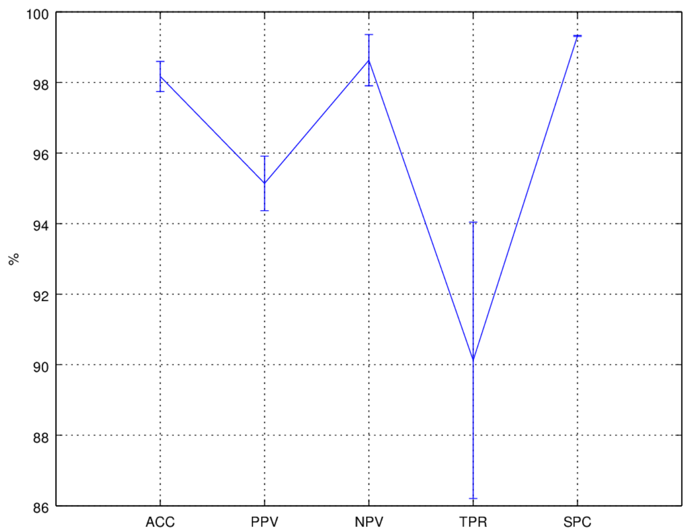


(j) Test 10.

Figure A.15: MSE vs Epochs: Mean Squared Error during the training of each test case.



(a) Comparative graph: Performance for each test case.



(b) Mean performance with error bars indicating standard deviation.

Figure A.16: Visualisation of recognition performance of ANNs with 60 hidden neurons.

Table A.16: Summary of the mean performance.

ACC (%)	PPV (%)	NPV (%)	TPR (%)	SPC (%)	Epochs	Training Time (s)
98.17 ± 0.43	95.14 ± 0.77	98.63 ± 0.73	90.12 ± 3.92	99.32 ± 0.01	518.40 ± 230.56	8.1295 ± 3.6050

Appendix B

Source Code

B.1 Laser Scanner Related Code

B.1.1 Serial Port Class

The serial port class is used for simple (*read line – write line*) UART communication between the PC and a serial device.

Listing B.1: Header file of the serial port class.

```
#ifndef SERIALPORT_HPP
#define SERIALPORT_HPP

#include <boost/asio.hpp>
#include <boost/chrono.hpp>
#include <boost/thread.hpp>
#include <iostream>

#define DEFAULT_DEVICE      "/dev/ttyACM0"
#define DEFAULD_BAUDRATE    9600

class SerialPort
{
public:
    SerialPort(std::string port = DEFAULT_DEVICE,
               int baudrate = DEFAULD_BAUDRATE);
    ~SerialPort();
    bool isOpen();
    void writeLine(std::string tx_string);
    std::string readLine();
private:
    boost::asio::io_service io;
    boost::asio::serial_port serial;
    bool PORT_OPENED;
};

#endif // SERIALPORT_HPP
```

Listing B.2: Implementation of the serial port class.

```
#include "serialport.hpp"

SerialPort::SerialPort(std::string port, int baudrate) : io(), serial(io)
{
    try {
        serial.open(port);
        if (!serial.is_open()) {
            throw "Serial port was not opened";
        } else {
            serial.set_option(boost::asio::serial_port_base::baud_rate(baudrate));
            serial.set_option(boost::asio::serial_port_base::character_size(8));
            serial.set_option(boost::asio::serial_port_base::parity(boost::asio::serial_port::parity::none));
            serial.set_option(boost::asio::serial_port_base::stop_bits(boost::asio::serial_port::stop_bits::one));
            serial.set_option(boost::asio::serial_port_base::flow_control(boost::asio::serial_port::flow_control::none));
            boost::this_thread::sleep(boost::posix_time::seconds(2));
            PORT_OPENED = true;
        }
    } catch (const char* msg) {
        std::cerr << "Exception in SerialPort::SerialPort: " << msg << std::endl;
        PORT_OPENED = false;
    } catch (boost::exception_detail::clone_impl<boost::exception_detail::error_info_injector<boost::system::system_error> > &e) {
        std::cerr << "Exception in SerialPort::SerialPort: " << e.what() << std::endl;
        PORT_OPENED = false;
    }
}

SerialPort::~SerialPort()
{
    serial.close();
}

bool SerialPort::isOpen()
{
    return PORT_OPENED;
}

void SerialPort::writeLine(std::string tx_string)
{
    boost::asio::write(serial, boost::asio::buffer(tx_string.c_str(),tx_string.size()));
}

std::string SerialPort::readLine()
{
    char c;
    std::string result;
    while (1) {
        boost::asio::read(serial,boost::asio::buffer(&c,1));
        switch (c) {
        case '\r':
            break;
        case '\n':
            return result;
        default:
            result+=c;
        }
    }
}
```

B.1.2 Controller Class

Listing B.3: Header file of the controller class.

```
#ifndef CONTROLLER_HPP
#define CONTROLLER_HPP

#include "serialport.hpp"
#include <string>

class Controller
{
    friend class LaserScanner;
public:
    Controller();

    // Getters
    double getRotationTime();

    // Functions
    bool ping();
    bool sentStartCaptureSignal();
    void rotateFull();

private:
    SerialPort serial;
    bool START_CAPTURE;
    double ROTATION_TIME;
};

#endif // CONTROLLER_HPP
```

Listing B.4: Implementation of the controller class.

```
#include "controller.hpp"

Controller::Controller()
{
    START_CAPTURE = false;
}

double Controller::getRotationTime()
{
    return ROTATION_TIME;
}

bool Controller::ping()
{
    try {
        std::cout << "Pinging scanner ......... ";
        std::cout.flush();
        if (!serial.isOpen()) {
            std::cout << "No controller found." << std::endl;
            throw "Device not found! Check connection.";
        } else {
            serial.writeLine("1");
            std::string ack_ping = "Received ping";
            if (ack_ping.compare(serial.readLine()) == 0) {
                std::cout << "Connected and ready." << std::endl;
                return true;
            } else {
            throw "Device not found! Check connection.";
            }
        }
    } catch (const char *msg) {
        std::cerr << "Exception: " << msg << std::endl;
        return false;
```

```
    } catch (boost::exception_detail::clone_impl<boost::exception_detail::error_info_injector<boost::system::system_error> > &e) {
        std::cerr << "Exception: " << e.what() << std::endl;
        return false;
    }
}

bool Controller::sentStartCaptureSignal()
{
    return START_CAPTURE;
}

void Controller::rotateFull()
{
    std::string ack_start = "Stepper started";
    std::string ack_stop = "Stepper complete";
    boost::chrono::system_clock::time_point start;
    boost::chrono::system_clock::time_point end;
    serial.writeLine("2");
    while (1) {
        std::string response = serial.readLine();
        if (response.compare(ack_start) == 0) {
            START_CAPTURE = true;
            start = boost::chrono::system_clock::now();
        } else if (response.compare(ack_stop) == 0) {
            START_CAPTURE = false;
            end = boost::chrono::system_clock::now();
            break;
        }
    }
    boost::chrono::duration<double> duration = end - start;
    ROTATION_TIME = double(duration.count());
}
```

B.1.3 Camera Class

Listing B.5: Header file of the camera class.

```
#ifndef CAMERA_HPP
#define CAMERA_HPP

#include <boost/chrono.hpp>
#include <boost/thread.hpp>
#include <iostream>
#include <opencv2/opencv.hpp>
#include <vector>

// Defaults
#define DEFAULT_CAMERA_ID       0
#define DEFAULT_FRAME_RATE      30.0f
#define DEFAULT_FRAME_HEIGHT    600.0f
#define DEFAULT_FRAME_WIDTH     800.0f

struct Pattern
{
    float squareSize;
    cv::Size size;
    std::vector<cv::Point3f> objectPoints()
    {
        std::vector<cv::Point3f> result;
        for (int i = 0; i < this->size.height; ++i) {
            for (int j = 0; j < this->size.width; ++j) {
                result.push_back(cv::Point3f(float(j*this->squareSize), float(i*this->
                    squareSize), 0));
            }
        }
        return result;
    }
};

class Camera
{
    friend class ImageProcessor;
    friend class LaserScanner;
public:
    Camera(int camera_id = DEFAULT_CAMERA_ID,
           double frame_height = DEFAULT_FRAME_HEIGHT,
           double frame_width = DEFAULT_FRAME_WIDTH,
           double frame_rate = DEFAULT_FRAME_RATE);

    // Getters

    // Setters
    void setId(int camera_id);
    void setFrameRate(double frame_rate);
    void setFrameHeight(double frame_height);
    void setFrameWidth(double frame_width);
    void startCapture();

    // Functions
    bool ping();
    void capture();
    bool calibrate();
    void saveCalibrationData(std::string filename = "/home/ivan/MEng/C++/Shared Files/
        calibration_data.xml");

private:
    // Variables
    int ID;
    double FRAME_RATE, FRAME_HEIGHT, FRAME_WIDTH;
    std::vector<cv::Mat> RGB_BUFFER, CALIBRATION_IMAGES;
    std::vector<double> TIME_STAMPS;
    bool START_CAPTURE;
    Pattern CHESSBOARD;
    std::vector<std::vector<cv::Point2f> > IMAGE_POINTS;
```

```
    std::vector<std::vector<cv::Point3f> > OBJECT_POINTS;
    cv::Mat CAMERA_MATRIX, DIST_COEFFS;
    std::vector<cv::Mat> R_VECS, T_VECS;
    double REPROJECTION_ERROR;

    // Functions
    void getPatternValues();
    bool captureCalibrationImages();
    bool performCalibration();
};

#endif // CAMERA_HPP
```

Listing B.6: Implementation of the camera class.

```
#include "camera.hpp"

Camera::Camera(int camera_id, double frame_height,
               double frame_width, double frame_rate)
{
    ID = camera_id;
    FRAME_HEIGHT = frame_height;
    FRAME_WIDTH = frame_width;
    FRAME_RATE = frame_rate;
}

void Camera::setId(int camera_id)
{
    ID = camera_id;
}

void Camera::setFrameRate(double frame_rate)
{
    FRAME_RATE = frame_rate;
}

void Camera::setFrameHeight(double frame_height)
{
    FRAME_HEIGHT = 600;//frame_height;
}

void Camera::setFrameWidth(double frame_width)
{
    FRAME_WIDTH = 800;//frame_width;
}

void Camera::startCapture()
{
    START_CAPTURE = true;
}

bool Camera::ping()
{
    try {
        std::cout << "Pinging camera " << ID << " ........ ";
        std::cout.flush();
        cv::VideoCapture cap(ID);
        if (!cap.isOpened()) {
            std::cout << "No camera found." << std::endl;
            throw "Device not found! Check connection.";
        } else {
            std::cout << "Connected and ready." << std::endl;
            cap.release();
            return true;
        }
    } catch (const char *msg) {
        std::cerr << "Exception: " << msg << std::endl;
        return false;
    }
}

