Design and Development of an Intelligent Neuro-Fuzzy System for Automated Visual Inspection

by

Jonathan Killing

A thesis submitted to the Department of Mechanical and Materials Engineering

in conformity with the requirements for the degree of

Master of Science (Engineering)

Queen's University

Kingston, Ontario, Canada

July 2007

Abstract

Automated visual inspection (AVI) is an attractive way for companies to reduce the cost of part inspection. By using a camera system to inspect parts, there is a reduction in the required labour and a potential for higher production rates. This is especially important in the automotive industry where product life is usually short and part suppliers are increasingly being expected to deliver perfect parts to their customers. However, the set-up of an AVI system can be difficult, especially for companies with little or no previous vision system experience. As a result, training times are long and the potential benefits are often not fully realized.

The work of this thesis focuses on the design and development of an image-processing algorithm that can help to reduce training time and compensate for variations in the physical environment such as the surface illumination. The developed software uses ‘intelligent’ algorithms (specifically neural networks and fuzzy logic) to allow the system to learn the inspection process from examples of good and bad parts. Through the example images, the user is not only providing the information to produce a correct classification, but also, indirectly, the expected variation in the images. This means that, unlike traditional systems, the intelligent algorithm does not need to be explicitly told the allowable magnitude of variation.

The algorithm was tested on both laboratory-generated images and a database of files provided by an industrial client. The results suggest that example based training significantly improves training time. For images where there is a clear difference between pass and fail, the classification performance was found to be comparable to traditional threshold based algorithms. When more subtle differences are present, the neuro-fuzzy system has a slight performance advantage. A user interface was developed to enable implementation and future testing in an industrial environment.

Acknowledgements

I would like to thank Dr. Brian Surgenor and Dr. Chris Mechefske for initiating such an interesting project and one that expands the boundaries of Mechanical Engineering. I also want to thank Dr. Surgenor especially, both for his enthusiasm for the project, and also his patience with the number of other activities I was involved with during my time at Queen's.

Kornel Farkas, Sam Li, and Jeff Liaw at Van Rob were instrumental in developing this project, providing an industry perspective, and giving feedback along the way.

I also want to recognize the financial support provided by the Mechanical and Materials Engineering Department, Queen's University, NSERC, and AUTO21 which has been crucial to my ability to continue my education.

Finally, many, many thanks to my family – my parents (Margaret and Steven) for their love, support, and advice, and my brother (David) for being a great sounding board for ideas during this project.

Table of Contents

List of Figures

List of Tables

Nomenclature

Glossary

antecedent	The input side of a fuzzy rule
backpropagation	A method for training neural networks
blob	A region of pixels grouped together by similar properties
consequent	The output side of a fuzzy rule
crisp set	A classification set that requires points to be fully associated with one and only one subset.
crisp values	Scalar numerical values
degree of association	The degree to which a crisp value belongs to a linguistic variable as determined by the membership function
false negative	A part which is failed but should have been passed
false positive	A part which is passed but should have been failed
fan-out	Layer of a neural network which passes values to the next set of nodes without modification
feature vector	A vector of scalar values which can be used to represent some aspect of an image or object
firing strength	The degree to which a fuzzy rule is satisfied
fixture	A reference feature or set of features that can be used to locate an object in an image
fuzzy set	A set of membership functions
golden image	An image that is used as a standard to which all subsequent images are compared
linguistic variables	Variables that represent concepts like 'hot' rather than numerical values
membership functions	Functions that relate crisp values to linguistic variables
neuron	One node of a neural network that computes its output based on the state of the nodes to which it is connected.
QVision	The name of the software developed during this project
singleton	A single scalar value used as the output from fuzzy rules

Acronyms

ANFIS	adaptive neuro-fuzzy inference system
AVI	automated visual inspection
DIC	discrete incremental clustering
FIS	fuzzy inference system
GUI	graphical user interface
HSV	hue saturation value
IAL	intelligent automation laboratory
MF	membership function
NF	neuro-fuzzy
RMS	root mean square
SIFT	scale invariant feature transform

Chapter 1

Introduction

Manufacturers are experiencing ever greater pressures to meet a multitude of business demands. In this complex manufacturing environment, the basic requirement of inspection and quality control is often overlooked as an area for research and improved productivity. With automotive companies introducing "zero defect" policies, inspection costs have risen dramatically (Luko, 2000). Improved automated inspection systems have the potential to counter these rising costs.

The objective of this research is to improve the parts inspection component of the manufacturing process through the application of intelligent (neuro-fuzzy) systems. The development of this system is being undertaken concurrently at five universities across Canada. Researchers at each university are looking at industrial inspection problems from partners in the automotive sector. These problems represent inspection tasks which are either not suited to current vision inspection systems, or are suffering poor performance with the existing algorithms. A summary of the project background is given in Norman et al. (2006).

A flexible inspection system such as is outlined in this thesis should be faster to train, more adaptable to changing conditions, and have the potential to expand automated visual inspection (AVI) systems as a practical and economical way for industry to perform inspection tasks.

1.1 Problem Overview

AVI systems have been used for many years in industrial applications. The flexibility of these systems is attractive as it means the same hardware can be used for several applications. There is also a level of comfort for operators to work with a system that acts on the same data humans use – visible light. However, there is also a tendency to forget that the human vision system is quite complex. Inspection tasks that humans do easily can be very difficult for a computer system to perform.

In an industrial environment, the problem is even worse due to conditions in the assembly area. Typically, the process being inspected is not shielded from the rest of the plant allowing light, smoke, dust, and other contaminates to enter the inspection area. Although these changes in the environment are automatically compensated for by the human visual system, they can render a finely tuned AVI system useless.

One problem that makes vision systems so vulnerable is the reliance on fixed thresholds to analyze the image. Fixed thresholds are easily disrupted by changes in the environment since the output changes suddenly whenever the input crosses the threshold.

A good real-world example of these types of problems occurred at Van Rob Stampings. This company produces stamped metal parts for the automotive industry. As part of the process to produce a particular dashboard mount, 46 metal clips are inserted into the beam in order to provide replaceable threads for the fasteners (see Figure 1.1). The clips are inserted both manually and by automated equipment during various stages of the process. At the final inspection station, the beams are checked to ensure that all of the clips are present and correctly inserted into the beam.

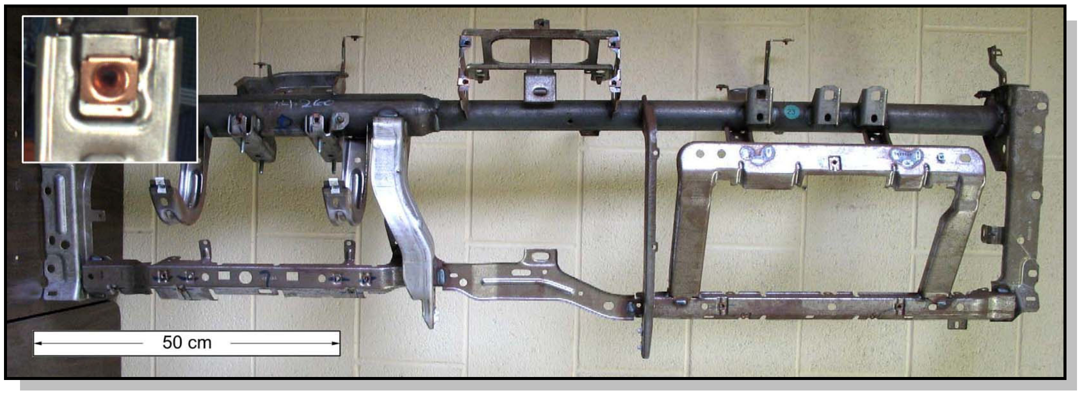


Figure 1.1: The cross-car beam provided by Van Rob Stampings for testing of the algorithm during development. A close-up of one of the clips is shown inset.

In the past, the inspection was done manually using two or three human inspectors to check each clip. Although this process works, it is slow and expensive. In addition, human fatigue can reduce the reliability of the inspection over time.

In an attempt to improve the process, an AVI system was installed that used a robotic arm to move the camera to various locations on the beam and inspect the clips. The area where the inspection was performed was open to the environment of the plant and also contained three other robots responsible for inserting clips into the beam. This setup created a number of problems for training:

- The light level varied depending on the time of day, other processes in the plant, and the flicker from the low frequency overhead lights.
- The background of the images contained motion from the other robots, workers loading and unloading parts, and machinery in other regions of the plant.
- The clip appearance changed drastically depending on the angle of the clip to the camera.

Through trial-and-error changes to the structure of the inspection area, the lighting, and the camera position, many of these problems were reduced. However, the upgrades took about a year to complete and the final system was still not as reliable as desired, still requiring a human inspector to double-check the output of the machine vision system (see Figure 1.2).

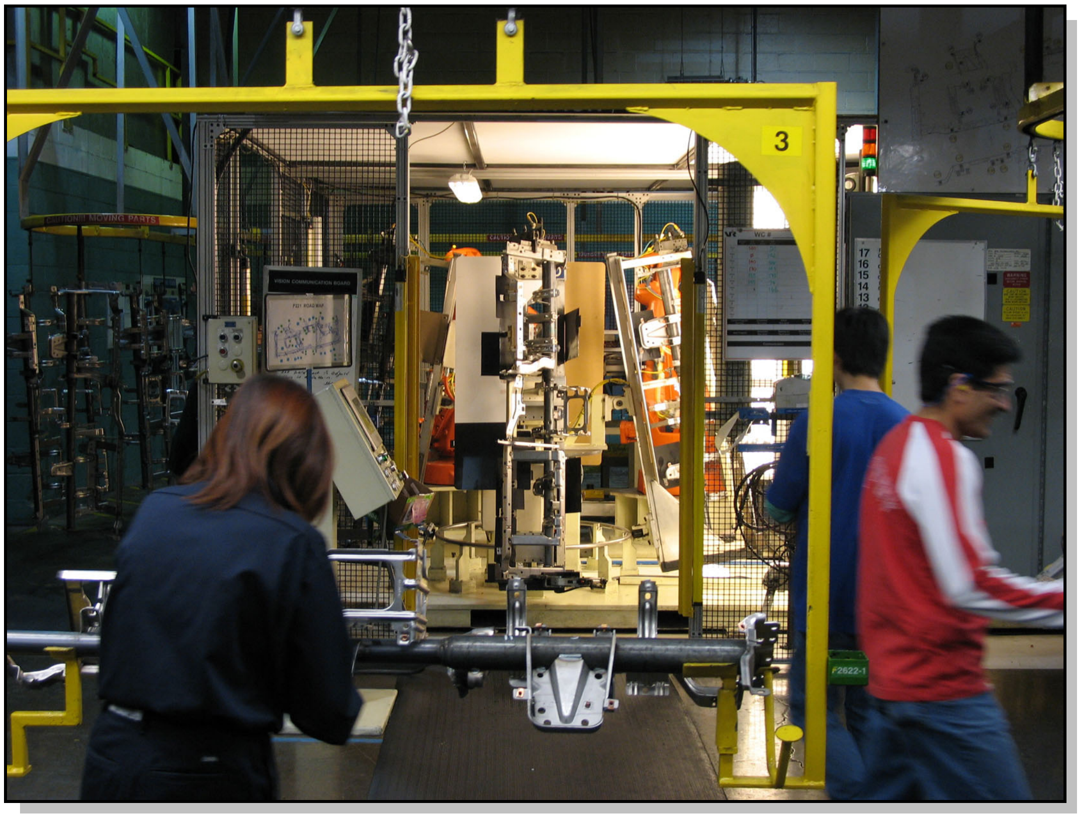


Figure 1.2: Human operators (foreground) double checking the output from the machine vision system (background) because of poor machine vision system performance.