```

```
void Camera::capture()
{
    boost::chrono::system_clock::time_point start;
    boost::chrono::system_clock::time_point capture_time;
    boost::chrono::duration<double> duration;
    try {
        cv::VideoCapture cap(ID);
        if (!cap.isOpened()) {
            throw "Device not found!";
        } else {
            cap.set(CV_CAP_PROP_FPS, FRAME_RATE);
            cap.set(CV_CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT);
            cap.set(CV_CAP_PROP_FRAME_WIDTH, FRAME_WIDTH);
            cv::Mat frame;
            std::cout << "Camera adjusting to ambient light .. ";
            std::cout << std::flush;
            while (!START_CAPTURE) { // ugly but effective hack to allow camera to:
                                     // 1) adjust to ambient light conditions before
                cap >> frame;        //    capturing frames to frame buffer
            }                        // 2) ensure synchronisation with turn table
            std::cout << "Complete." << std::endl;

            std::cout << "Capturing images of object ......... ";
            std::cout << std::flush;
            start = boost::chrono::system_clock::now();
            while (1) {
                cap >> frame;
                RGB_BUFFER.push_back(frame.clone());
                capture_time = boost::chrono::system_clock::now();
                duration = capture_time - start;
                TIME_STAMPS.push_back(double(duration.count()));
                boost::this_thread::interruption_point();
            }
        }
    } catch (const char* msg) {
        std::cerr << "Exception: " << msg << std::endl;
    } catch (boost::thread_interrupted&) {
        std::cout << "Complete." << std::endl;
        return;
    }
}

bool Camera::calibrate()
{
    getPatternValues();
    if (!captureCalibrationImages()) return false;
    else if (!performCalibration()) return false;
    else return true;
}

void Camera::saveCalibrationData(std::string filename)
{
    cv::FileStorage fs(filename, cv::FileStorage::WRITE);
    fs << "CameraMatrix" << CAMERA_MATRIX;
    fs << "DistCoeffs" << DIST_COEFFS;
    fs << "rvecs" << R_VECS;
    fs << "tvecs" << T_VECS;
    fs << "ReprojectionError" << REPROJECTION_ERROR;
    fs.release();
}

void Camera::getPatternValues()
{
    std::cout << "Enter the number of corners along the width of the board  : ";
    std::cin >> CHESSBOARD.size.width;
    std::cout << "Enter the number of corners along the height of the board : ";
    std::cin >> CHESSBOARD.size.height;
    std::cout << "Enter the length of the sides of the squares in milimeters: ";
    std::cin >> CHESSBOARD.squareSize;
}

bool Camera::captureCalibrationImages()
{
```

```
    try {
        cv::VideoCapture cap(ID);
        if (!cap.isOpened()) {
            throw "Device not found!";
        } else {
            cap.set(CV_CAP_PROP_FRAME_HEIGHT, 600);
            cap.set(CV_CAP_PROP_FRAME_WIDTH, 800);
            cap.set(CV_CAP_PROP_FPS, FRAME_RATE);
            char c = 0;
            cv::Mat frame;
            int counter = 0;
            std::cout << "Press <ESC> to stop capturing images . . . " << std::endl;
            while (1) {
                cap >> frame;
                cv::imshow("Calibrate", frame);
                c = cv::waitKey(10);
                counter++;
                if (counter == 30) {
                    RGB_BUFFER.push_back(frame.clone());
                    cv::Mat subMat(frame.rows, frame.cols, frame.type(),cv::Scalar::
                        all(255));
                    cv::imshow("Calibrate", subMat - frame);
                    cv::waitKey(5);
                    counter = 0;
                }
                if (c == 27) {break;}
            }
            cv::destroyAllWindows();
            return true;
        }
    } catch (const char* msg) {
        std::cerr << "Exception: " << msg << std::endl;
        return false;
    } catch (cv::Exception &e) {
        std::cerr << "Exception: " << e.what() << std::endl;
        return false;
    }
}

bool Camera::performCalibration()
{
    try {
        cv::Mat gray;
        std::vector<cv::Point2f> corners;

        // Find good images for calibration (ie images in RGB_BUFFER containing
            chessboard pattern)
        for (int i = 0; i < RGB_BUFFER.size(); ++i) {
            bool found;
            found = cv::findChessboardCorners(RGB_BUFFER[i],
                                              CHESSBOARD.size,
                                              corners,
                                              CV_CALIB_CB_ADAPTIVE_THRESH |
                                                  CV_CALIB_CB_FAST_CHECK |
                                                  CV_CALIB_CB_NORMALIZE_IMAGE);
            if (found) {
                cv::cvtColor(RGB_BUFFER[i], gray, CV_BGR2GRAY);
                cv::cornerSubPix(gray, corners, cv::Size(11,11), cv::Size(-1,-1),
                                 cv::TermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER,
                                     30, 0.001));
                IMAGE_POINTS.push_back(corners);
                OBJECT_POINTS.push_back(CHESSBOARD.objectPoints());
                cv::drawChessboardCorners(RGB_BUFFER[i], CHESSBOARD.size, corners,
                    found);
                CALIBRATION_IMAGES.push_back(RGB_BUFFER[i].clone());
            }
        }

        if (CALIBRATION_IMAGES.size() < 10) {
            throw "Too few images to perform a reliable calibration!";
        } else {
            REPROJECTION_ERROR = cv::calibrateCamera(OBJECT_POINTS,
                                                     IMAGE_POINTS,
```

```
                                                     cv::Size(FRAME_WIDTH,
                                                         FRAME_HEIGHT),
                                                     CAMERA_MATRIX,
                                                     DIST_COEFFS,
                                                     R_VECS, T_VECS);

            for (int i = 0; i < CALIBRATION_IMAGES.size(); ++i) {
                cv::imshow("Calibration Images", CALIBRATION_IMAGES[i]);
                std::stringstream ss;
                ss << "calibration/" << i << ".jpg";
                std::string filename;
                ss >> filename;
                cv::imwrite(filename.c_str(), CALIBRATION_IMAGES[i]);
                cv::waitKey(100);
            }
            return true;
        }

    } catch (const char* msg) {
        std::cerr << "Exception: " << msg << std::endl;
        return false;
    }
}
```

B.1.4 Image Processing Class

Listing B.7: Header file of the image processing class.

```
#ifndef IMAGEPROCESSOR_HPP
#define IMAGEPROCESSOR_HPP

#include <boost/filesystem.hpp>
#include "camera.hpp"

namespace fs = boost::filesystem;

#define BLUE_CHANNEL     0
#define GREEN_CHANNEL    1
#define RED_CHANNEL      2
#define THRESHOLD_METHOD_1  1
#define THRESHOLD_METHOD_2  2

#define DEFAULT_FRAME_RATE 30.0f

class ImageProcessor
{
    friend class LaserScanner;
public:
    ImageProcessor();

    // Getters

    // Setters
    void setCameraMatrix(cv::Mat cameraMatrix);
    void setDistortionCoefficients(cv::Mat distCoeffs);
    void loadCalibrationDataFrom(std::string filename);

    bool readImagesInDirectory(fs::path imageDirectory);

    // Functions
    bool Undistort(Camera camera);
    bool Threshold(int channel = RED_CHANNEL,
                   int threshold_method = THRESHOLD_METHOD_2);
    bool Thin();

private:
    // Variables
    cv::Mat CAMERA_MATRIX, DISTORTION_COEFFICIENTS;
    std::vector<cv::Mat> UNDISTORTED_IMAGES;
    std::vector<cv::Mat> THRESHOLD_IMAGES;
    std::vector<cv::Mat> THINNED_IMAGES;
    std::vector<std::vector<cv::Point2d> > IMAGE_POINTS;

    std::vector<fs::path> list;
    //double THRESHOLD_TIME, THINNING_TIME;

    // Functions
    double ThresholdMethod_1(cv::Mat input);
    double ThresholdMethod_2(cv::Mat input);

    bool readFileNames2List(fs::path dir);

};

#endif // IMAGEPROCESSOR_HPP
```

Listing B.8: Implementation of the image processing class.

```
#include "imageprocessor.hpp"
#include <string>
#include <sstream>

```

```
ImageProcessor::ImageProcessor(){}

void ImageProcessor::setCameraMatrix(cv::Mat cameraMatrix)
{
    CAMERA_MATRIX = cameraMatrix;
}

void ImageProcessor::setDistortionCoefficients(cv::Mat distCoeffs)
{
    DISTORTION_COEFFICIENTS = distCoeffs;
}

void ImageProcessor::loadCalibrationDataFrom(std::string filename)
{
    cv::FileStorage fs(filename, cv::FileStorage::READ);
    fs["CameraMatrix"] >> CAMERA_MATRIX;
    fs["DistCoeffs"] >> DISTORTION_COEFFICIENTS;
    fs.release();
}

bool ImageProcessor::readImagesInDirectory(boost::filesystem::path imageDirectory)
{
    try {
        if (!readFileNames2List(imageDirectory)) {
            throw "readFileNames2List(imageDirectory) failed.";
        } else {
            fs::path tempPath;
            for(int i = 0; i < list.size(); i++)
            {
                tempPath = imageDirectory;
                tempPath /= list[i];
                UNDISTORTED_IMAGES.push_back(cv::imread(tempPath.c_str()));
            }
            return true;
        }
    } catch (const char* msg) {
        std::cerr << "Error in ImageProcessor::readImagesInDirectory:" << msg << std::endl;
        return false;
    } catch (...) {
        std::cerr << "Unknown error in: ImageProcessor::readImagesInDirectory" << std::endl;
        return false;
    }
}

bool ImageProcessor::Undistort(Camera camera)
{
    try {
        if (camera.RGB_BUFFER.empty()) {
            throw "Buffer empty. No images to undistort.";
        } else {
            std::cout << "Undistorting images ............... ";
            std::cout << std::flush;

            //boost::chrono::system_clock::time_point start = boost::chrono::system_clock::now();
            for(int i = 0; i < camera.RGB_BUFFER.size(); i++)
            {
                cv::Mat tempResult;
                cv::undistort(camera.RGB_BUFFER[i],
                              tempResult,
                              CAMERA_MATRIX,
                              DISTORTION_COEFFICIENTS);
                UNDISTORTED_IMAGES.push_back(tempResult.clone());
            }
            //boost::chrono::system_clock::time_point stop = boost::chrono::system_clock::now();
            //boost::chrono::duration<double> sec = stop - start;
            //UNDISTORT_TIME = double(sec.count());
            std::cout << "Complete." << std::endl;
            return true;
        }
```

```
    } catch (const char* msg) {
        std::cerr << "Exception: " << msg << std::endl;
        return false;
    }
}

bool ImageProcessor::Threshold(int channel, int threshold_method)
{
    try {
        if (UNDISTORTED_IMAGES.empty()) {
            throw "No source image to perform threshold operations on!";
        } else {
            std::cout << "Performing threshold operation ..... ";
            std::cout << std::flush;
            double thresh;
            cv::Mat input, result;
            std::vector<cv::Mat> BGR;
            //boost::chrono::system_clock::time_point start = boost::chrono::system_clock::now();
            for (int i = 0; i < UNDISTORTED_IMAGES.size(); i++) {
                cv::split(UNDISTORTED_IMAGES[i], BGR);
                input = BGR[channel].clone();

                switch (threshold_method) {
                case THRESHOLD_METHOD_1:
                    thresh = ThresholdMethod_1(input);
                    break;
                case THRESHOLD_METHOD_2:
                    thresh = ThresholdMethod_2(input);
                    break;
                default:
                    std::cerr << "Warning: Undefined threshold option selected! Using method 2 as default.";
                    thresh = ThresholdMethod_2(input);
                    break;
                }

                cv::threshold(input, result, thresh, 255, 0);
                std::stringstream ss;
                ss << "segmented/" << i << ".jpg";
                std::string filename;
                ss >> filename;
                cv::imwrite(filename.c_str(), result.clone());
                THRESHOLD_IMAGES.push_back(result.clone());
            }
            //boost::chrono::system_clock::time_point stop = boost::chrono::system_clock::now();
            //boost::chrono::duration<double> duration = stop - start;
            //THRESHOLD_TIME = double(duration.count());
            std::cout << "Complete." << std::endl;
            return true;
        }
    } catch (const char *msg) {
        std::cerr << "Exception: " << msg << std::endl;
        return false;
    }
}

bool ImageProcessor::Thin()
{
    try {
        if (THRESHOLD_IMAGES.empty()) {
            throw "No binary thresholded images to thin!";
        } else {
            //boost::chrono::system_clock::time_point start = boost::chrono::system_clock::now();
            std::cout << "Performing thinning operation ...... ";
            std::cout << std::flush;

            for (int i = 0; i < THRESHOLD_IMAGES.size(); i++) {
                cv::Mat input = THRESHOLD_IMAGES[i];
                std::vector<cv::Point2d> tempPoints;
                cv::Mat result = cv::Mat::zeros(input.rows, input.cols, CV_8UC1);
```

```

                    for (int j = 0; j < input.rows; j++) {
                        int left_pixel, right_pixel;

                        for (int k = 0; k < input.cols; k++) {
                            cv::Scalar intensity = input.at<uchar>(j,k);
                            if (intensity[0] > 0) {
                                left_pixel = k;
                                break;
                            }
                        }

                        for (int k = input.cols - 1; k >= 0; k--) {
                            cv::Scalar intensity = input.at<uchar>(j,k);
                            if (intensity[0] > 0) {
                                right_pixel = k;
                                break;
                            }
                        }

                        //double center = left_pixel + 0.5*(right_pixel - left_pixel);
                        double center = 0.5*(left_pixel + right_pixel);
                        cv::Scalar intensity = input.at<uchar>(j, int(center));
                        if (intensity[0] > 0) {
                            tempPoints.push_back(cv::Point2d(center, j));
                            result.at<uchar>(j, int(center)) = 255;
                        }
                    }
                    std::stringstream ss;
                    ss << "thinned/" << i << ".jpg";
                    std::string filename;
                    ss >> filename;
                    cv::imwrite(filename.c_str(), result.clone());
                    IMAGE_POINTS.push_back(tempPoints);
                    THINNED_IMAGES.push_back(result.clone());
                }
                //boost::chrono::system_clock::time_point stop = boost::chrono::
                    system_clock::now();
                //boost::chrono::duration<double> duration = stop - start;
                //THINNING_TIME = double(duration.count());
                std::cout << "Complete." << std::endl;
                return true;
            }
        } catch (const char *msg) {
            std::cerr << "Exception: " << msg << std::endl;
            return false;
        }
}

double ImageProcessor::ThresholdMethod_1(cv::Mat input)
{
    double min, max, thresholdValue;

    cv::minMaxIdx(input, &min, &max);
    if(max == 255)
        thresholdValue = 253;
    else
        thresholdValue = max - 2;

    return thresholdValue;
}

double ImageProcessor::ThresholdMethod_2(cv::Mat input)
{
    double min, max, thresholdValue;
    cv::Mat mean, stddev;

    cv::meanStdDev(input, mean, stddev);
    cv::minMaxIdx(input, &min, &max);

    thresholdValue = mean.at<double>(0,0) + 4*stddev.at<double>(0,0);
    if (thresholdValue > max) {
        thresholdValue = mean.at<double>(0,0) + 3.5*stddev.at<double>(0,0);
```

```
        if (thresholdValue > max) {
            thresholdValue = mean.at<double>(0,0) + 3*stddev.at<double>(0,0);
            if (thresholdValue > max) {
                thresholdValue = mean.at<double>(0,0) + 2.5*stddev.at<double>(0,0);
                if (thresholdValue > max) {
                    thresholdValue = mean.at<double>(0,0) + 2*stddev.at<double>(0,0);
                    if (thresholdValue > max) {
                        thresholdValue = mean.at<double>(0,0) + 1.5*stddev.at<double>(0,0);
                        if (thresholdValue > max) {
                            thresholdValue = mean.at<double>(0,0) + 1.0*stddev.at<double>(0,0);
                            if (thresholdValue > max) {
                                thresholdValue = mean.at<double>(0,0) + 0.5*stddev.at<double>(0,0);
                            }
                        }
                    }
                }
            }
        }
    }
    return thresholdValue;
}

bool ImageProcessor::readFileNames2List(fs::path dir)
{
    try {
        list.clear();
        std::vector<fs::path> v;
        copy(fs::directory_iterator(dir), fs::directory_iterator(), back_inserter(v));
        sort(v.begin(), v.end());
        for(int i = 0; i < v.size(); i++)
            list.push_back(v[i].filename());
        return true;
    } catch (...) {
        std::cerr << "Error in private method: ImageProcessor::readFileNames2List" << std::endl;
        return false;
    }
}
```

B.1.5 Laser Scanner Class

Listing B.9: Header file of the laser scanner class.

```
#ifndef LASERSCANNER_HPP
#define LASERSCANNER_HPP

#include "camera.hpp"
#include "imageprocessor.hpp"
#include "controller.hpp"

#include <boost/filesystem.hpp>

#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/visualization/pcl_visualizer.h>

#define OPENCV_VERSION 100*CV_MAJOR_VERSION + 10*CV_MINOR_VERSION +
    CV_SUBMINOR_VERSION

const float PI = atan(1)*4;

struct CameraParameters
{
    int id;
    double frame_rate, frame_height, frame_width;
};

class LaserScanner
{
public:
    LaserScanner(std::string sim_data_filename);
    LaserScanner(CameraParameters params);

    // Getters
    bool ping();

    // Setters
    void setScannerParametersFromFile(std::string filename);
    void setSimulationImageDirectory(fs::path directory);

    // Functions
    bool calibrate();
    void saveCalibrationData(std::string image_filename = "/home/ivan/MEng/C++/Shared
        Files/result.png",
                             const char *imagePoints_filename = "/home/ivan/MEng/C++/
                                 Shared Files/imagePoints.csv",
                             std::string calibrationData_filename = "/home/ivan/MEng/C
                                 ++/Shared Files/scanner_calibration_data.xml");
    bool Scan(int channel = RED_CHANNEL,
              int threshold_method = THRESHOLD_METHOD_2);
    void showPointCloud();
    void displayImageBuffers();
    void savePointCloudToFile(std::string filename = "/home/ivan/MEng/C++/Temp/scan.
        pcd");

private:
    // Objects
    bool simulation;
    Camera camera;
    Controller controller;
    ImageProcessor imgProcessor;

    // Variables
    std::vector<cv::KeyPoint> keypoints;
    cv::Mat frame, calibrationResult;
    cv::Mat cameraMatrix, distCoeffs, rotationMatrix, translationVector, inv_H;
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud;

    // Functions
    void writeToCsv(const char *filename, std::vector<cv::KeyPoint> keypoints);
```

```
    std::vector<cv::Point2f> getImagePointsFrom(std::vector<cv::KeyPoint> keypoints);
    std::string intToString(int input);
    void numberKeyPoints(cv::Mat &image, std::vector<cv::KeyPoint> keypoints);
    std::vector<cv::Point3f> readObjectPointsFrom(const char *filename);
    void readCameraParameters(std::string filename);
    void writeScannerParameters(std::string filename,
                                cv::Mat cameraMatrix,
                                cv::Mat distCoeffs,
                                cv::Mat rotationMatrix,
                                cv::Mat translationVector);
    bool captureCalibrationImage();
    bool performCalibration();
    double deg2rad(double degrees);
    cv::Mat RotateZ(double z_deg);
    cv::Mat calculateCoordinate(cv::Point2d imagePoint);
    bool calculatePointCloud();
    void displayBuffer(std::string window_name,
                       std::vector<cv::Mat> buffer,
                       double frame_rate);
};

#endif // LASERSCANNER_HPP
```