Through discussions with several other companies it was found that similar problems are encountered by small to medium sized businesses that are attempting to implement machine vision into existing production processes.

1.2 Thesis Objectives

The purpose of this project is to try to improve the performance of existing AVI systems. Specifically, the goal is to use a neuro-fuzzy approach to make the system more tolerant to changes in the process environment.

In order to develop a working algorithm more quickly, five different universities are participating in the project, working on the same set of data. Since there are many possible approaches, each school is attempting to specialize on one approach only. The research at Queen's University focuses on using the Adaptive Neural Fuzzy Inference System (ANFIS) implementation in Matlab®. This is a well supported platform and has shown positive results for other projects undertaken by the Queen's Intelligent Automation Laboratory (IAL) (Samhouri, 2005).

Through the whole development process, the focus is on the final use. The system can not be a solution to one specific problem, but needs to be a general algorithm that can work with minimal modifications for a variety of applications. In addition, it has to be intuitive and familiar in order to gain acceptance into the industrial environment.

1.3 Organization of Thesis

This thesis is organized as follows. Chapter 2 gives a review of the literature on topics relevant to the research undertaken, both as background and to indicate the current state of development. Chapter 3 discusses the development of the algorithm and the reasons behind the inclusion of the various components. At the end of Chapter 3, the structure of the final algorithm is given. Chapter 4 covers the user interface developed as the user-friendly link to the algorithm functions. Chapter 5 outlines the test apparatus that was constructed in the IAL at Queen's University, and the results of using this apparatus to test the algorithm performance on both lab-generated images, and images supplied by Van Rob. Finally, Chapter 6 summarizes the major findings of this thesis and presents possibilities for future work.

Chapter 2

Literature Review

2.1 Quality Inspection

In general, quality control in the manufacturing process aims to reduce the variability in the product in order to meet functional and/or aesthetic requirements. Quality inspection is one method to ensure that the product quality is maintained. Depending on the production rates and the complexity of inspection, some or all parts may be inspected. If only a sample is checked, then the results are analysed statistically to draw conclusions about the remainder. Full inspection is usually preferred as it does not require statistical extrapolation in order to determine the overall quality of parts. However, for full inspection to be feasible, the parts must either be produced at a low rate, have very simple inspection needs, or automated inspection methods must be used.

Low production rates are not common since they are not desirable for many reasons. Similarly, simple inspection is rarely the choice since the more detailed the inspection, the more defects can be found and classified in order to correct the process that creates them. Therefore, many companies are looking to automated inspection systems to increase productivity and reduce rework (Thomas, 2006).

Inspection requirements for most parts can be classified into three main categories: part location, dimensional analysis, and surface condition (Hunter, 1995). Part location involves verifying correct assembly of parts by ensuring that components are present and properly aligned. Dimensional analysis measures the size of various components and component features to ensure correct production of the part. Surface condition observations verify surface finish requirements and check for damage sustained during the manufacturing process.

To automate these processes, a wide variety of methods have been developed. Semi-automated approaches use equipment and/or software to assist a human inspector in making the correct measurements and recording the results (see Figure 2.1). These devices remove much of the thinking process from the human operator, but still require a labour force. This negates the potential cost savings possible with a fully automated system.

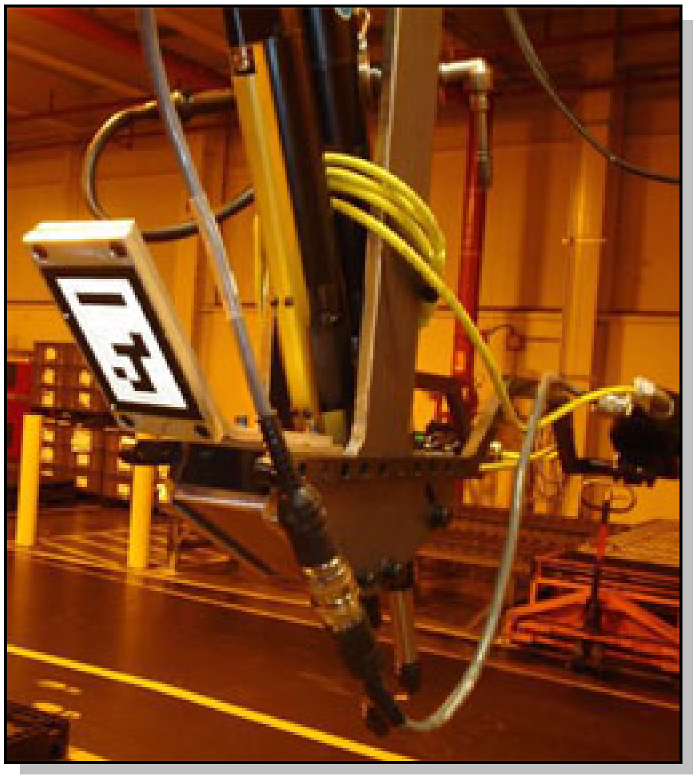


Figure 2.1: Semi-automated inspection systems assist a human operator by tracking tools to ensure they have completed all tasks at a particular assembly station (Wilson, 2006).

Coordinate measuring machines (CMMs) are another example of semi-automated inspection whereby a part is analyzed by touching a probe to key reference points on the component (see Figure 2.2). The inspection process is relatively slow and requires isolated facilities away from the dust and vibration of the assembly area. Therefore, it is typically only used for a small sample of parts or when very high precision is required.

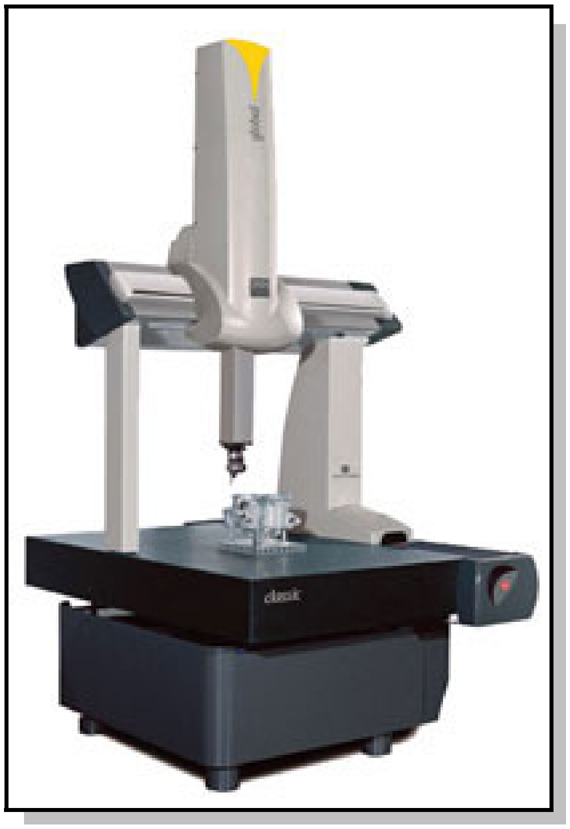


Figure 2.2: A coordinate measuring machine (Brown & Sharpe, 2007)

For high speed, and to reduce the quantity of custom tooling that is required for inspection, it is desirable to avoid mechanical contact with the parts. Automated Visual Inspection (AVI) is one such method that has been steadily gaining popularity as more flexible approaches are required. An AVI system uses a camera to capture a visual image of the part. Computer processing is then performed to extract relevant features of the object and determine if the part has been correctly manufactured. An example of an industrial AVI system is shown in Figure 2.3.

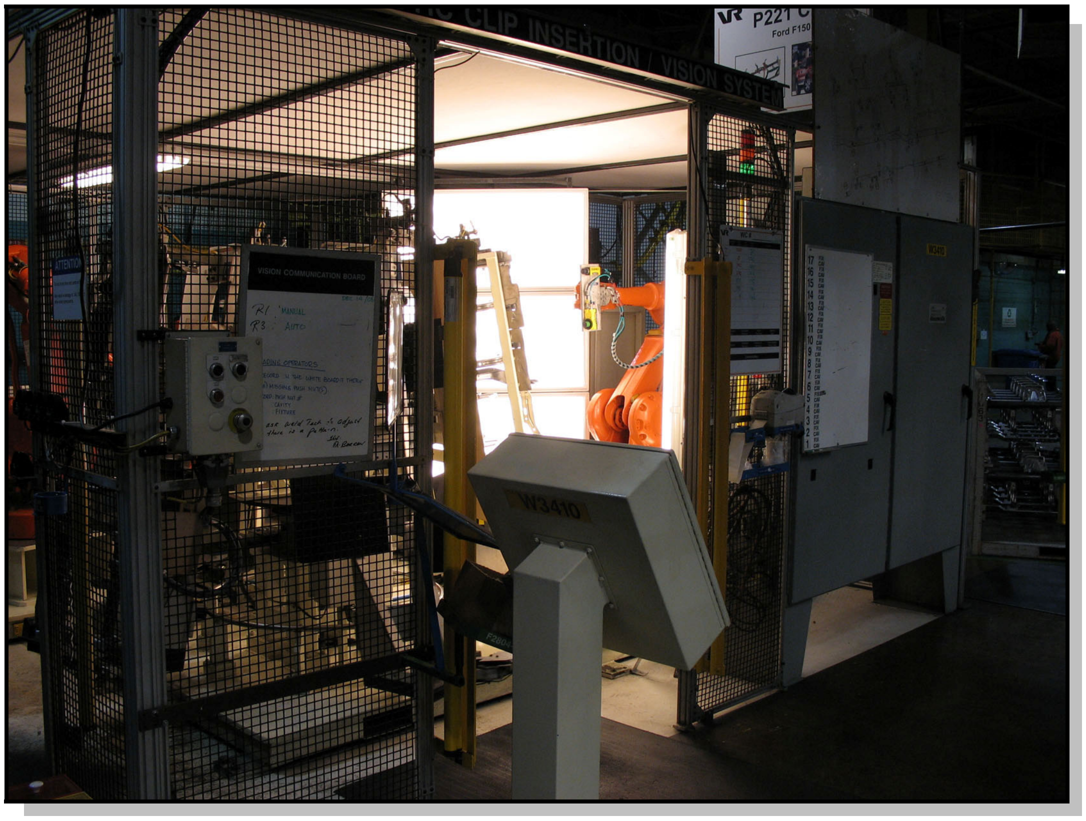


Figure 2.3: An example of an Automated Visual Inspection system installed at Van Rob Stampings. The system uses two cameras mounted on a robot (visible inside the cage) and displays the results to the operator via the console unit (foreground)

2.2 Automated Visual Inspection

Visual inspection has been the way that parts have been inspected by humans for a long time. The appearance of the part can show surface defects, incorrect assembly, and part damage. AVI systems attempt to use this same information about the appearance of the part to do the inspection without human intervention. If AVI systems were as capable as a human inspector, no physical changes to the inspection environment would be required.

However, the human visual system is quite complex and what seems a trivial task may require months of training for an automated system.