Listing B.10: Implementation of the laser scanner class.

```
#include "laserscanner.hpp"

LaserScanner::LaserScanner(std::string sim_data_filename) : cloud(new pcl::PointCloud<pcl::PointXYZ>)
{
    simulation = true;
    cv::FileStorage fs(sim_data_filename, cv::FileStorage::READ);
    fs["CameraMatrix"] >> cameraMatrix;
    fs["RotationMatrix"] >> rotationMatrix;
    fs["TranslationVector"] >> translationVector;
    fs.release();

    cv::Mat B(3,3, CV_64F);                      //       |r12 r13 tx|
    rotationMatrix.col(1).copyTo(B.col(0));      //  B = |r22 r23 ty| , when X = 0
    rotationMatrix.col(2).copyTo(B.col(1));      //       |r32 r33 tz|
    translationVector.col(0).copyTo(B.col(2));//
    cv::Mat H = cameraMatrix*B; // equation #, H = AB
    inv_H = H.inv(); // H^(-1)
}

LaserScanner::LaserScanner(CameraParameters params) : cloud(new pcl::PointCloud<pcl::PointXYZ>)
{
    simulation = false;
    camera.setId(params.id);
    camera.setFrameRate(params.frame_rate);
    camera.setFrameHeight(params.frame_height);
    camera.setFrameWidth(params.frame_width);

    imgProcessor.loadCalibrationDataFrom("calibration_data.xml");
}

bool LaserScanner::ping()
{
    if (camera.ping() && controller.ping()) {
        return true;
    } else {
        return false;
    }
}

void LaserScanner::setScannerParametersFromFile(std::string filename)
{
    cv::FileStorage fs(filename, cv::FileStorage::READ);
    fs["CameraMatrix"] >> cameraMatrix;
    fs["DistCoeffs"] >> distCoeffs;
```

```
    fs["RotationMatrix"] >> rotationMatrix;
    fs["TranslationVector"] >> translationVector;
    fs.release();

    // Calculate inv(H) required for equations #, #, and # here instead of
    // repeatedly in calulateCoordinate(point(u,v))
    cv::Mat B(3,3, CV_64F);                    //      |r12 r13 tx|
    rotationMatrix.col(1).copyTo(B.col(0));    //  B = |r22 r23 ty| , when X = 0
    rotationMatrix.col(2).copyTo(B.col(1));    //      |r32 r33 tz|
    translationVector.col(0).copyTo(B.col(2));//
    cv::Mat H = cameraMatrix*B; // equation #, H = AB
    inv_H = H.inv(); // H^(-1)

    imgProcessor.setCameraMatrix(cameraMatrix);
    imgProcessor.setDistortionCoefficients(distCoeffs);
    return;
}

void LaserScanner::setSimulationImageDirectory(boost::filesystem::path directory)
{
    imgProcessor.readImagesInDirectory(directory);
}

bool LaserScanner::calibrate()
{
    if (!captureCalibrationImage()) return false;
    else if (!performCalibration()) return false;
    else return true;
}

void LaserScanner::saveCalibrationData(std::string image_filename,
                                       const char* imagePoints_filename,
                                       std::string calibrationData_filename)
{
    cv::imwrite(image_filename, calibrationResult);
    writeToCsv(imagePoints_filename, keypoints);
    writeScannerParameters(calibrationData_filename,
                           cameraMatrix,
                           distCoeffs,
                           rotationMatrix,
                           translationVector);
}

bool LaserScanner::Scan(int channel, int threshold_method)
{
    try {
        if (simulation) {
            if (!imgProcessor.Threshold(channel, threshold_method)) {
                throw "Threshold operation failed.";
            } else if (!imgProcessor.Thin()) {
                throw "Thinning operation failed.";
            } else if (!calculatePointCloud()) {
                throw "Point cloud calculation failed.";
            } else {
                return true;
            }
        } else {
            boost::thread cameraThread(&Camera::capture, &camera);
            boost::thread controllerThread(&Controller::rotateFull, &controller);
            while (!controller.sentStartCaptureSignal());
            camera.startCapture();
            while (controller.sentStartCaptureSignal());
            cameraThread.interrupt();
            cameraThread.join();
            controllerThread.join();
            if (!imgProcessor.Undistort(camera)) {
                throw "Undistort operation failed.";
            } else if (!imgProcessor.Threshold(channel, threshold_method)) {
                throw "Threshold operation failed.";
            } else if (!imgProcessor.Thin()) {
                throw "Thinning operation failed.";
            } else if (!calculatePointCloud()) {
                throw "Point cloud calculation failed.";
```

```
            } else {
                return true;
            }
        }
    } catch (const char* msg) {
        std::cerr << "Exception: " << msg << std::endl;
        return false;
    }
}

void LaserScanner::showPointCloud()
{
    try {
        if (cloud->empty()) {
            throw "Cannot display empty point cloud!";
        } else {
            std::cout << "Displaying point cloud ............ ";
            std::cout << std::flush;
            pcl::visualization::PCLVisualizer viewer;
            if (simulation) {
                viewer.setWindowName("Point cloud of simulated scan");
            } else {
                viewer.setWindowName("Point cloud of scan");
            }
            viewer.addPointCloud<pcl::PointXYZ>(cloud, "cloud");
            viewer.spin();
            viewer.close();
            std::cout << "Complete." << std::endl;
            return;
        }
    } catch (const char* msg) {
        std::cerr << "Exception: " << msg << std::endl;
        return;
    }
}

void LaserScanner::displayImageBuffers()
{
    double frame_rate;
    if (!simulation) {
        frame_rate = camera.RGB_BUFFER.size()/controller.ROTATION_TIME;
        displayBuffer("Captured frames (undistorted)", imgProcessor.UNDISTORTED_IMAGES
            , frame_rate);
    } else {
        frame_rate = 30;
        displayBuffer("Blender simulation", imgProcessor.UNDISTORTED_IMAGES,
            frame_rate);
    }

    displayBuffer("Frames after segmentation", imgProcessor.THRESHOLD_IMAGES,
        frame_rate);
    displayBuffer("Frames after thinning", imgProcessor.THINNED_IMAGES, frame_rate);
}

void LaserScanner::savePointCloudToFile(std::string filename)
{
    try {
        if (cloud->empty()) {
            throw "Cannot save empty cloud to file";
        } else {
            std::cout << "Saving point cloud ................ ";
            std::cout << std::flush;
            pcl::io::savePCDFile(filename, *cloud);
            std::cout << "Complete." << std::endl;
            return;
        }
    } catch (const char* msg) {
        std::cerr << "Exception: " << msg << std::endl;
        return;
    }
}

```

```
void LaserScanner::writeToCsv(const char *filename, std::vector<cv::KeyPoint>
    keypoints)
{
    std::ofstream fout;
    fout.open(filename);

    for (int i = 0; i < keypoints.size(); i++) {
        fout << keypoints[i].pt.x
             << ","
             << keypoints[i].pt.y
             << std::endl;
    }
    fout.close();
    return;
}

std::vector<cv::Point2f> LaserScanner::getImagePointsFrom(std::vector<cv::KeyPoint>
    keypoints)
{
    std::vector<cv::Point2f> imagePoints;
    for (int i = 0; i < keypoints.size(); i++) {
        imagePoints.push_back(keypoints[i].pt);
    }
    return imagePoints;
}

std::string LaserScanner::intToString(int input)
{
    std::ostringstream convert;
    convert << input;
    return convert.str();
}

void LaserScanner::numberKeyPoints(cv::Mat &image, std::vector<cv::KeyPoint> keypoints
    )
{
    for (int i = 0; i < keypoints.size(); i++) {
        cv::putText(image,
                    intToString(i + 1),
                    keypoints[i].pt + cv::Point2f(5,5),
                    CV_FONT_HERSHEY_PLAIN,
                    0.7,
                    cv::Scalar(0,255,0),
                    1,
                    8,
                    false);
    }
}

std::vector<cv::Point3f> LaserScanner::readObjectPointsFrom(const char *filename)
{
    try
    {
        std::vector<cv::Point3f> objectPoints;
        std::ifstream file(filename);
        std::string line;
        while(std::getline(file, line))
        {
            std::stringstream ss(line);
            std::string X_string, Y_string, Z_string;
            float X, Y, Z;
            while(std::getline(ss, X_string, ','))
            {
                std::stringstream Xs(X_string);
                Xs >> X;

                std::getline(ss, Y_string, ',');
                std::stringstream Ys(Y_string);
                Ys >> Y;

                std::getline(ss, Z_string, ',');
                std::stringstream Zs(Z_string);
                Zs >> Z;
```

```
            }
            objectPoints.push_back(cv::Point3f(X, Y, Z));
        }
        return objectPoints;
    }
    catch(std::exception &e)
    {
        std::cerr << "Error: " << e.what() << std::endl;
    }
}

void LaserScanner::readCameraParameters(std::string filename)
{
    cv::FileStorage fs(filename, cv::FileStorage::READ);
    fs["CameraMatrix"] >> cameraMatrix;
    fs["DistCoeffs"] >> distCoeffs;
    fs.release();
}

void LaserScanner::writeScannerParameters(std::string filename, cv::Mat cameraMatrix,
    cv::Mat distCoeffs, cv::Mat rotationMatrix, cv::Mat translationVector)
{
    cv::FileStorage fs(filename, cv::FileStorage::WRITE);
    fs << "CameraMatrix" << cameraMatrix;
    fs << "DistCoeffs" << distCoeffs;
    fs << "RotationMatrix" << rotationMatrix;
    fs << "TranslationVector" << translationVector;
    fs.release();
}

bool LaserScanner::captureCalibrationImage()
{
    try {
        cv::VideoCapture cap(camera.ID);
        if (!cap.isOpened()) {
            throw "Device not found!";
        } else {
            cap.set(CV_CAP_PROP_FRAME_HEIGHT, 600.00);
            cap.set(CV_CAP_PROP_FRAME_WIDTH, 800.00);
            //cv::Mat frame, result, temp;

            // Blob detector parameters
            cv::SimpleBlobDetector::Params params;
            params.filterByInertia = true;
            params.minInertiaRatio = 0.01;
            params.filterByCircularity = true;
            params.minCircularity = 0.01;
            params.filterByConvexity = true;
            params.minConvexity = 0.01;

#if OPENCV_VERSION == 300 // OpenCV 3.0.0
            cv::SimpleBlobDetector detector;
            detector.create(params);
#else
            cv::SimpleBlobDetector detector(params);
#endif


            while (1) {
                cap >> frame;
                cv::cvtColor(frame, frame, CV_BGR2GRAY);
                detector.detect(frame, keypoints);
                cv::drawKeypoints(frame, keypoints, calibrationResult, cv::Scalar
                    (0,0,255));
                numberKeyPoints(calibrationResult, keypoints);
                cv::imshow("Keypoints", calibrationResult);
                char c = cv::waitKey(33);
                if (c == 27 || keypoints.size() == 23)
                    break;
            }
            cap.release();
            cv::destroyAllWindows();
            std::cout << "Press any to quit ..." << std::endl;
```

```
            cv::imshow("Final Result", calibrationResult);
            cv::waitKey();
            cv::destroyAllWindows();
            return true;
        }
    } catch (const char* msg) {
        std::cerr << "Exception: " << msg << std::endl;
        return false;
    } catch (cv::Exception &e) {
        std::cerr << "Exception: " << e.what() << std::endl;
        return false;
    }
}

bool LaserScanner::performCalibration()
{
    try {
        // Blob detector parameters (Describes dots to be found in calibration pattern)
        cv::SimpleBlobDetector::Params params;
        params.filterByInertia = true;
        params.minInertiaRatio = 0.01;
        params.filterByCircularity = true;
        params.minCircularity = 0.01;
        params.filterByConvexity = true;
        params.minConvexity = 0.01;

#if OPENCV_VERSION == 300 // OpenCV 3.0.0
        cv::SimpleBlobDetector detector;
        detector.create(params);
#else
        cv::SimpleBlobDetector detector(params);
#endif

        cv::Mat temp;
        readCameraParameters("/home/ivan/MEng/C++/Shared Files/calibration_data.xml");
        cv::Mat rotationVector(1, 3, CV_64F);

        if (cameraMatrix.empty()) {
            throw "Camera Matrix is empty. Perform a camera calibration first.";
        }

        if (distCoeffs.empty()) {
            throw "No distortion coefficients. Perform a camera calibration first.";
        }

        cv::undistort(frame, temp, cameraMatrix, distCoeffs);
        detector.detect(temp, keypoints);

        if (keypoints.size() != 23) {
            throw "Not enough keypoints found to perform calibration!";
        } else {
            std::vector<cv::Point3f> objectPoints = readObjectPointsFrom("/home/ivan/MEng/C++/Shared Files/objectPoints.csv");
            std::vector<cv::Point2f> imagePoints = getImagePointsFrom(keypoints);

            // Finds R and t (camera pose w.r.t. object coordinate system, i.e. calibration)
            cv::solvePnP(objectPoints,
                         imagePoints,
                         cameraMatrix,
                         distCoeffs,
                         rotationVector,
                         translationVector);

            cv::Rodrigues(rotationVector, rotationMatrix); // convert rotation vector to rotation matrix
            std::cout << "Calculation complete." << std::endl;
            return true;
        }
    } catch(const char* msg) {
        std::cerr << "Exception: " << msg << std::endl;
        return false;
```

```
    }
}

double LaserScanner::deg2rad(double degrees)
{
    return degrees*PI/180;
}

cv::Mat LaserScanner::RotateZ(double z_deg)
{
    double phi = deg2rad(z_deg);
    cv::Mat RotZ = (cv::Mat_<double>(3,3) << cos(phi),   -sin(phi),  0,
                                             sin(phi),   cos(phi),   0,
                                             0,          0,          1);
    return RotZ;
}

cv::Mat LaserScanner::calculateCoordinate(cv::Point2d imagePoint)
{
    cv::Mat m = (cv::Mat_<double>(3,1) << imagePoint.x, imagePoint.y, 1); // m = [u; v; 1]
    cv::Mat q = inv_H*m;            // equation #
    double s = 1/q.at<double>(2,0); // equation #
    double X = 0; // laser plane coincident with YZ-plane , X = 0
    double Y = s*q.at<double>(0,0); // equation #
    double Z = s*q.at<double>(1,0); // equation #

    cv::Mat M = (cv::Mat_<double>(3,1) << X, Y, Z);
    return M; // return M = [X=0; Y; Z]
}

bool LaserScanner::calculatePointCloud()
{
    if (imgProcessor.IMAGE_POINTS.empty()) {
        std::cerr << "There are no image points to calculate a point cloud from!" << std::endl;
        return false;
    } else {
        std::cout << "Calculating point cloud ........... ";
        std::cout << std::flush;

        //boost::chrono::system_clock::time_point start = boost::chrono::system_clock::now();
        for(int i = 0; i < imgProcessor.IMAGE_POINTS.size(); i++)
        {
            cv::Mat RotZ;
            if (simulation) {
                RotZ = RotateZ(-360.0*i/imgProcessor.IMAGE_POINTS.size());
            } else {
                RotZ = RotateZ((-360.0)*(camera.TIME_STAMPS[i]-camera.TIME_STAMPS[0])/controller.ROTATION_TIME);
            }
            for(int j = 0; j < imgProcessor.IMAGE_POINTS.at(i).size(); j++)
            {
                cv::Mat Point = calculateCoordinate(imgProcessor.IMAGE_POINTS.at(i).at(j));
                cv::Mat Result = RotZ*Point;
                if (simulation && Result.at<double>(2) > 2.5) {
                    cloud->push_back(pcl::PointXYZ(Result.at<double>(0)/1000,   // divide by 1000 to go from milimeters to meters
                                                   Result.at<double>(1)/1000,   //
                                                   Result.at<double>(2)/1000)); // (base unit for Point Cloud Library is m)
                }
                if (!simulation) {
                    cloud->push_back(pcl::PointXYZ(Result.at<double>(0)/1000,   // divide by 1000 to go from milimeters to meters
                                                   Result.at<double>(1)/1000,   //
                                                   Result.at<double>(2)/1000)); // (base unit for Point Cloud Library is m)
                }
```

```
                Point.release();
                Result.release();
            }
            RotZ.release();
        }
        //boost::chrono::system_clock::time_point stop = boost::chrono::system_clock::
            now();
        //boost::chrono::duration<double> duration = stop - start;
        //CALCULATION_TIME = double(duration.count());
        std::cout << "Complete." << std::endl;
        return true;
    }
}

void LaserScanner::displayBuffer(std::string window_name,
                                 std::vector<cv::Mat> buffer,
                                 double frame_rate)
{
    cv::namedWindow(window_name, CV_WINDOW_AUTOSIZE);
    cv::moveWindow(window_name, 10, 10);
    for (int i = 0; i < buffer.size(); ++i) {
        cv::imshow(window_name, buffer[i]);
        cv::waitKey(int(1000/frame_rate));
    }
    cv::destroyWindow(window_name);
}
```

B.1.6 Arduino Uno Controller

Listing B.11: Code for the Areduino Uno Controller.

```
char message = 0;
int stepPin = 12;
int directionPin = 11;
int sleepPin = 10;
int laserPin = 9;
int laserStatus = 0;
int halfPeriod = 4; // milliseconds
int fullRotation = 1600; // steps @ 8x microstepping


void setup()
{
  Serial.begin(9600);
  pinMode(stepPin, OUTPUT);
  pinMode(directionPin, OUTPUT);
  pinMode(sleepPin, OUTPUT);
  pinMode(laserPin, OUTPUT);

  digitalWrite(stepPin, LOW);
  digitalWrite(directionPin, LOW);
  digitalWrite(sleepPin, LOW);
  digitalWrite(laserPin, LOW);
}

void loop()
{
  if (Serial.available() > 0) {
    message = Serial.read();
  }

  if (message == '0') {
    toggleLaser();
    clearMessage();
  }
  if (message == '1') {
    replyToPing();
    clearMessage();
  }
  if (message == '2') {
    rotateStepper();
    clearMessage();
  }
}

void toggleLaser()
{
  if (laserStatus == 0) {
    digitalWrite(laserPin, HIGH);
    laserStatus = 1;
  } else {
    digitalWrite(laserPin, LOW);
    laserStatus = 0;
  }
  return;
}

void clearMessage()
{
  message = ' ';
  return;
}

void replyToPing()
{
  delay(200);
  Serial.println("Received ping");
  return;
```

```
}

void rotateStepper()
{
  digitalWrite(sleepPin, HIGH);
  digitalWrite(laserPin, HIGH);
  delay(5000); // Give camera time to adjust to ambient light conditions.
  Serial.println("Stepper started"); // Tells PC to start capturing images.
  for (int i = 0; i < fullRotation; i++) {
    digitalWrite(stepPin, HIGH);
    delay(halfPeriod);
    digitalWrite(stepPin, LOW);
    delay(halfPeriod);
  }
  Serial.println("Stepper complete");
  delay(500);
  digitalWrite(sleepPin, LOW);
  digitalWrite(laserPin, LOW);
  laserStatus = 0;
  return;
}
```

B.1.7 3D Laser Scanner: main.cpp

Listing B.12: Main file for 3D laser scanner.

```
#include "laserscanner.hpp"
#include <boost/program_options.hpp>

int main(int argc, char *argv[])
{
    int camera_id = 0;
    int threshold_method = 2;
    bool show_point_cloud = false;
    bool show_buffers = false;
    bool sim = false;
    fs::path sim_images_dir = "/home/ivan/MEng/Blender/Images";

    boost::program_options::options_description description("\nUsage: ./3d-scanner [options]\n\nAllowed options");
    description.add_options()
            ("help", "produces this help information")
            ("simulate-scanner", "Simulates a 3d scanner (specify path to images with --image-dir")
            ("image-dir", boost::program_options::value<fs::path>(&sim_images_dir), "Specify input image directory for scanner simulation")
            ("cam-id", boost::program_options::value<int>(&camera_id), "Camera ID (default=0)")
            ("thresh", boost::program_options::value<int>(&threshold_method),
             "Threshold method (1 or 2, default=2)")
            ("show-cloud", "Show point cloud")
            ("show-buffers", "Show image buffers")
            ("calibrate-camera", "Calibrate the scanner's camera")
            ("calibrate-scanner", "Calibrate the scanner (Perform after calibrating camera)")
    ;

    boost::program_options::variables_map vm;
    boost::program_options::store(boost::program_options::parse_command_line(argc, argv, description), vm);
    boost::program_options::notify(vm);

    if (vm.count("help")) {
        std::cout << description << std::endl;
        return 0;
    }

    if (vm.count("calibrate-camera")) {
        Camera camera(camera_id, 30, 600, 800);
        if (!camera.calibrate()) {
            return -1;
        } else {
            camera.saveCalibrationData();
            return 0;
        }
    }

    if (vm.count("calibrate-scanner")) {
        CameraParameters params;
        params.id = camera_id;
        params.frame_rate = 30;
        params.frame_height = 600;
        params.frame_width = 800;
        LaserScanner scanner(params);
        if (!scanner.calibrate()) {
            return -1;
        } else {
            scanner.saveCalibrationData();
            return 0;
        }
    }
    if (vm.count(("simulate-scanner"))) {
        sim = true;
```

```
    }

    if (vm.count("show-cloud")) {
        show_point_cloud = true;
    }

    if (vm.count("show-buffers")) {
        show_buffers = true;
    }

    if (sim) {
        std::cout << "Running scanner simulation . . ." << std::endl;
        if (!fs::exists(sim_images_dir)) {
            std::cerr << "Image directory does not exist!" << std::endl;
            return -1;
        }

        if (sim_images_dir.empty()) {
            std::cerr << "Image directory is empty!" << std::endl;
            return -1;
        }
        LaserScanner scanner("/home/ivan/MEng/C++/Shared Files/scanner_sim_data.xml");
        scanner.setSimulationImageDirectory(sim_images_dir);
        scanner.Scan(RED_CHANNEL, threshold_method);
        if (show_buffers) {
            scanner.displayImageBuffers();
        }
        if (show_point_cloud) {
            scanner.showPointCloud();
        }
        scanner.savePointCloudToFile("/home/ivan/MEng/C++/Temp/scan.pcd");
        return 0;
    } else {
        CameraParameters params;
        params.id = camera_id;
        params.frame_rate = 30;
        params.frame_height = 600;
        params.frame_width = 800;
        LaserScanner scanner(params);

        if (!scanner.ping()) {
            return -1;
        } else {
            scanner.setScannerParametersFromFile("/home/ivan/MEng/C++/Shared Files/
                scanner_calibration_data.xml");
        }
        if (!scanner.Scan(RED_CHANNEL, threshold_method)) {
            return -1;
        }
        if (show_buffers) {
            scanner.displayImageBuffers();
        }
        if (show_point_cloud) {
            scanner.showPointCloud();
        }
        scanner.savePointCloudToFile("/home/ivan/MEng/C++/Temp/scan.pcd");
        return 0;
    }
}
```