Because of the wide variety of operating conditions in manufacturing facilities, AVI systems vary greatly in design. However, almost all systems have three main components: a camera, an illumination system, and some form of processing device as shown in Figure 2.4. In many cases some or all of these functions are combined into a single unit.

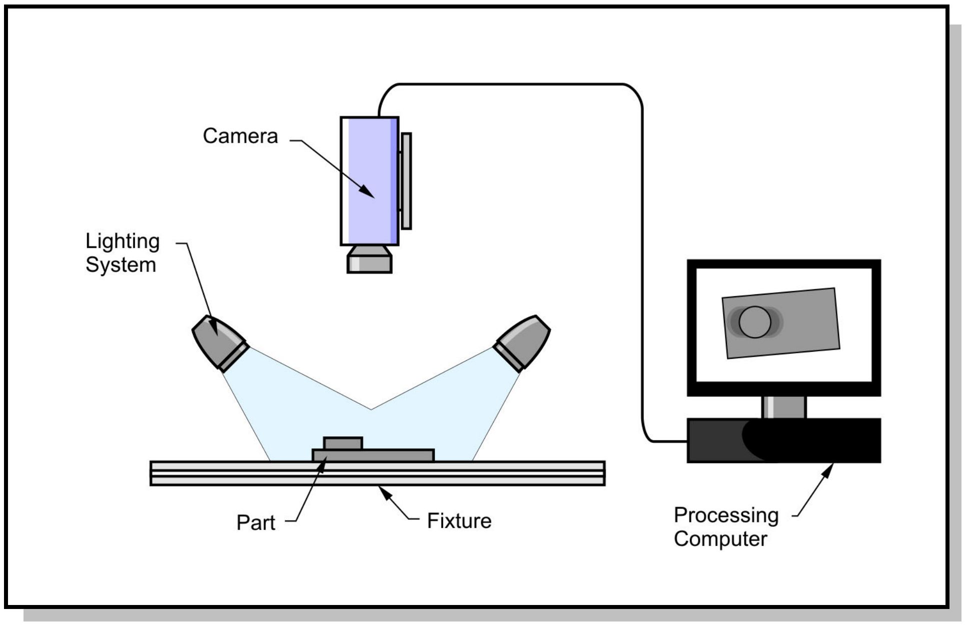


Figure 2.4: General components of an Automated Visual Inspection system.

The process by which the appearance of the part is captured and then processed to determine a final accept/reject decision can be broken into four main parts as shown in Figure 2.5 and described below.

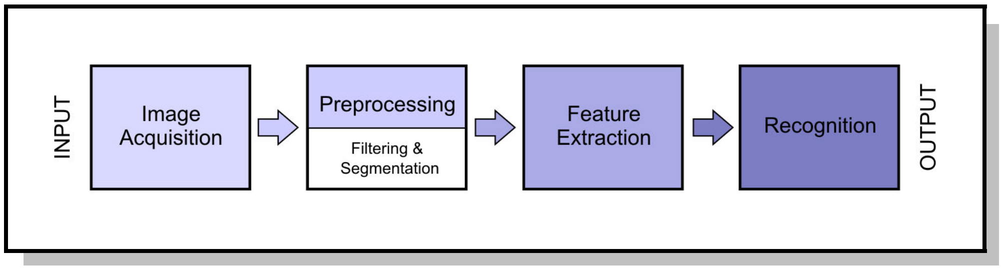


Figure 2.5: Steps in the AVI process

2.2.1 Image Acquisition

Acquiring the image consists not only of the optical capturing process occurring within the camera, but also of the lighting that makes the objects in the scene visible. One of the most difficult stages of the setup process is to find a camera and lighting configuration that gives the best view of the feature to be inspected. The lighting should highlight the feature in a way that accentuates the definition (Novini, 1985).

For example, when surface finish is being inspected, a low angle light will accentuate the surface texture. However, for inspecting the diameter of a hole on the same part, a strong backlight may give a better contrast to extract the circle shape. Although the ideal inspection conditions are rarely met due to constraints on the physical system, time spent at this stage can make the software design and implementation process much easier. Approaches to sensor planning are well summarized by Tarabanis et al. (1995).

In challenging environments, it may be necessary to actually add information to the scene. For example, attaching fiducial markers (targets of a known size, pattern, or location) to the parts can assist the vision system in determining the orientation and scale of the part (Fiala, 2004). In some cases, the markers are temporary, while others are applied as part of the manufacturing process. An added advantage of this system is that the marker can also contain identification numbers which allow the parts to be tracked through the assembly process.

Another method of adding information to the scene is to project structured light onto the part. If the light pattern is known, then the distortion of that pattern on the part can give dimensional information. This technique is particularly useful where information about the depth of the object is required. For example, Chung et al.(Chung, 2006) looked at using structured lighting to inspect cars on an automotive assembly line. To inspect for a vehicle door being open, a line of light was projected onto the body panels. If the line was continuous, the door was determined to be closed. If the line was not continuous, it was possible to tell that the door was ajar. This determination would be much more difficult if general illumination was used.

2.2.2 Pre-Processing

Pre-processing steps include transformations such as altering contrast, enhancing edges, or changing tone. The goal at this stage is to highlight the features of interest and diminish background noise.

In some cases, pre-processing can be used to compensate for deficiencies in the physical conditions. For example, lighting-related problems such as colour shift, shadows, and specular reflections are quite common. Finlayson (2001a, 2001b, and 2002) has published several papers on correcting some of these problems using illumination models. For most industrial cases, however, the correction of illumination simply consists of having an auto-gain or auto-iris control on the camera in order to keep the overall intensity constant. Since many algorithms rely on being able to segment the image into light and dark regions, accurate control over lighting intensity can be a crucial.

2.2.3 Feature Extraction

In order to simplify the image, it is typically broken into features. These are representations of a group of pixels with some common characteristic such as texture, colour, or intensity. More complex features group pixels which appear to represent edges, lines, or other shapes.

The parameters of each feature are usually specified in terms of scalar values which can be combined to form a *feature vector* for the entire image. The feature vector can be considered a compact representation of the contents of the image. Thus, a feature-based approach reduces the quantity of information that needs to be processed by subsequent stages. Feature extraction methods are discussed further in Section 2.5.

2.2.4 Recognition

In this final stage, the feature vector is processed to decide if the part is acceptable. This may be a fairly simple test such as the presence or absence of a single feature, or may involve very complex processing. Programming an AVI system to perform recognition is usually labour intensive and requires many trial-and-error steps. Improving the recognition stage of the AVI system is the main goal of this thesis work.

2.3 Camera Configuration

In order to obtain a clear image from the camera with the most useful information possible, the camera must be configured both physically and in software.

2.3.1 Lens Selection

Although most camera specifications focus on the number of pixels on the sensor, the lens used determines what portion of the scene each of the pixels represent. Because most machine vision lenses do not have a zoom control, the field-of-view of the camera is fixed. One of the benefits of this system is that objects in the image will remain the same size for a given distance from the camera. This allows real-world distances to be computed from the number of pixels an object occupies in the image.

Because the angle of view is fixed, it must be selected to give the desired field of view at the operating distance to the part. In situations where the camera position is also a variable, there are then a number of focal length/camera distance combinations that will produce the same field of view (see Figure 2.6).

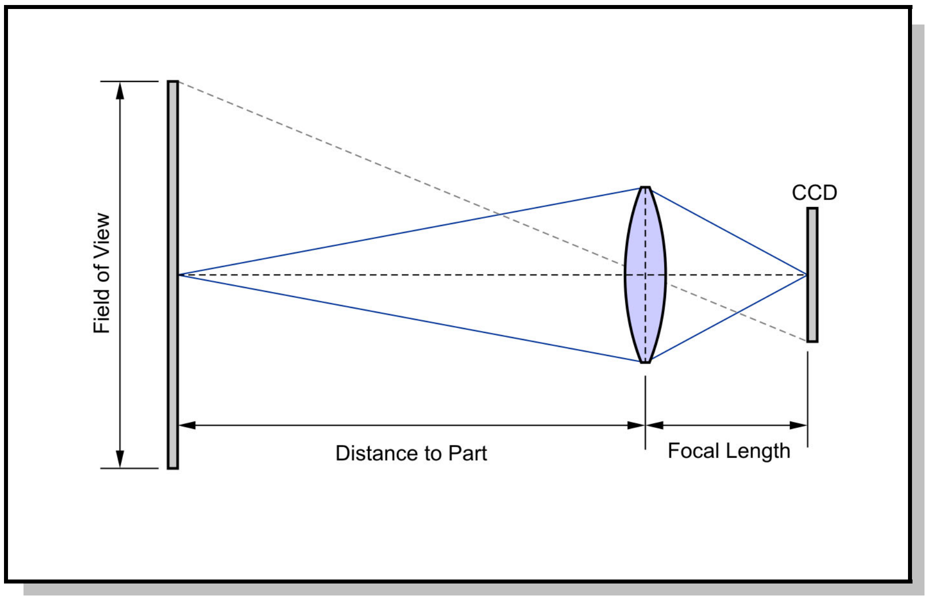


Figure 2.6: The geometry of a simple camera lens system

The difference is that if the camera is further away from the part, the effect of perspective will be decreased. That is parallel lines on the part will appear more parallel. A camera that is very close to the part with a wide angle lens will tend to distort the image.

On the other hand, having the camera far away from the part means there is more chance for vibration in the camera mount to create blur, and for particles in the air to reduce the image quality. Therefore, a good balance between distance and distortion must be found for the particular application.

2.3.2 Camera Setup

Once the camera has been mounted in place and the lens fitted, the image can be displayed on-screen and adjusted to give the clearest image. First, focus must be adjusted to make the part crisp. The overall intensity of the image can then be set using the gain control of the camera and/or the iris control. A higher gain or longer exposure tends to create more noise in the image. To use lower gains or shorter exposures, the iris of the camera (also called the aperture), must be opened wider to allow more light into the camera. However, opening the iris also influences the depth-of-field of the camera which is the range of distances that are in focus. A larger aperture creates a narrower depth-of-field and thus only objects a specific distance from the camera will be in focus (an example is shown in Figure 2.7). A good balance should be achieved between the depth-of-focus, and the gain of the camera such that noise from the sensor electronics does not start to interfere with the image and the entire part is in focus.