B.2 Object Recognition Related Code

B.2.1 Data Generation for Training and Testing

Listing B.13: Main file for data generation

```
#include <floatfann.h>
#include <fann_cpp.h>
#include <boost/chrono.hpp>
#include <boost/random.hpp>
#include <cmath>
#include <ctime>
#include <Eigen/Core>
#include <fstream>
#include <pcl/common/transforms.h>
#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/filters/filter.h>
#include <pcl/features/vfh.h>
#include <pcl/features/normal_3d.h>
#include <pcl/kdtree/kdtree_flann.h>
#include <pcl/surface/mls.h>
#include <pcl/visualization/pcl_visualizer.h>
#include <pcl/visualization/histogram_visualizer.h>

const float PI = atan(1.0)*4;

boost::random::mt19937 generator(std::time(0));

float noise(void)
{
    boost::random::uniform_real_distribution<float> num(-0.0005f, 0.0005f);
    return num(generator);
}

float randomAngle(void)
{
    boost::random::uniform_real_distribution<float> num(-PI, PI);
    return num(generator);
}

float randomDist(void)
{
    boost::random::uniform_real_distribution<float> num(-0.010f, 0.010f);
    return num(generator);
}

void write_data2File(const char *filename,
                     std::vector< std::vector<float> > inputs,
                     std::vector< std::vector<float> > targets)
{
    std::ofstream fout;
    fout.open(filename);
    fout << inputs.size() << " " << inputs[0].size() << " " << targets[0].size()
         << std::endl;

    for(int i = 0; i < inputs.size(); i++)
    {
        for(int j = 0; j < inputs[i].size(); j++)
        {
            fout << inputs[i][j] << " ";
        }
        fout << std::endl;

        for(int j = 0; j < targets[i].size(); j++)
        {
            fout << targets[i][j] << " ";
```

```
        }
        fout << std::endl;
    }
    fout.close();
    return;
}

void VoxelGrid(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud,
               pcl::PointCloud<pcl::PointXYZ>::Ptr &filtered_cloud)
{
    pcl::VoxelGrid<pcl::PointXYZ> sor;
    sor.setInputCloud (cloud);
    sor.setLeafSize (0.001f, 0.001f, 0.001f);
    sor.filter (*filtered_cloud);
    return;
}

void NormalEstimate(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud,
                    pcl::PointCloud<pcl::PointNormal> &CloudNormals)
{
    pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> ne;
    ne.setInputCloud (cloud);
    pcl::search::KdTree<pcl::PointXYZ>::Ptr
            tree (new pcl::search::KdTree<pcl::PointXYZ> ());
    ne.setSearchMethod (tree);
    pcl::PointCloud<pcl::Normal>::Ptr cloud_normals (new pcl::PointCloud<pcl::Normal>)
        ;
    ne.setRadiusSearch (0.006);
    ne.compute (*cloud_normals);

    pcl::concatenateFields(*cloud, *cloud_normals, CloudNormals);

    std::vector<int> indices;
    pcl::removeNaNFromPointCloud(CloudNormals, CloudNormals, indices);
    return;
}

void ViewpointFeatureHistogram(pcl::PointCloud<pcl::PointNormal> mls_points,
                               pcl::PointCloud<pcl::VFHSignature308>::Ptr &vfhs)
{
    pcl::VFHEstimation<pcl::PointNormal, pcl::PointNormal, pcl::VFHSignature308> vfh;
    vfh.setInputCloud (mls_points.makeShared());
    vfh.setInputNormals (mls_points.makeShared());
    pcl::search::KdTree<pcl::PointNormal>::Ptr
            tree (new pcl::search::KdTree<pcl::PointNormal>);
    vfh.setSearchMethod (tree);
    vfh.setNormalizeBins(true);
    vfh.compute (*vfhs);
    return;
}

void addRandomNoise(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud,
                    pcl::PointCloud<pcl::PointXYZ>::Ptr &noisy_cloud)
{

    noisy_cloud->clear();
    noisy_cloud->resize(cloud->size());
    for(int i = 0; i < cloud->points.size(); i++)
    {
        noisy_cloud->points[i].x = cloud->points[i].x + noise();
        noisy_cloud->points[i].y = cloud->points[i].y + noise();
        noisy_cloud->points[i].z = cloud->points[i].z + noise();
    }
    return;
}

void addRandomTransRot(pcl::PointCloud<pcl::PointXYZ>::Ptr &cloud)
{
    float angle_Z = randomAngle();
    float x_dist = randomDist();
    float y_dist = randomDist();
    Eigen::Matrix4f TransRot;
    TransRot << cos(angle_Z),  -sin(angle_Z),  0,  x_dist,
```

```
                sin(angle_Z),   cos(angle_Z),   0,  y_dist,
                    0,              0,          1,    0,
                    0,              0,          0,    1;

    pcl::transformPointCloud(*cloud, *cloud, TransRot);
    return;
}

std::vector<int> getParams(int argc, char *argv[])
{
    std::vector<int> result; result.clear();

    if(argc < 2)
    {
        result.push_back(100);  // num_data       [0]
        result.push_back(135);  // num_inputs     [1]
        result.push_back(8);    // num_outputs    [3]
    }
    else
    {
        for(int i = 1; i < argc; i++)
        {
            std::istringstream iss(argv[i]);
            float val;
            if(iss >> val) result.push_back(val);
        }
    }
    return result;
}

void saveData(const char* filename, std::vector< std::vector<float> > data)
{
    std::cout << "Saving " << filename << " ..." << std::endl;
    int rows = data.size();
    int cols = data[0].size();

    std::ofstream fout;
    fout.open(filename);

    for(int i = 0; i < rows; i++)
    {
        for(int j = 0; j < cols; j++)
        {
            fout << data[i][j] << " ";
        }
        fout << "\n";
    }
    fout.close();
    std::cout << "Saving complete." << std::endl;
    return;
}

int main (int argc, char** argv)
{
    /* Declare and initialize variables ***************************************/
    std::vector<int> arguments = getParams(argc, argv);
    const unsigned int num_data = arguments[0];
    const unsigned int num_inputs = arguments[1];
    const unsigned int num_outputs = arguments[2];

    // Allocate memory for point clouds
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud1 (new pcl::PointCloud<pcl::PointXYZ> ())
        ;
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud2 (new pcl::PointCloud<pcl::PointXYZ> ())
        ;
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud3 (new pcl::PointCloud<pcl::PointXYZ> ())
        ;
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud4 (new pcl::PointCloud<pcl::PointXYZ> ())
        ;
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud5 (new pcl::PointCloud<pcl::PointXYZ> ())
        ;
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud6 (new pcl::PointCloud<pcl::PointXYZ> ())
        ;
```

```
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud7 (new pcl::PointCloud<pcl::PointXYZ> ());
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud8 (new pcl::PointCloud<pcl::PointXYZ> ());

    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud1a (new pcl::PointCloud<pcl::PointXYZ> ());
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud2a (new pcl::PointCloud<pcl::PointXYZ> ());
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud3a (new pcl::PointCloud<pcl::PointXYZ> ());
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud4a (new pcl::PointCloud<pcl::PointXYZ> ());
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud5a (new pcl::PointCloud<pcl::PointXYZ> ());
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud6a (new pcl::PointCloud<pcl::PointXYZ> ());
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud7a (new pcl::PointCloud<pcl::PointXYZ> ());
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud8a (new pcl::PointCloud<pcl::PointXYZ> ());

    // Load point clouds from file into memory
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object1.pcd", *cloud1);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object2.pcd", *cloud2);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object3.pcd", *cloud3);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object4.pcd", *cloud4);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object5.pcd", *cloud5);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object6.pcd", *cloud6);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object7.pcd", *cloud7);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object8.pcd", *cloud8);

    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object1a.pcd", *cloud1a);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object2a.pcd", *cloud2a);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object3a.pcd", *cloud3a);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object4a.pcd", *cloud4a);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object5a.pcd", *cloud5a);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object6a.pcd", *cloud6a);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object7a.pcd", *cloud7a);
    pcl::io::loadPCDFile("/home/ivan/Applications/bin/TrainingObjects/Object8a.pcd", *cloud8a);

    // Accumulating clouds
    std::vector< pcl::PointCloud< pcl::PointXYZ >::Ptr > clouds;
    clouds.push_back(cloud1);
    clouds.push_back(cloud2);
    clouds.push_back(cloud3);
    clouds.push_back(cloud4);
    clouds.push_back(cloud5);
    clouds.push_back(cloud6);
    clouds.push_back(cloud7);
    clouds.push_back(cloud8);

    clouds.push_back(cloud1a);
    clouds.push_back(cloud2a);
    clouds.push_back(cloud3a);
    clouds.push_back(cloud4a);
    clouds.push_back(cloud5a);
```

```
    clouds.push_back(cloud6a);
    clouds.push_back(cloud7a);
    clouds.push_back(cloud8a);
    std::cout << clouds.size() << std::endl;
    /**********************************************************************/

    /* Generate input vectors from point clouds ******************************/
    std::cout << "Generating input values ...\t" << std::flush;
    std::vector< std::vector<float> > training_inputs;
    for(int i = 0; i < num_data; i++)
    {
        for(int j = 0; j < 2*num_outputs; j++)
        {
            //
            pcl::PointCloud<pcl::PointXYZ>::Ptr
                    reduced_cloud (new pcl::PointCloud<pcl::PointXYZ> ());
            pcl::PointCloud<pcl::PointNormal> point_normal;
            pcl::PointCloud<pcl::VFHSignature308>::Ptr
                    vfhs (new pcl::PointCloud<pcl::VFHSignature308> ());
            std::vector<float> signature;

            //
            if(i == 0) // don't add noise
            {
                VoxelGrid(clouds[j], reduced_cloud);
                NormalEstimate(reduced_cloud, point_normal);
            }
            else // add noise
            {
                pcl::PointCloud<pcl::PointXYZ>::Ptr
                        noisy_cloud (new pcl::PointCloud<pcl::PointXYZ> ());
                addRandomNoise(clouds[j], noisy_cloud);
                VoxelGrid(noisy_cloud, reduced_cloud);
                addRandomTransRot(reduced_cloud);
                NormalEstimate(reduced_cloud, point_normal);
            }

            ViewpointFeatureHistogram(point_normal, vfhs);
            for(int k = 0; k < num_inputs; k++)
            {
                signature.push_back(vfhs->points[0].histogram[k]);
            }
            training_inputs.push_back(signature);
        }
    }
    std::cout << "Complete." << std::endl;
    /**********************************************************************/

    /* Generate corresponding target values ***********************************/
    std::cout << "Generating target values ...\t" << std::flush;
    std::vector< std::vector<float> > training_targets;
    int placeholder = 0;
    for(int i = 0; i < 2*num_data*num_outputs; i++)
    {
        std::vector<float> targets; targets.clear();
        for(int j = 0; j < num_outputs; j++)
        {
            if(j == placeholder) targets.push_back(1.0f);
            else targets.push_back(-1.0f);
        }
        training_targets.push_back(targets);
        placeholder++;
        if(placeholder == num_outputs)
            placeholder = 0;
        else continue;
    }
    std::cout << "Complete." << std::endl;
    /**********************************************************************/

    /* Write data to file ****************************************************/
    std::cout << "Writing data to file ...\t" << std::flush;
    write_data2File("training.data", training_inputs, training_targets);
    std::cout << "Complete." << std::endl;
```

```
    /*********************************************************************/
    return 0;
}
```