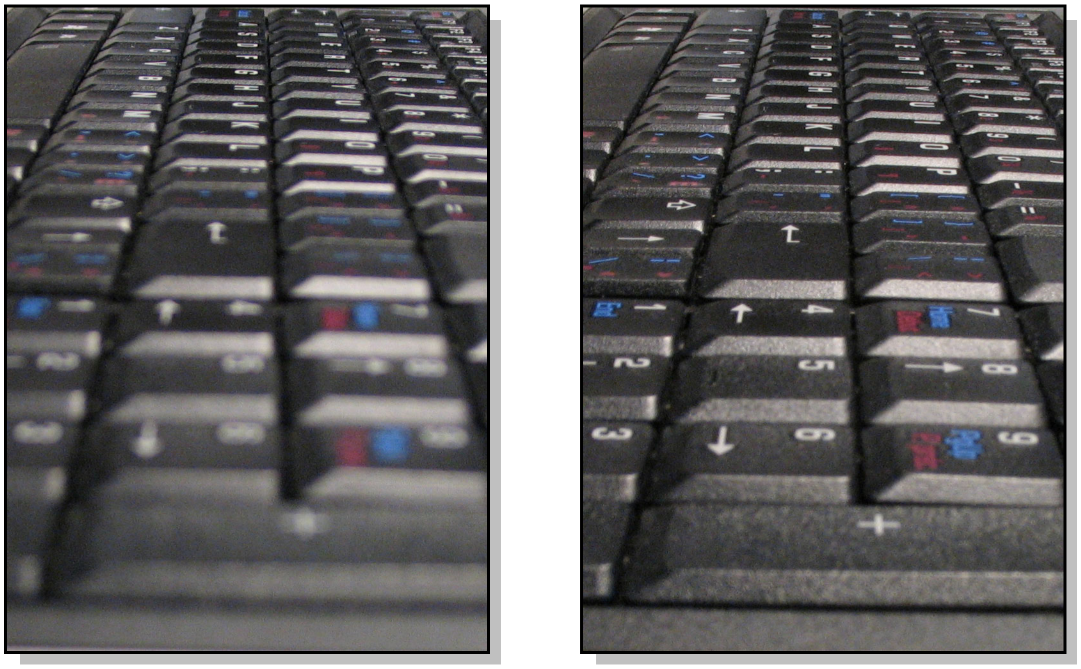


Figure 2.7: These two images were taken with the same focus setting but different iris sizes. The image on the left has a narrow depth-of-field due to a wide open iris, resulting in blurred foreground and background objects.

In balancing the gain, shutter speed, and aperture size, the final goal should be to arrive at an overall intensity that captures both the bright and dark regions of the part. If possible, the brightest highlights should not exceed the maximum pixel value as this means that information is being lost. Additionally, no pixel in the dark regions of the part should have a value of zero as this could be interpreted as any intensity below the threshold of the camera. Meeting these requirements in practice is often difficult (especially for metallic parts), because the intensity of highlights can often be hundreds of times the intensity of the surrounding surface. The normal range of digital camera pixels is only 0-255, so some clipping often occurs as shown in Figure 2.8.

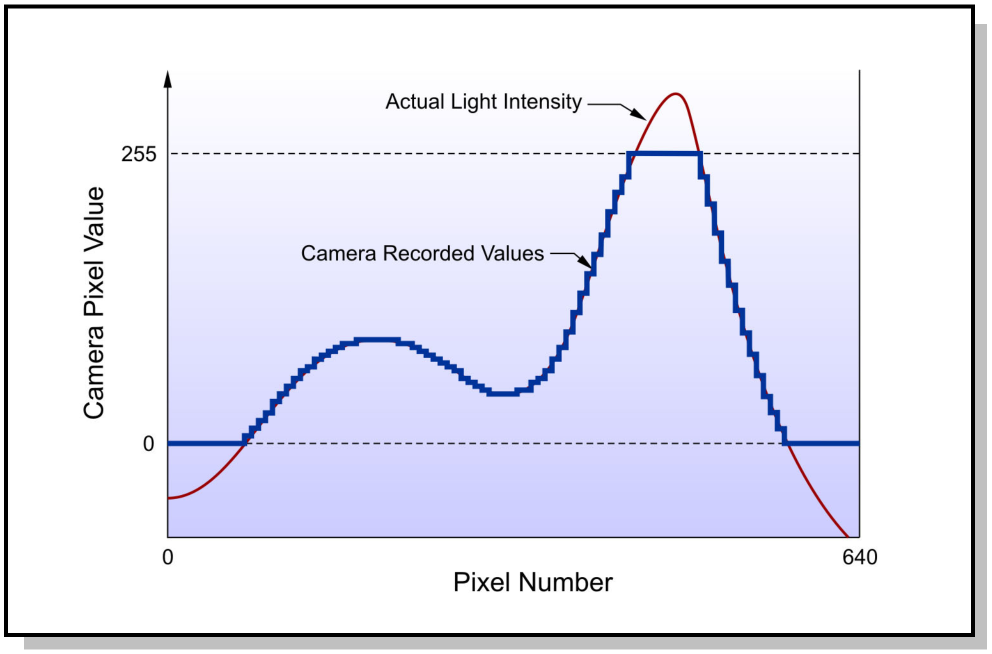


Figure 2.8: Clipping can occur if the range of intensities in the scene falls outside the available range of the camera sensor.

For a colour camera, there are some additional settings. One of the most important is the white balance which determines the correction that is applied to the red, green, and blue pixel values in order to make the colours appear natural in the final image. White balance is done by focusing the camera on a white surface under the expected lighting. The correction values are then adjusted in order to achieve an average grey image (all colour intensities equal).

2.4 Image Processing Approaches

Since almost all computer images are captured as an array of pixel values, all computations on images must start from the pixel values. However, there are many ways to interpret the image.

2.4.1 Pixel-Based

The simplest processing method is to work on the pixel values directly. Image properties such as average intensity, ratio of dark to light pixels, intensity deviation, colour histogram and similar overall measures are quick to compute and in some cases can deliver all the required information.

However, the intensity and colour of a single pixel often has very little to do with the contents of the image. A simple example would be the case where the part has shifted position relative to the camera so that the part is no longer represented by the same set of pixels.

2.4.2 Pattern Matching

For situations where the part appearance is known and is distinctly different than the surroundings, pattern matching can be used. This technique works by comparing a model of the object against the source image. By trying all possible locations of the part at all possible rotations and/or scales, the most probable location of the part can be found. The main issue with a pattern matching approach is that as the number of allowed degrees-offreedom of the part increases, the problem becomes very computationally expensive (Davies, 2005).

A more flexible version of pattern matching is to match points on the model against points in the image. A mapping can then be determined based on the match locations. The Scale Invariant Feature Transform (SIFT) method developed by Lowe (2004) is one way to match points between the model and the image. The method develops key-points based on peaks in the gradient image at various resolutions. A vector of local descriptors (such as gradient directions, and magnitudes) for the key-points in the model is created. These descriptors can then be compared to the descriptor vectors for the key-points in the image to determine likely matches.

2.4.3 Feature-Based

For inspection tasks where specific dimensions and relationships between objects are important, it is usually necessary to extract high level features from the image such as lines, circles, and colour patches. This approach attempts to mimic the way that human visual processing is thought to break down scenes into basic shapes before they are processed for recognition. As such, it is very intuitive to see an image represented by outlines, colours, and basic shapes.

Using high level features also reduces the amount of data that must be handled by the recognition system since each feature represents the characteristics of a region of pixels.

For example, a line feature summarizes all the pixels in the region of that line segment with only three values: the perpendicular distance to the origin, the angle of the line, and how strongly the pixels support the existence of that particular line.

2.5 Feature Extraction Methods

Technically, feature extraction is any process which acts on an image and returns some description of the image. This may be the position of lines, pixels which satisfy a certain criteria, or more statistical values such as the mean and standard deviation of each row or column. Which feature description is used depends highly on the application. The goal is to produce the most meaningful data that will clearly highlight the object of interest.

This section focuses on feature extraction methods for high level features such as lines, circles, and colours. These features are not always the best descriptors for analysis of an image, but have the advantage of being very intuitive for the user to interpret.

2.5.1 Edge Finding Methods

Finding shapes in the image usually starts with locating edges. In image processing, an edge is defined as a location where the intensity of the pixels changes suddenly. Reflected light intensity is strongly related to the angle of the reflecting surface to the camera. So, locations of sudden pixel intensity change usually correspond well with the physical edges of the part, where faces with different angles are meeting. Of course, there are other conditions that can trigger false detection of an edge such as surface printing (text, graphics, etc.), shadows, and highlights.

One of the simplest ways to detect an edge is to produce a gradient map. This is an image in which the value of each pixel represents the magnitude of gradient in the original image. The gradient magnitude is calculated by:

$$G_x = \frac{\partial f}{\partial x},\ G_y = \frac{\partial f}{\partial y},\ |G| = \sqrt{{G_x}^2 + {G_y}^2}$$

where f is the intensity of the image at location x, y (Gonzalez et al., 2004). The direction of the edge can also be determine by the relative strength of the x and y components.

Candidate edge points are highlighted by looking for the strongest gradient values in the image. In some cases, a threshold is used to count all gradient values above a certain value as being edges. In other methods, a search is performed starting from the peaks in the gradient and tracing out continuous edge paths. The Canny edge detection method is one such method that is very popular (Gonzalez et al., 2004). An example of the output of the Canny edge detector on an image is shown in Figure 2.9.

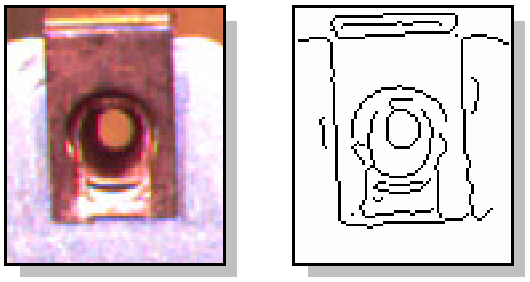


Figure 2.9: Output of the Canny edge detector (right) on the image of a J-clip (left).

2.5.2 Hough Transform for Lines

The edge finding methods only produce a list of pixels that are potential edges. However, for many applications, it is more useful to know if these edge points form a particular shape. The Hough Transform is one way to perform this search (Davies, 2005).

The Hough Transform uses a mapping of the *x*, *y* image space to a virtual space which describes some parameter of the pattern of interest. For example, the most common Hough Transform is for the detection of lines in the image. Each pixel that was identified as an edge in the image is considered as a vote for all possible lines that go through that point. These votes are transferred into a Hough space with axes of slope and y-intercept. With these axes, each pixel in the image will create a line of votes in the Hough space (see Figure 2.10).

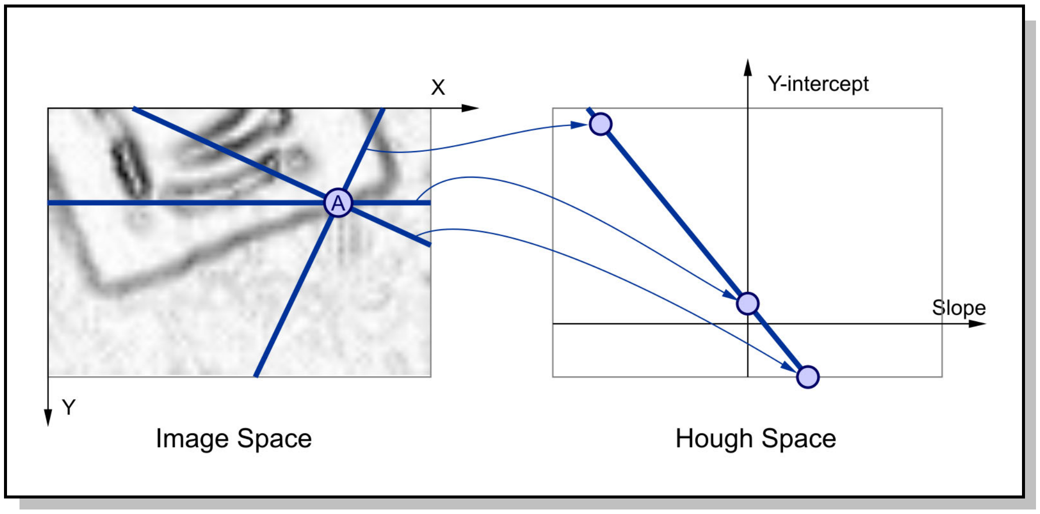


Figure 2.10: Mapping of the image space to Hough space. Each of the infinite family of lines that go through point A in the image space votes for a particular slope and y-intercept combination (three examples are shown). All the votes for one family produce a straight line when plotted in Hough Space.

In the simple case where the edge image contains only a single line of pixels, the effect of adding votes to the Hough space will be to create a peak where all the voting lines intersect. This point will represent the slope and y-intercept parameters of the line that was in the source image (see Figure 2.11).

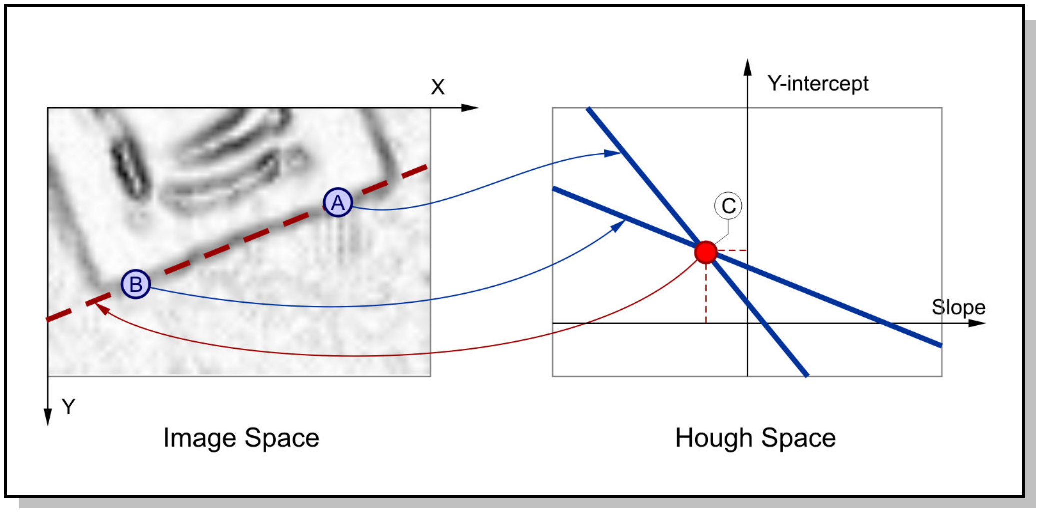


Figure 2.11: Repeating the mapping procedure outlined in Figure 2.10 to all of the edge points (two examples are shown) results in multiple voting lines in Hough Space. The coordinates of the region in Hough Space with the largest number of votes (C), gives the slope and y-intercept of the most probable line represented by the edge pixels.

The basic Hough transform has a duality of mapping points in the image to lines in Hough space, and points in Hough space to lines in the image. However, there is a problem when the slope of the line approaches infinity (vertical line) because the Hough space has to be of finite size. To get around this problem, the lines can be represented using the angle of the line and the perpendicular distance to the origin. Since the angle is bounded from -180 to 180 degrees, and the perpendicular distance cannot exceed the diagonal dimension of the source image, the Hough space has a finite size. The challenge with this representation is that edge points in the image now map to sinusoidal curves in the Hough Space making computation slightly more complex. However, the concept is still the same – the strongest intersection of the sinusoidal curves identifies the angle and perpendicular distance of the strongest line in the source image.

2.5.3 Hough Transform for Circles

Since the first basic Hough method was developed, it has been modified to suit detection of many different shapes. For instance, there is also a Hough Transform method for detection of circles. The problem with circles is that there are three parameters for each circle (radius, X position, and Y position). This means that either the search must be performed for each potential radius (as shown in Figure 2.12), or the Hough space must be made three dimensional. A three dimensional voting space is much harder to handle, both in memory requirements, and in determining the location of peak values.

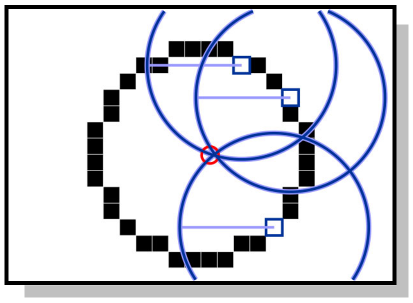


Figure 2.12: A basic Hough circle detection method. Each pixel in the edge image creates a circle of votes at a fixed radius from the point. The location with the highest number of votes is the most likely centre-point location.

Peng (2006) developed code that implements a more computationally efficient method of Hough circle detection. As mentioned earlier, computing the gradient provides both magnitude and direction information. By using the directional information, it is possible to separate the search for circles into independent searches for the centre of the circle and for the radius.

The possible centre-point locations are determined by extending lines perpendicular to the gradient direction at each edge point (see Figure 2.13). Considering each line as votes for a particular circle centre-point, a peak in votes will occur at the actual circle centre. Once the circle centre is identified, the radius of the circle can then be found by locating the radius which encompasses the highest number of edge pixels.

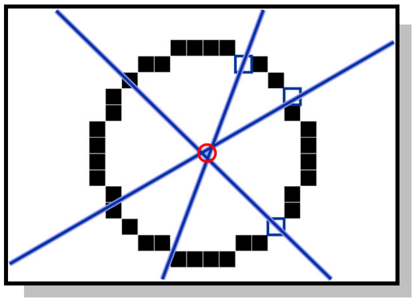


Figure 2.13: A more computationally efficient method of locating circle centres from the edge points. Each edge point votes for a line of centre-point locations. The direction of the gradient at each edge point is used to orient the line direction.

2.5.4 Grayscale Hough Transform

Normally, the Hough Transform is computed from binary image edge points. However, this means that all pixels vote equally in Hough space regardless of how strong the gradient is at that point. In order to give more proportional voting, the votes can be based on the magnitude of the gradient corresponding to each edge point. This method typically improves results but requires more computation.

2.5.5 Colour Processing

So far, all the methods that have been discussed are based on processing of greyscale images. However, in some applications the colour information is significant to the problem. Colour processing is challenging as the representation of colour in terms of pixel values often has little to do with the human visual interpretation of colour. This is especially true in looking at a range of colours. For example, to consider a range of red colours, it may be tempting to consider only those pixels with red component values that fall within specified limits. However, this does not consider the effects of the other two colour components.

One way to relate the colours to a more intuitive model is to transform from the standard red, green, blue (RGB) colour space to the hue, saturation, value (HSV) space (see Figure 2.14). This model makes it easier to isolate specific colours (hue), identify only "colourful" objects (saturation), or ignore the colour and look only at how bright the object is (value) (Gonzalez et al., 2004).

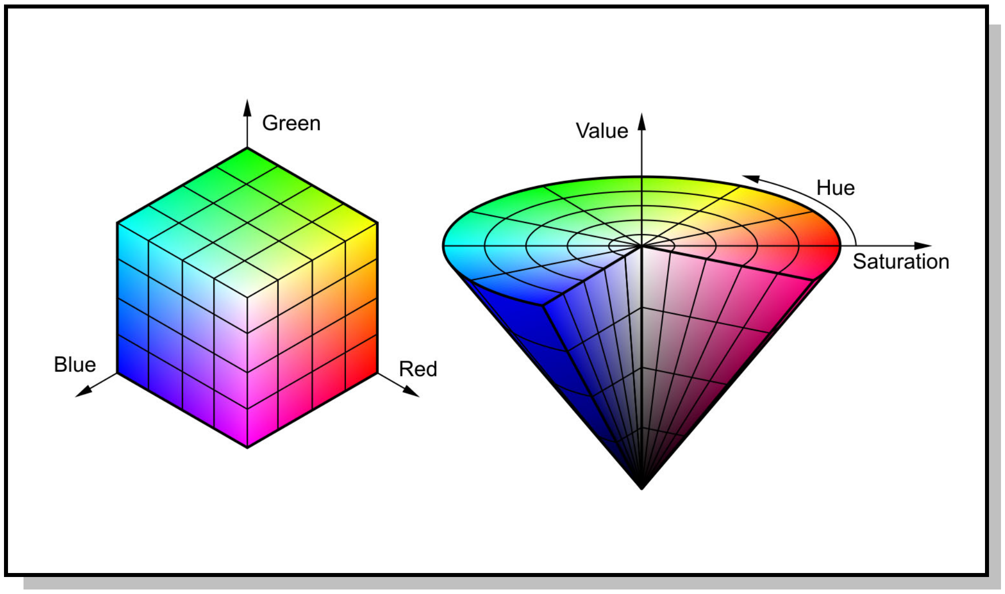


Figure 2.14: The RGB colour space (left) is native to the camera, but the HSV colour space (right) can help in processing colour information. (The Mathworks, 2004).

The main challenge with the HSV space is that the Hue is expressed as an angle between 0 and 360 degrees. This means that there is a discontinuity between the colour at 0 and the colour at 360. In most models, this occurs in the red region.

In order to actually extract colour information as a feature, the colours of each pixel in the region-of-interest must be summarized in some way. One way is to look at colour patches or blobs. Blobs have a specific meaning in image analysis and refer to contiguous regions of pixels that have similar properties. In this case, hue is the property of interest. By segmenting the hue space into equally spaced divisions, it is possible to group pixels by colour and then look for the largest groups of similarly coloured pixels. The overall properties of these regions (area, centroid, average colour) can then be used as features.

2.6 Threshold-Based Classification

In the case of quality inspection, the recognition stage typically consists of a simple classification of the part as either pass or fail.

If there are a total of *n* feature attributes extracted from the image and each attribute is considered to be a dimension of the input space, then the role of any classification method is to divide the input space into *n*-dimensional regions that denote the boundaries of pass and fail zones. Input values are then classified by which zone they fall into.

Traditional AVI systems use a fixed threshold method where the feature values are compared to limits set manually during the setup processes. Each dimension is considered critical so that if any dimension fails a threshold test, the entire part fails. This approach is common for dimensional checking where sizes must be within a specific range in order to fit with other assemblies.

Threshold cannot always produce the class division required. For example, if there are two input dimensions, then the input space is planar and the threshold will divide the input space into rectangular regions. However, if the true division follows an angled or curved boundary, threshold-based classification will not be able to correctly classify all of the points (see Figure 2.15). The same principle holds for higher dimension input spaces.

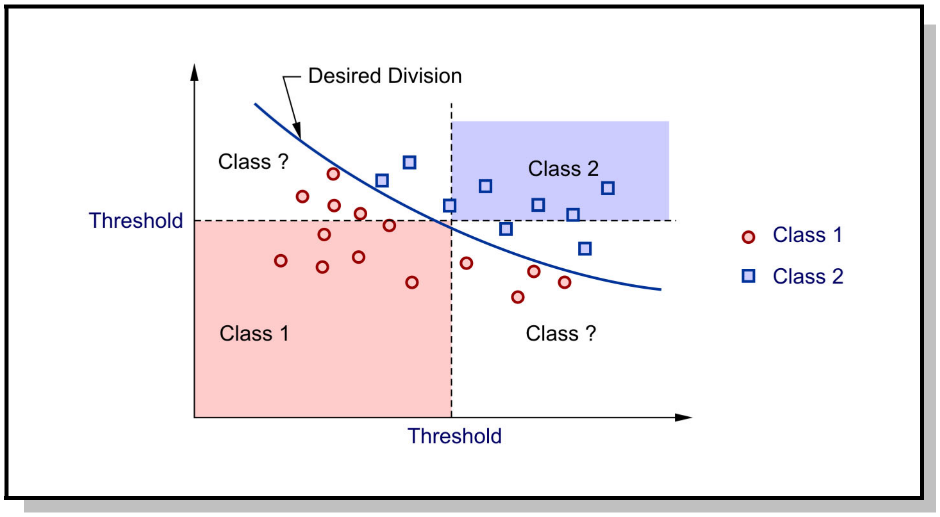


Figure 2.15: Dividing the input space using threshold values for each dimension results in perpendicular boundaries which may not capture the desired class division.