B.2.2 Training and Testing

Listing B.14: Main file for neural network training .

```
#include <ctime>
#include <boost/chrono.hpp>
#include <boost/thread.hpp>
#include <floatfann.h>
#include <fann_cpp.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <cmath>

void writeMSE_Errors(const char* filename, std::vector<float> TrainingMSE,
              std::vector<float> TestingMSE)
{
    std::ofstream fout;
    fout.open(filename);
    fout << "Training Errors, Testing Errors" << std::endl;
    for(int i = 0; i < TrainingMSE.size(); i++)
    {
        fout << TrainingMSE[i] << "," << TestingMSE[i] << std::endl;
    }
    fout.close();
    return;
}

void writeTrainingDuration(const char* filename, float MSE, int epochs,
                           double training_duration)
{
    std::ofstream fout;
    fout.open(filename);
    fout << MSE << " " << epochs << " " << training_duration << std::endl;
    fout.close();
    return;
}

std::vector<float> getParams(int argc, char *argv[])
{
    std::vector<float> result; result.clear();

    if(argc < 2)    // Default parameters
    {
        result.push_back(135.0f);   // num_input            [0]
        result.push_back(10.0f);    // num_hidden           [1]
        result.push_back(8.0f);     // num_output           [2]
        result.push_back(0.15f);    // learning_rate        [3]
        result.push_back(0.1f);     // learning_momentum    [4]
        result.push_back(1000);     // epochs               [5]
        result.push_back(0.0005f);  // desired_MSE          [6]
    }
    else
    {
        for(int i = 1; i < argc; i++)
        {
            std::istringstream iss(argv[i]);
            float val;
            if(iss >> val) result.push_back(val);
        }
    }
    return result;
}

int main(int argc, char *argv[])
{
    /* Declare and initialise some variables *********************************/
    boost::chrono::system_clock::time_point start;
    boost::chrono::system_clock::time_point stop;
    boost::chrono::duration<double> training_duration;// = stop - start;
```

```
    std::vector<float> arguments = getParams(argc, argv);
    const unsigned int num_layers = 3;
    const unsigned int num_input = (int)arguments[0];
    const unsigned int num_hidden = (int)arguments[1];
    const unsigned int num_output = (int)arguments[2];
    const float learning_rate = arguments[3];
    const float learning_momentum = arguments[4];
    const float desiredMSE = arguments[6];
    const unsigned int max_iterations = arguments[5];
    const float desiredGradient = 0.000001f;
    std::vector<float> TrainMSE; TrainMSE.clear();
    std::vector<float> TestMSE; TestMSE.clear();
    /**************************************************************************/

    /* Create Artificial Neural Network ***************************************/
    FANN::neural_net ANN;
    ANN.create_standard(num_layers, num_input, num_hidden, num_output);
    ANN.set_learning_rate(learning_rate);
    ANN.set_learning_momentum(learning_momentum);
    ANN.set_activation_steepness_hidden(1.0);
    ANN.set_activation_steepness_output(1.0);
    ANN.set_activation_function_hidden(FANN::SIGMOID_SYMMETRIC);
    ANN.set_activation_function_output(FANN::SIGMOID_SYMMETRIC);
    ANN.set_training_algorithm(FANN::TRAIN_BATCH);
    /**************************************************************************/

    /* Prepare training and testing data **************************************/
    //
    FANN::training_data training_data;
    training_data.read_train_from_file("training.data");
    training_data.shuffle_train_data();
    FANN::training_data testing_data;
    testing_data.read_train_from_file("testing.data");
    testing_data.shuffle_train_data();
    /**************************************************************************/

    /* Perform training and cross validation **********************************/
    std::cout << "Commence ANN training ...\t" << std::flush;
    //
    start = boost::chrono::system_clock::now();
    int gradientCount = 0;
    int mseDiffCount = 0;
    bool minimumGradient = false;
    bool minimumMSE = false;
    bool maximumEpochs = false;
    bool overfitting = false;
    ANN.randomize_weights(-0.5f, 0.5f);
    ANN.train_epoch(training_data);
    float trainingMSE = ANN.get_MSE();
    float testingMSE = ANN.test_data(testing_data);
    float MSEdiff = std::abs(testingMSE - trainingMSE);
    float previousMSEdiff = MSEdiff;
    float previousMSE = testingMSE;
    float gradient;
    TrainMSE.push_back(trainingMSE);
    TestMSE.push_back(testingMSE);
    int epoch = 1;
    for(int i = 1; i < max_iterations; i++)
    {
        epoch++;
        ANN.train_epoch(training_data);
        trainingMSE = ANN.get_MSE();
        testingMSE = ANN.test_data(testing_data);
        gradient = std::abs(testingMSE - previousMSE);
        MSEdiff = std::abs(testingMSE - trainingMSE);
        previousMSE = testingMSE;
        TrainMSE.push_back(trainingMSE);
        TestMSE.push_back(testingMSE);

        if(MSEdiff > previousMSEdiff)
        {
            ++mseDiffCount;
            previousMSEdiff = MSEdiff;
```

```
        }
        else
        {
            mseDiffCount = 0;
            previousMSEdiff = MSEdiff;
        }

        if(gradient <= desiredGradient)
        {
            ++gradientCount;
        }
        else
        {
            gradientCount = 0;
        }

        if(gradientCount == 20)
        {
            minimumGradient = true;
            break;
        }
        else if(testingMSE < desiredMSE)
        {
            minimumMSE = true;
            break;
        }
        else if(mseDiffCount == 20)
        {
            overfitting = true;
            break;
        }
        else continue;
    }
    if(epoch == max_iterations) maximumEpochs = true;
    stop = boost::chrono::system_clock::now();
    training_duration = stop - start;
    std::cout << "Complete." << std::endl;
    /***********************************************************************/

    /* Save ANN and MSEs *****************************************************/
    std::cout << "Saving ANN and MSEs ...\t\t" << std::flush;
    ANN.save("ANN.net");
    writeMSE_Errors("Errors.csv", TrainMSE, TestMSE);
    writeTrainingDuration("training_time.csv", trainingMSE,
                          epoch, double(training_duration.count()));
    std::cout << "Complete." << std::endl;
    /***********************************************************************/

    /* Exit report ***********************************************************/
    std::cout << "Training terminating condition: " << std::flush;
    if(maximumEpochs)
        std::cout << "Reached maximum epochs." << std::endl;
    else if(minimumGradient)
        std::cout << "Achieved minimum MSE gradient." << std::endl;
    else if(overfitting)
        std::cout << "Overfitting detected." << std::endl;
    else
        std::cout << "Achieved desired MSE." << std::endl;
    std::cout << "Training time: "
              << double(training_duration.count())
              << "seconds" << std::endl;
    /***********************************************************************/
    return 0;
}
```

Listing B.15: Main file for neural network validation.

```
#include <cstdio>
#include <floatfann.h>
#include <fann_cpp.h>
#include <fstream>
```

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>

int myRound(float val)
{
    int result;
    if(val < 0) result = -1;
    else result = 1;
    return result;
}

float class_ACC(int tp, int tn, int fp, int fn)
{
    return (float)100*(tp + tn)/(tp + tn + fp +fn);
}

float class_PPV(int tp, int fp)
{
    if(tp == 0) return 0.0f;
    else
        return (float)100*(tp)/(tp + fp);
}

float class_NPV(int tn, int fn)
{
    if(tn == 0) return 0.0f;
    else
        return (float)100*(tn)/(tn + fn);
}

float class_TPR(int tp, int fn)
{
    if(tp == 0) return 0.0f;
    else
        return (float)100*(tp)/(tp + fn);
}

float class_SPC(int tn, int fp)
{
    if(tn == 0) return 0.0f;
    else
        return (float)100*(tn)/(tn + fp);
}

float network_ACC(std::vector<int> tp, std::vector<int> tn,
                       std::vector<int> fp, std::vector<int> fn)
{
    float sum = 0;
    int I = tp.size();
    for(int i = 0; i < I; i++)
    {
        sum += class_ACC(tp[i], tn[i], fp[i], fn[i]);
    }
    return sum/I;
}

float network_PPV(std::vector<int> tp,std::vector<int> fp)
{
    float sum = 0;
    int I = tp.size();
    for(int i = 0; i < I; i++)
    {
        sum += class_PPV(tp[i], fp[i]);
    }
    return sum/I;
}

float network_NPV(std::vector<int> tn,std::vector<int> fn)
{
    float sum = 0;
    int I = tn.size();
```

```
    for(int i = 0; i < I; i++)
    {
        sum += class_PPV(tn[i], fn[i]);
    }
    return sum/I;
}

float network_TPR(std::vector<int> tp,std::vector<int> fn)
{
    float sum = 0;
    int I = tp.size();
    for(int i = 0; i < I; i++)
    {
        sum += class_TPR(tp[i], fn[i]);
    }
    return sum/I;
}

float network_SPC(std::vector<int> tn,std::vector<int> fp)
{
    float sum = 0;
    int I = tn.size();
    for(int i = 0; i < I; i++)
    {
        sum += class_SPC(tn[i], fp[i]);
    }
    return sum/I;
}

void writeResults_LaTeX(std::vector<int> tp,
                        std::vector<int> tn,
                        std::vector<int> fp,
                        std::vector<int> fn,
                        std::vector<float> acc,
                        std::vector<float> ppv,
                        std::vector<float> npv,
                        std::vector<float> tpr,
                        std::vector<float> spc,
                        int TP, int TN, int FP, int FN,
                        float ACC, float PPV, float NPV, float TPR, float SPC,
                        std::vector<float> MSE_epochs_time,
                        const char* filename)
{
    int I = acc.size();
    std::ofstream fout;
    fout.open(filename);

    fout << "Object,TP,TN,FP,FN,ACC,PPV,NPV,TPR,SPC,MSE,Epochs,Training Time" << std::endl;

    for(int i = 0; i < I; i++)
    {
        fout << i+1 << "," << std::setprecision(4)
             << tp[i] << ","
             << tn[i] << ","
             << fp[i] << ","
             << fn[i] << ","
             << acc[i] << ","
             << ppv[i] << ","
             << npv[i] << ","
             << tpr[i] << ","
             << spc[i] << ",-,-,-" << std::endl;
    }
    fout << "Total:,"
         << TP << ","
         << TN << ","
         << FP << ","
         << FN << ","
         << ACC << ","
         << PPV << ","
         << NPV << ","
         << TPR << ","
         << SPC << ","
```

```
        << MSE_epochs_time[0] << ","
        << MSE_epochs_time[1] << ","
        << MSE_epochs_time[2] << std::endl;
    fout.close();
    return;
}

void readCSV(const char* filename, std::vector<float> &MSE_epochs_time)
{
    MSE_epochs_time.clear();
    std::ifstream fin;
    fin.open(filename);
    float value;
    while(!fin.eof())
    {
        fin >> value;
        MSE_epochs_time.push_back(value);
    }
    fin.close();
    return;
}

void writeResults(std::vector<int> tp,
                  std::vector<int> tn,
                  std::vector<int> fp,
                  std::vector<int> fn,
                  std::vector<float> acc,
                  std::vector<float> ppv,
                  std::vector<float> npv,
                  std::vector<float> tpr,
                  std::vector<float> spc,
                  int TP, int TN, int FP, int FN,
                  float ACC, float PPV, float NPV, float TPR, float SPC,
                  std::vector<float> MSE_epochs_time,
                  const char* filename)
{
int I = acc.size();
std::ofstream fout;
fout.open(filename);

fout << "Object,TP,TN,FP,FN,ACC,PPV,NPV,TPR,SPC,MSE,Epochs,Training Time" << std::endl
    ;

for(int i = 0; i < I; i++)
{
  fout << i+1 << ","
       << tp[i] << ","
       << tn[i] << ","
       << fp[i] << ","
       << fn[i] << ","
       << acc[i] << ","
       << ppv[i] << ","
       << npv[i] << ","
       << tpr[i] << ","
       << spc[i] << ",-,-,-" << std::endl;
}
fout << "Total:,"
   << TP << ","
   << TN << ","
   << FP << ","
   << FN << ","
   << ACC << ","
   << PPV << ","
   << NPV << ","
   << TPR << ","
   << SPC << ","
   << MSE_epochs_time[0] << ","
   << MSE_epochs_time[1] << ","
   << MSE_epochs_time[2] << std::endl;
fout.close();
return;
}
```