An extension to the basic threshold system is to consider each dimension independently. After the class of each input dimension is determined, they are combined to decide on an overall decision. This means that if one feature is poorly extracted or missing, it is still possible to make a judgment based on the results of the majority. The method by which the inputs are combined may be as simple as taking the mean or mode of the classifications for each input.

One of the main issues with a threshold-based system is its sensitivity to noise in the image. Since each input decision is made on the basis of a threshold value, a small change in the input value can cause the classification to transition suddenly from pass to fail. This may be a favourable attribute for dimensional inspection where each dimension is critical. However, for other types of problems such as surface and assembly inspection, it is preferable to have the classifier change its output gradually to allow for changes in the image due to environmental conditions, lighting, and part position.

"Intelligent" or "soft-computing" algorithms are one way to achieve more gradual changes in output. Two popular concepts in this area are Fuzzy Logic, and Neural Networks.

2.7 Fuzzy Logic Classifiers

Fuzzy logic evolved from the desire to train computer systems with human expertise. Converting the knowledge of the human expert to an equation that a computer can process is difficult when the process involves a number of variables and conditions. By encoding the human experience in a series of IF-THEN rules, fuzzy logic is able to produce a smooth output surface for all input combinations without an explicit model of the process involved (Nguyen and Walker, 2006).

Fuzzy inference systems have three main stages: input fuzzificaton, rule evaluation, and output defuzzification as shown in Figure 2.16 and described below.

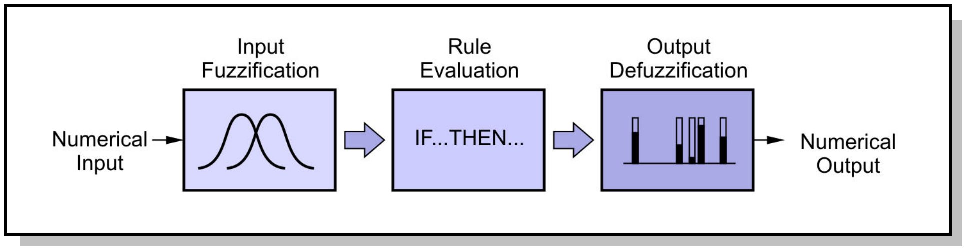


Figure 2.16: Steps in a fuzzy inference system.

2.7.1 Input Fuzzification

Input fuzzification involves conversion of the numerical input values (called *crisp values*) into *linguistic variables*. Linguistic variables are labels such as 'HOT' or 'COLD' which relate more to the way that a human would describe the value. Each input has its own set of *membership functions* which define how input values are mapped to the linguistic variables. Membership functions are typically trapezoidal, triangular or Gaussian in shape and are permitted to overlap. Each membership function is a subset of the overall *fuzzy set* for that input dimension.

As opposed to traditional crisp sets, fuzzy sets allow data points to be partial members of a subset and also members of more than one subset. The degree to which an input value belongs to a particular subset is referred to as the *degree of association* and takes a value anywhere from zero (no association) to one (complete association).

The concept of fuzzy sets can best be illustrated through an example. Consider a scenario where temperature is being classified. A crisp set approach would be to make two subsets: HOT and COLD. If the boundary between these subsets was at 20°C as shown in Figure 2.15, then 19.9°C would be considered COLD whereas 20.1°C would be considered HOT. This does not fit well with the human interpretation of hot and cold as there is some grey area around 20°C where the temperature would be 'a bit cold', or 'not very hot'.

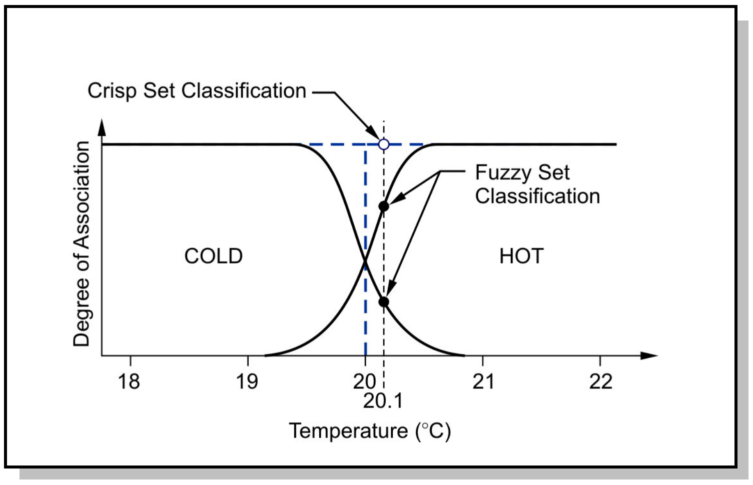


Figure 2.17: Example of the difference between crisp and fuzzy sets. The crisp set changes classification suddenly at 20°C, but the fuzzy set allows for a gradual change.

To capture these concepts, a fuzzy logic set would have overlapping membership functions with sloping sides so that temperatures around 20°C would have some degree of association with both the HOT and COLD subsets. For the case of membership functions shown in Figure 2.17, the temperature of 20.1°C would have a degree of association with the HOT subset of about 0.7 and with the COLD subset of 0.3.

2.7.2 Rule Evaluation

Once the inputs are fuzzified into linguistic variables, they can be used for evaluating rules without explicitly referring to the underlying numerical values. The basic structure of a rule is given as an IF, THEN statement as shown in Figure 2.18.

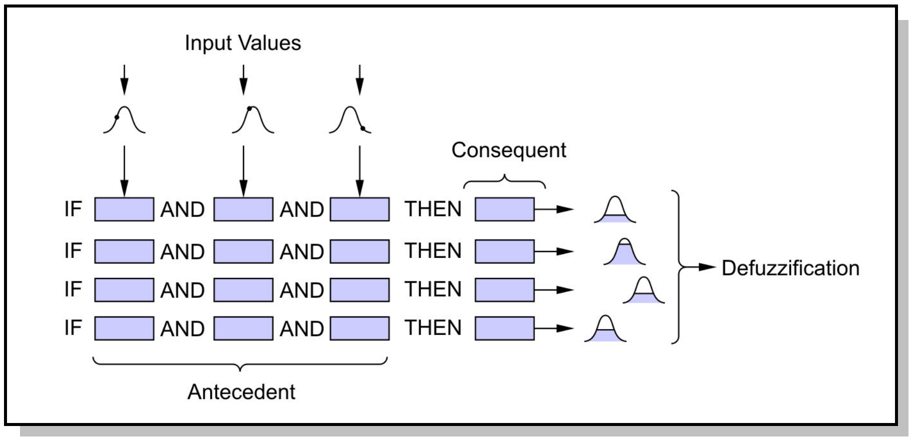


Figure 2.18: The structure of fuzzy rules and relationship to the rest of the fuzzy system. Inputs are fuzzified into linguistic variables which form the antecedent part of the rule. The antecedent parts are used to compute the output of each consequent part.

Continuing with the previous example of temperature, there may be a rule such as:

IF (T is COLD) AND (ΔT is NEGATIVE) THEN (Heater Power is HIGH)

The first portion of the rule is referred to as the *antecedent* part. Terms in the antecedent part refer to the input linguistic variables (which in turn are defined by the input membership functions) and are joined using a *union* operator such as AND or OR. It is the degree of association of each antecedent variable that is ANDed or ORed together to evaluate the rule. However, the standard AND/OR functions can only be evaluated when the input variables are binary (logical one or zero). Since the degree of association value is continuous, several AND/OR operator alternatives have been developed (see Table 2.1 for some examples).

Table 2.1: A sample of AND/OR operators that can be used with continuous variables ($\mu_A(x)$, $\mu_B(x)$). These are the four operators available in the MATLAB® environment (Kecman, 2001).

AND	**OR**
Minimum $\mathrm{MIN}(\mu_A(x), \mu_B(x))$	Maximum $\mathrm{MAX}(\mu_A(x), \mu_B(x))$
Algebraic Product $\mu_A(x)\mu_B(x)$	Algebraic Sum $\mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)$

The union of the antecedent degree of association values forms the *firing strength* for the rule. The firing strength can be considered a measure of how well the rule has been satisfied. If a rule is fired strongly, then the right side of the statement (called the *consequent*) should contribute strongly to the final output.

The consequent side of the rule is linked to output linguistic variables. These variables may be represented by membership functions (similar to the input side), or can be single scalar values (called *singletons*). The firing strength of the associated rule is used as a weight for that consequent output during the defuzzification process.

2.7.3 Output Defuzzification

The purpose of the defuzzification process is to combine the outputs from each rule into a single crisp output value for the whole network. If the outputs of the rules are membership functions, then the firing strength of the rule is used to calculate a truncated area underneath the membership function. The areas from each rule are then combined and the centroid is taken as the output value (see Figure 2.19). For faster processing, the centre of area calculation can be modified from a true centroid calculation to simpler approximations.

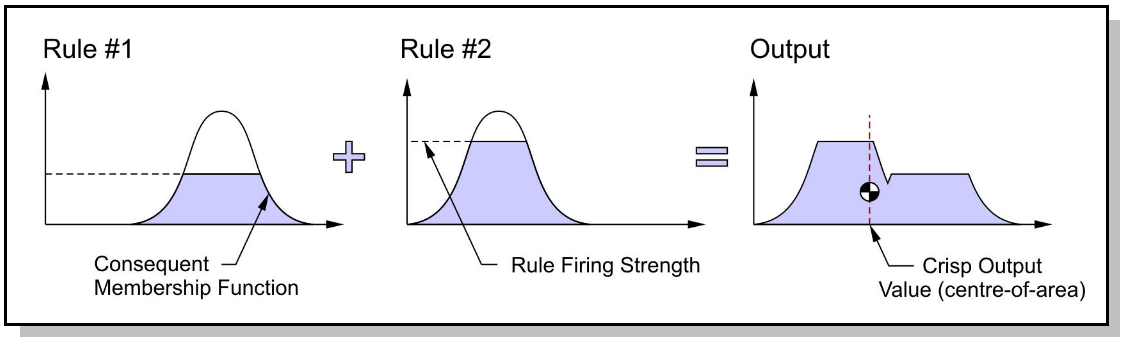

Rule #1
Rule #2
Output
Consequent Membership Function
Rule Firing Strength
Crisp Output Value (centre-of-area)

Figure 2.19: The process by which the rule consequent membership functions are combined to calculate a single crisp output for the network.

For the case of singleton consequents, the calculation is simply a weighted average of each singleton where the weights are the firing strengths of the rules (see Figure 2.20). This calculation is much less computationally expensive than the area centroid method so singleton outputs are preferred where speed is an issue.

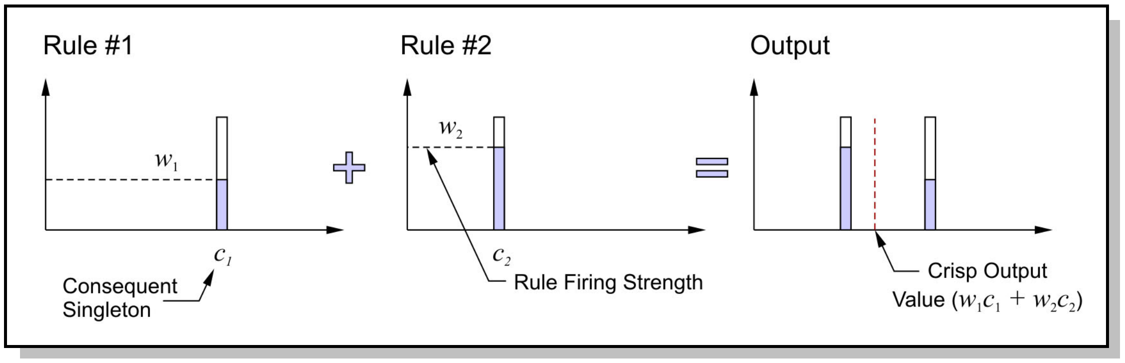


Figure 2.20: Defuzzificiation of singleton outputs becomes a simple weighted average of the singleton values.

By using either the centroid or weighted average calculations, fuzzy logic systems are able to produce a smooth continuous output, even for inputs that are different from the training set. This simulates the human ability to make reasonable interpolations between known information and to make judgements about previously unknown conditions.

2.8 Neural Network Classifiers

Artificial neural networks attempt to mimic the learning ability of the human brain using a series of interconnected *neurons*. Each neuron is fairly simple, containing only an activation function that causes the neuron to *fire* when the weighted sum of its inputs exceeds a threshold value. By connecting all of the neurons into a large network, and connecting them with weighted links, it is possible to form fairly complex relationships between the inputs and outputs (see Figure 2.21).

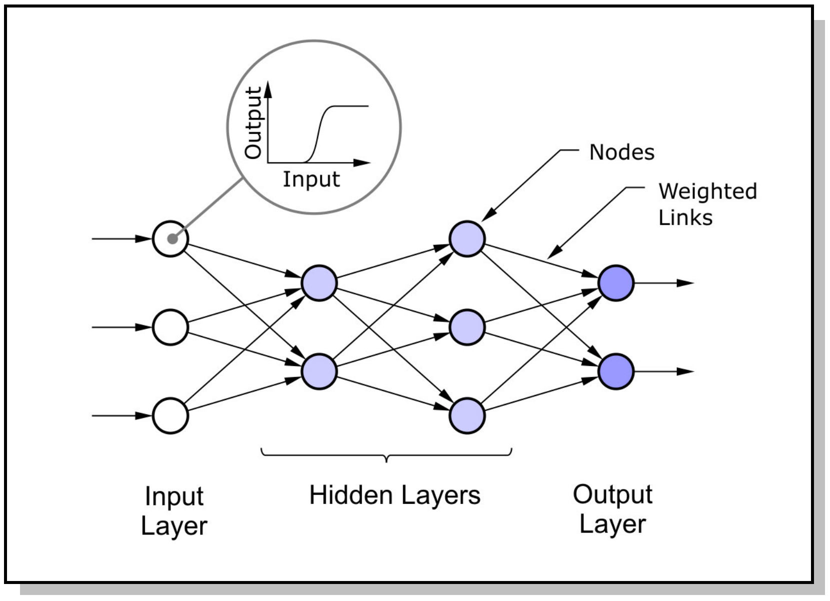


Figure 2.21: A generic neural network structure.

The main advantage of a neural network is that it is possible to use examples of inputs along with the desired output to train the network without knowing the relationship that links the data. Once trained, the network is able to produce outputs for new input data as long as it is similar to the training data. This technique is very useful in cases where the relationship between inputs and outputs is very complex and/or there is no expert knowledge available.

2.8.1 Network Structure

The term "neural network" covers a wide range of network structures and designs. One of the most common types consists of several layers of nodes. Input and output nodes are connected by one or more hidden layers which allow processing to occur. Data is transferred along nodal links to other parts of the network but the importance of each link can be modified by adjusting the link weight. The link weight will scale the value travelling along that path before it reaches the next node. Each layer of the network is computed in parallel and then passed on to the next layer. The process continues until the last layer where a numerical classification is output. This type of network is referred to as a feed-forward neural network (Mehrotra et al., 2000).

More advanced networks may have interconnections between nodes on the same layer and/or connections to previous layers. However, these types are more complex to process and train, as they require an iterative approach during evaluation in order to have the output settle to a consistent value.

2.8.2 Nodal Activation Functions

To evaluate each node, the input values are added together in order to determine a single value that will be used as an input to the activation function. The activation function can be any function that maps the input sum to an output between zero and one. Often this is a fixed threshold, but sigmoid functions are also common. Training is easier if the function is continuously differentiable as many of the training algorithms utilize the derivative (Mehrotra et al., 2000).

In addition to the links from previous layers, many neural networks also have a fixed input to each node. This input provides a bias to the summation in order to shift the activation point up or down. The value of the bias node is always set to one, and the weight of the link between it and the node is determined through the training process.

2.8.3 Training of the Network

The training of a feed-forward neural network is essentially a large optimization problem. The weights of the links between nodes are the variables, and the difference between the network output and the desired output is the error to be minimized. *Backpropagation* is one method to perform the optimization and works by making incremental changes to the link weights.

As the name suggests, the backpropagation method starts at the output nodes and computes the error in the network output. The amount each node has contributed to the error is then calculated and link weights are adjusted to attempt to correct the problem. Since so many parameters are being changed at the same time, each link weight is only changed by a very small amount so that the process is kept under control (Mehrotra et al., 2000).

2.9 Neuro-Fuzzy Classifiers

Fuzzy logic has been introduced as a way to mimic human ability to draw inference between known states. The neural network has also been discussed as a way to have some degree of learning from example data. By combining neural networks and fuzzy logic, it is possible to create a system that can not only learn but can also produce a smooth output between trained examples.

One of the problems with a pure neural network is that once it is trained there is little information that can be extracted on how the system is performing the calculations. Consequently, if the network is not performing as desired, it is difficult to improve without starting from scratch and training with new data.

In order to increase the ability to interpret the neural network, neuro-fuzzy systems impose a strict structure on the neural network. The layers of the neural network become the stages of the fuzzy logic system. This means that after training, the network parameters can still be extracted and examined.

The use of fuzzy logic with neural networks also means that the fuzzy logic parameters, instead of requiring expert input, can be trained through the neural network optimization process.

2.9.1 ANFIS

Since there are many different neural networks and types of fuzzy logic, there are also many types of neuro-fuzzy implementations. Adaptive-Network-based Fuzzy Inference System (ANFIS) is one implementation that has gained some popularity partly because it is included in the MATLAB® family of products.

The ANFIS structure as defined by Jang (1993) is quite general. However, the MATLAB® implementation is more restricted. For the MATLAB® ANFIS, the adaptive network consists of a five layer feed-forward network as shown in Figure 2.22, all of the links in the network have unity weight so the network is only changed through changes to the connection pattern and the functions inside the nodes.

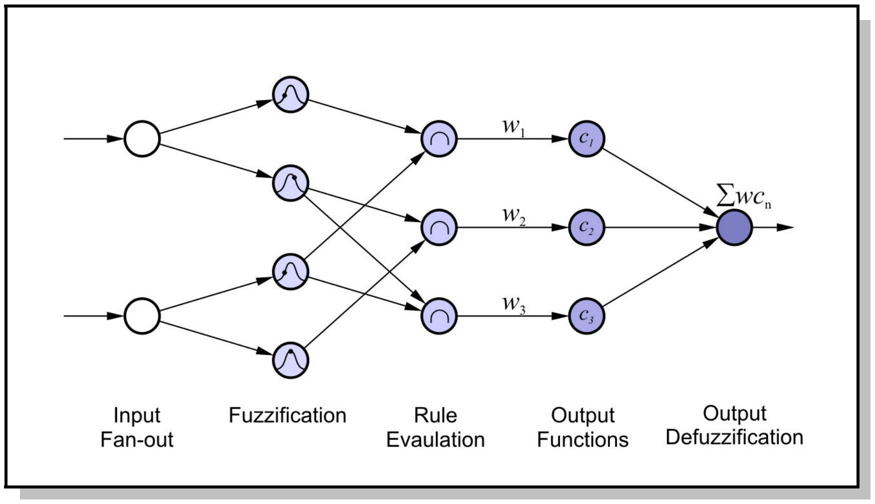


Figure 2.22: The parts of the ANFIS network

The first layer is a *fan-out* layer which passes the input values to the nodes in the next layer without modification. The second layer contains the membership functions that convert the crisp input variables into linguistic variables (fuzzification). The third layer contains the rules. The connection of each rule node to the input membership function nodes determines the antecedent parts of the rule. The fourth layer contains the output membership functions. The third and fourth layers always contain the same number of nodes and are connected one-to-one. This is because each rule must have its own output function. The fifth and final layer combines the inputs from each output in order to form a single numerical output (defuzzification).

Through the training process, the parameters of the membership functions (Layer 1 nodes) can be changed as well as the value of the output functions or singletons (Layer 4 nodes). The number of membership functions, which membership functions are connected to which rules, and the number of rules is not changed by the training algorithm. Because of this, it is important to develop a good prototype network before training begins.

2.10 Creating a Neuro-Fuzzy Network

A key step in developing a good prototype network is creating an appropriate number of membership functions. Well defined membership functions should each define some characteristic of the input data. They should also cover the used portion of the input space with some overlap but be distinct enough that two membership functions do not represent the same data. When two membership functions cover the same area, it is difficult for the training algorithm to differentiate which function should be modified. It also makes it difficult for later interpretation of the rules as two rules may have different antecedent membership functions but actually be acting on the same data.

The most common methods of creating membership functions are grid clustering, discrete incremental clustering, and subtractive clustering.

2.10.1 Grid Clustering

The simplest method of distributing membership functions is to create a grid. Each dimension of the input space is divided into equal size portions and a membership function is applied to each division. This ensures that the functions are evenly distributed and properly overlapped. However, it does not account for any pre-existing distribution patterns in the data, so some areas may be over-segmented while others lack division.

2.10.2 Discrete Incremental Clustering

To better consider the data distribution, Discrete Incremental Clustering (DIC) can be used (Wang et al., 2004). DIC works by taking each data point from the training data and deciding if a new membership function is required (see Figure 2.23). If an existing membership function already covers the new data point, then either no changes are made, or the existing set is expanded to further cover the new point. If the new point is outside any existing functions then a new membership function is created. Limits can be set on the amount of allowed overlap between functions, the maximum function size, and the rate of function growth. Because the DIC algorithm considers points consecutively, it cannot consider the overall distribution.