```
int main()
{
    /* Declare and initialise some variables *********************************/
    FANN::training_data testing_data;
    testing_data.read_train_from_file("validation.data");
    const unsigned int num_output = testing_data.num_output_train_data();
    const unsigned int num_data = testing_data.length_train_data();
    FANN::neural_net ANN;
    ANN.create_from_file("ANN.net");

    std::vector<int> tp, fp, tn, fn;
    int TP = 0, TN = 0, FP = 0, FN = 0;

    // Performance measures
    std::vector<float> acc, ppv, npv, tpr, spc; // micro
    float ACC, PPV, NPV, TPR, SPC;              // macro

    for(int j = 0; j < num_output; j++)
    {
        tp.push_back(0);
        fp.push_back(0);
        tn.push_back(0);
        fn.push_back(0);
    }
    std::vector<float> MSE_epochs_time;
    readCSV("training_time.csv", MSE_epochs_time);
    /*************************************************************************/

    /* Determine True Pos'ves, True Negs, False Pos'ves, and False Negs *******/
    for(int i = 0; i < num_data; i++)
    {
        fann_type *result = ANN.run(testing_data.get_input()[i]);

        for(int j = 0; j < num_output; j++)
        {
            int output = myRound(result[j]);
            int target = (int)testing_data.get_output()[i][j];

            if(output == target && output > 0)
            {
                // We have a true positive
                ++tp[j];
                ++TP;
            }
            else if(output == target && output < 0)
            {
                // We have a true negative
                ++tn[j];
                ++TN;
            }
            else if(output > target)
            {
                // We have a false positive
                ++fp[j];
                ++FP;
            }
            else
            {
                // We must be left with a false negative
                ++fn[j];
                ++FN;
            }

        }
    }
    /*************************************************************************/

    /* Calculate Performance Measures ****************************************/
    for(int j = 0; j < num_output; j++)
    {
        acc.push_back(class_ACC(tp[j], tn[j], fp[j], fn[j]));
        ppv.push_back(class_PPV(tp[j], fp[j]));
        npv.push_back(class_NPV(tn[j], fn[j]));
```

```
            tpr.push_back(class_TPR(tp[j], fn[j]));
            spc.push_back(class_SPC(tn[j], fp[j]));
        }
        ACC = network_ACC(tp, tn, fp, fn);
        PPV = network_PPV(tp, fp);
        NPV = network_NPV(tn, fn);
        TPR = network_TPR(tp, fn);
        SPC = network_SPC(tn, fp);
        /**************************************************************************/

        /* Write test results to file ********************************************/
        writeResults(tp, tn, fp, fn,
                     acc, ppv, npv, tpr, spc,
                     TP, TN, FP, FN,
                     ACC, PPV, NPV, TPR, SPC,
                     MSE_epochs_time,
                     "validation_results.csv");

        writeResults_LaTeX(tp, tn, fp, fn,
                           acc, ppv, npv, tpr, spc,
                           TP, TN, FP, FN,
                           ACC, PPV, NPV, TPR, SPC,
                           MSE_epochs_time,
                           "validation_results_LaTeX.csv");
        /**************************************************************************/
        return 0;
}
```

B.2.3 Object Recognition

Listing B.16: ANN based 3D object recognition code.

```cpp
// Required headers from std library
#include <ctime>
#include <iomanip>
#include <iostream>
#include <sstream>

// Required headers from Boost library
#include <boost/chrono.hpp>
//#include <boost/random.hpp>
#include <boost/thread.hpp>

// Required headers from FANN library
#include <floatfann.h>
#include <fann_cpp.h>

// Required headers from Point Cloud Library
#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/filters/filter.h>
#include <pcl/features/normal_3d.h>
#include <pcl/features/vfh.h>
#include <pcl/kdtree/kdtree_flann.h>
#include <pcl/surface/mls.h>
#include <pcl/visualization/pcl_visualizer.h>
#include <pcl/visualization/histogram_visualizer.h>

// Define
#define INPUTS  135
#define CLASSES 8

// Declare and instantiate random number generator
boost::random::mt19937 generator(std::time(0));

float random_num(float min, float max)
{
    boost::random::uniform_real_distribution<float> num(min, max);
    return num(generator);
}

void VoxelGrid(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud,
               pcl::PointCloud<pcl::PointXYZ>::Ptr &filtered_cloud)
{
    pcl::VoxelGrid<pcl::PointXYZ> sor;
    sor.setInputCloud (cloud);
    sor.setLeafSize (0.001f, 0.001f, 0.001f);
    sor.filter (*filtered_cloud);
    return;
}

void NormalEstimate(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud,
                    pcl::PointCloud<pcl::PointNormal> &CloudNormals)
{
    pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> ne;
    ne.setInputCloud (cloud);
    pcl::search::KdTree<pcl::PointXYZ>::Ptr
            tree (new pcl::search::KdTree<pcl::PointXYZ> ());
    ne.setSearchMethod (tree);
    pcl::PointCloud<pcl::Normal>::Ptr cloud_normals (new pcl::PointCloud<pcl::Normal>);
    ne.setRadiusSearch (0.006);
    ne.compute (*cloud_normals);

    pcl::concatenateFields(*cloud, *cloud_normals, CloudNormals);

    std::vector<int> indices;
    pcl::removeNaNFromPointCloud(CloudNormals, CloudNormals, indices);
```

```
    return;
}

void ViewpointFeatureHistogram(pcl::PointCloud<pcl::PointNormal> mls_points,
                               pcl::PointCloud<pcl::VFHSignature308>::Ptr &vfhs)
{
    pcl::VFHEstimation<pcl::PointNormal, pcl::PointNormal,
            pcl::VFHSignature308> vfh;
    vfh.setInputCloud (mls_points.makeShared());
    vfh.setInputNormals (mls_points.makeShared());
    pcl::search::KdTree<pcl::PointNormal>::Ptr
            tree (new pcl::search::KdTree<pcl::PointNormal>);
    vfh.setSearchMethod (tree);
    vfh.setNormalizeBins(true);
    vfh.compute (*vfhs);
    return;
}

void visualizeClouds(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud,
                     pcl::PointCloud<pcl::PointNormal> mls_points,
                     int object_class,
                     bool known_object)
{
    std::ostringstream output_string;

    if(known_object)
        output_string << "Object classification: Object " << object_class;
    else
        output_string << "Object classification: Object " << object_class << "?";

    pcl::visualization::PCLVisualizer viewer("Point Cloud Viewer");
    int v1(0);
    viewer.createViewPort(0.0, 0.0, 0.5, 1.0, v1);
    viewer.addPointCloud<pcl::PointXYZ>(cloud, "cloud1", v1);
    viewer.addText("Raw point cloud.", 10, 30, 14, 1, 1, 1, "title v1", v1);
    viewer.addText(output_string.str(), 10, 10, 14, 1, 1, 1, "classification", v1);

    int v2(0);
    viewer.createViewPort(0.5, 0.0, 1.0, 1.0, v2);
    viewer.addPointCloud<pcl::PointNormal>(mls_points.makeShared(), "filtered cloud",
        v2);
    viewer.addText("Filtered point cloud.", 10, 30, 14, 1, 1, 1, "title v2", v2);
    viewer.resetCamera();
    viewer.spin();
    return;
}

int main(int argc, char *argv[])
{
    /* Declare and initialise variables ************************************/
    bool known_object;
    int object_class = 1;
    boost::chrono::system_clock::time_point start;
    boost::chrono::system_clock::time_point stop;
    boost::chrono::duration<double> OCR_duration;// = stop - start;

    pcl::PointCloud<pcl::PointXYZ>::Ptr
            cloud(new pcl::PointCloud<pcl::PointXYZ>);
    pcl::PointCloud<pcl::PointXYZ>::Ptr
            filtered_cloud(new pcl::PointCloud<pcl::PointXYZ>);
    pcl::PointCloud<pcl::PointNormal> mls_points;
    pcl::PointCloud<pcl::VFHSignature308>::Ptr
            vfhs(new pcl::PointCloud<pcl::VFHSignature308>);
    pcl::io::loadPCDFile (argv[1], *cloud);

    FANN::neural_net ANN;
    ANN.create_from_file("/home/ivan/MEng/C++/Shared Files/ANN.net");
    fann_type input[INPUTS], *result;
    /*********************************************************************/

    /* Point Cloud Processing ***********************************************/
    start = boost::chrono::system_clock::now();
    std::cout << "Reducing input data (voxel grid filter) ...\t" << std::flush;
```

```
    VoxelGrid(cloud, filtered_cloud);
    std::cout << "Complete." << std::endl;
    std::cout << "Estimating surface normals ... \t\t\t" << std::flush;
    NormalEstimate(filtered_cloud, mls_points);
    std::cout << "Complete." << std::endl;
    std::cout << "Calculating Feature Histogram ... \t\t" << std::flush;
    ViewpointFeatureHistogram(mls_points, vfhs);

    // Get first 135 entries of viewpoint feature histogram
    for(int i = 0; i < INPUTS; i++)
    {
        input[i] = vfhs->points[0].histogram[i];
    }
    std::cout << "Complete." << std::endl;
    /************************************************************************/

    /* Classification/Recognition *******************************************/
    result = ANN.run(input);
    int max_index = 0;
    float max_val = result[0];
    for(int i = 1; i < CLASSES; i++)
    {
        float val = result[i];
        if(val > max_val)
        {
            max_index = i;
            max_val = val;
        }
    }
    stop = boost::chrono::system_clock::now();
    OCR_duration = stop - start;
    object_class += max_index;
    /************************************************************************/

    /* Print results to console *********************************************/
    if(max_val > 0) // threshold operation
        known_object = true;
    else
        known_object = false;

    std::cout << "\n***Results***" << std::endl;
    for(int i = 0; i < CLASSES; i++)
        std::cout << "Object " << i + 1 << ": " << result[i]
                  << std::endl;
    std::cout << std::endl;
    if(known_object)
        std::cout << "The object is: Object " << object_class << std::endl;
    else
        std::cout << "The object is: Object " << object_class << "?" <<
                     std::endl;
    std::cout << std::endl;
    std::cout << "Object classification took: "
              << double(OCR_duration.count()) << " seconds\n" << std::endl;
    /************************************************************************/

    /* Start visualisation thread *******************************************/
    boost::thread visualisation_thread(visualizeClouds,
                                       cloud, mls_points,
                                       object_class,
                                       known_object);
    /************************************************************************/
    visualisation_thread.join(); // join thread
    return 0;
}
```