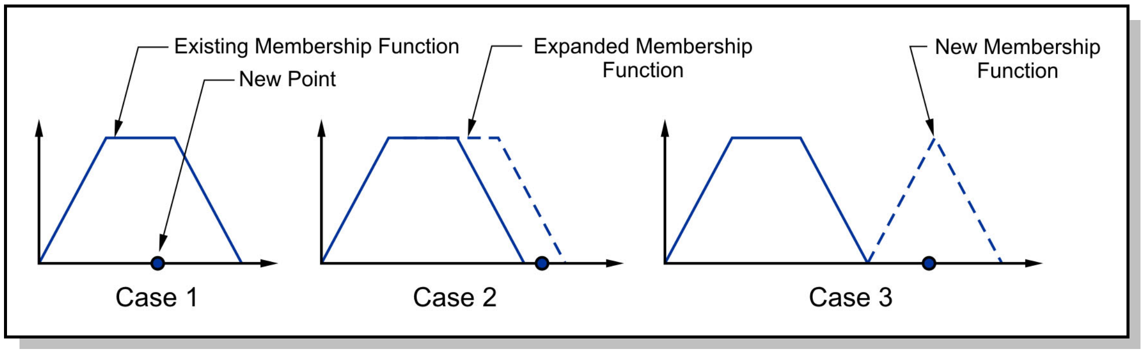


Figure 2.23: The three possible cases for the DIC method: (1) the existing membership functions are sufficient, (2) existing membership functions are expanded to cover the new point, (3) a new membership function is created.

2.10.3 Subtractive Clustering

Subtractive clustering looks for overall clusters in the data. Subtractive clustering is a modification of the Mountain Clustering method (Yager and Filev, 1994). As shown in Figure 2.24, the concept is to consider densely populated regions of the dataset as *peaks* or *mountains*. The highest peak is identified and considered as the first cluster centre. All the points that are within the *radius of influence* of the first peak are then suppressed so that two very close peaks are not both selected. This helps to avoid excessive overlap of the clusters. With nearby peaks suppressed, the next highest peak is selected and stored as a cluster centre. The process is repeated until no significant peaks remain.

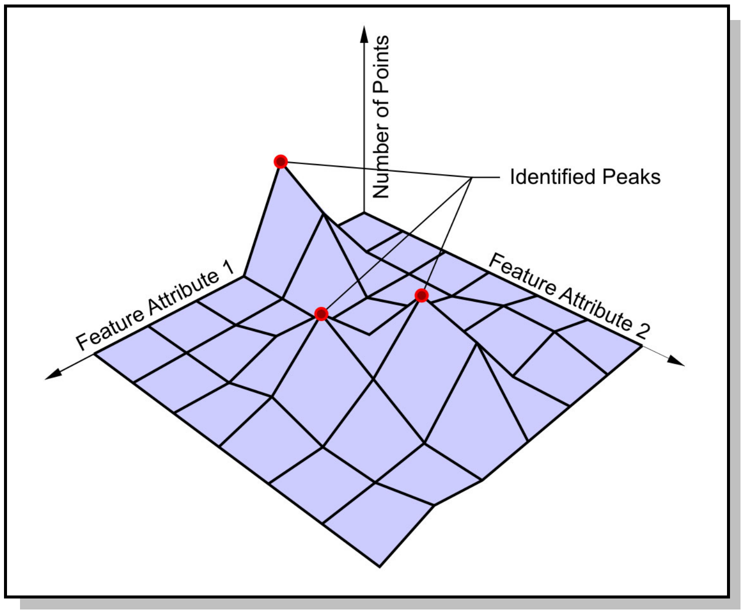


Figure 2.24: Concept of the Mountain Clustering Method which is the basis for Subtractive Clustering. The height of each peak is related to the number of data points which fall into that region of the grid.

For each cluster centre that is identified, a membership function is created in each dimension that is aligned with the centre of the cluster. When forming an ANFIS structure, a new rule is also created with the new membership functions as antecedents. This approach maintains a link between the data, the membership functions, and the rules.

2.11 User Interface

The earliest AVI systems were programmed in a text-based environment and provided little visual feedback to the user about what was being inspected. However, it became clear that improved productivity and acceptance could be achieved with graphical user interfaces (GUIs). Some early work in this area was done by Hunter, Graham, and Taylor (1995). Their work focused on an intuitive description language for defining the feature extraction and a graphical display of the extracted points and lines.

Graphical interaction is now common in human-computer interaction and is expected as the norm in most software packages. In AVI processes, the graphical nature of the data makes displays even more important. It also reduces the amount of time required to setup the problem by allowing point-and-click operation.

2.12 Previous Work

The use of soft-computing techniques to improve AVI systems is not new. It has often been applied in situations where traditional methods have not produced satisfactory results. Typically, however, this is done on a case-by-case basis and does not generalize well to other applications.

The seaming of food cans was the focus of an inspection system developed by Marino et al. (2001). Their paper is a detailed documentation of the process from identification of the needs of the problem to development of a hardware and software solution. The authors use traditional image extraction methods to keep up with the high speed process.

The cans are then classified based on a fuzzy network which was selected for its ability to mimic the reasoning of the human inspectors.

Lee and Bien (1997) applied fuzzy logic to match models to image scenes for the purpose of locating parts in random orientations. Both the model reference points and the matching algorithm utilized fuzzy logic in order to allow for some distortion of the data due to noise, occlusion, and changes in orientation. The use of fuzzy logic was an easy way to allow some variation between model and image without explicitly having to define the tolerance for each model and model point.

Pure neural network classification options were investigated by Pham and Bayro-Corrochano (1995) for the inspection of valve stem seals. This was a case where human inspectors were not able to inspect the parts satisfactorily due to the very small defects and poor contrast of defects with the rest of the part. By using a high number of feature parameters to measure the defects, they were able to successfully implement a multilayer perceptron type of neural network to classify the defects. However, they needed to use multiple networks in order to deal with the variation in the size of the defects observed.

The surface roughness of machined parts was predicted in work by Ho et al. (2002). The system used very general information from images of the parts along with information about the machining process. This data was combined with actual surface roughness measurements to provide a training set for an ANFIS network. The results showed that an

ANFIS-based system improved performance over a previous polynomial network-based system and gave an overall error of less than one percent.

Chen and Hoberock (1996) also implemented a neuro-fuzzy system for machine vision inspection, but developed their own neuro-fuzzy structure (termed FUZAMP). This structure is able to perform both online and offline training and is good at dealing with uncertain visual measurement data. To show the performance of the method, they used the system to classify cutlery in a commercial washing process. The new network obtained higher accuracy than other neuro-fuzzy methods, but also took significantly more processing time.

König, Genther, and Glesner (1993) developed a system that is most similar to the work of this project in terms of general applicability. Their system is a generalized structure which incorporates multiple methods for feature extraction, processing, and classification. The use of multiple neural networks allows the one that is most effective for a particular application to be selected. Neuro-fuzzy networks are not incorporated, but the structure has been developed in an open way such that new classifiers and feature extractors can be added.

Shafi (2004) published work on a very similar problem to the sample problem. The inspection involved verification of dashboard beam insert clips similar to the *push clips* from Van Rob. The paper provides a good insight into some of the trail-and-error steps required to train a traditional system using Cognex® software. In this case, the Cognex® PatFind tool was used to locate the clip centres and thus determine if the clip was present or not.

2.13 Summary

This chapter presents some of the key concepts which were investigated in developing the algorithm for this thesis. It also discusses some of the previous work done in the area of neuro-fuzzy AVI systems. AVI is a rapidly developing field and rapid advances both in hardware and software mean that there is a large body of work on the subject. For the most part, however, the papers deal with solutions to specific applications. Although these are important, it does not make the process of implementing a new system significantly easier.

Chapter 3

Algorithm Development

For the Van Rob problem (presented in Section 1.2), machine vision is not the only solution that will satisfy the inspection problem. Other methods such as contact-based systems (probes), laser range finders, photo-eyes, or acoustic sensors could also be employed. The advantage of Automated Visual Inspection over those (usually custom) systems is that AVI requires little in the way of specialized hardware. So, if an assembly line changes parts, the same camera and lighting can be used, and only the software needs to be updated. However, software programming for existing AVI systems can be slow and frustrating, especially for users not trained in image processing techniques (Garcia, 2006).

Another way to improve the chances of success with the Van Rob system is to alter the physical conditions. In the current setup, there are two other robots working in the area of the inspection system. These robots are not coordinated and so often enter or leave the image frame during image capture. The very bright lighting also causes strong reflections from the part. These reflections change depending on the angle and shape of the part. The background also changes since the camera sees through the part to the rest of the plant. Finally, the parts are held on a revolving carousel which has four fixtures. Each fixture is slightly different so the location of the part in the image is not always the same.

All of these problems have physical solutions that could be achieved with careful design of the inspection environment. However, these types of problems are commonly found in industry, especially at companies that are just starting to implement machine vision systems. It is therefore desirable to have an algorithm that can compensate for, or deal with, these environmental effects and make the AVI system a truly flexible solution.

For this project, each component of the processing algorithm was considered separately for its ability to contribute to a generalized, robust system. The reasoning for inclusion of each component is given in the following sections and the final algorithm layout is described in Section 3.7.

3.1 Feature Extraction

As discussed previously, high level feature extraction has many advantages over a pixel-based approach. The features are clearer to the user, and the number of inputs to the classifier is reduced. Since the time to train and run a classification system is strongly related to the number of inputs (Mehrotra et al., 2000), reducing the inputs can lead to faster processing. If the image were processed on a pixel by pixel basis, either the number of inputs to the network would become large or the network would need to be run on each pixel independently. The processing time issues associated with this approach were illustrated in the work by Norman (2005).

One notable exception to the problem with pixel-based approaches is techniques that use the entire image, for example, using a *golden image* to which all subsequent images are compared. By detecting differences between the reference image and the captured image, the system is able to detect errors. This method is very fast, but tends to be sensitive to changes in lighting, shadow, and part position. For these reasons, this approach was not pursued. For similar reasons, basic pattern matching methods were also not used.

The SIFT method has the potential to minimize the effect of appearance change as it can focus on matching individual points. However, in the case of these components, the SIFT method also had trouble making the correct matches. It is hypothesised that this is due to the similarity between features on the parts. For instance, the corner of a clip against a featureless metal background will be similar to the other corners of the clip.

Considering the issues with lighting and appearance change, a feature-based approach seemed to be a good solution. However, even after a feature-based approach was selected, particular feature extraction methods had to be selected. Although the purpose of this project was not to improve feature extraction techniques, much of the performance of the system is dependent on how well the features are extracted from the image.

3.1.1 Image Processing Methods

Many feature extraction methods are based on processing of a binary image. The main reason is that binary processing is very fast and is easy to implement into hardware. It also makes it easy to measure properties of regions such as area, perimeter, etc. without too much computational overhead. These attributes make binary processing popular for high speed inspection where there is only limited processing time available.

The main problem with binary processing is that the binary image must first be created from the greyscale or colour image from the camera. Typically this is accomplished using a fixed threshold value. Pixel intensities above this threshold are set to white and those below, to black. However, changes to the lighting, shading, or dust build-up can all contribute to changes in the overall intensity level that can disturb the threshold system.

Even with carefully aligned parts in a controlled environment, fixed thresholding can be problematic due to changes in the part surface or colour.

An improvement to fixed thresholding is to determine the threshold-based on the average value of the image, or the average of the local neighbourhood of the pixel. This *local thresholding* method allows for some variation in the overall intensity levels.

If the local thresholding technique is extended one step further, variation on a pixel level can be examined. This is the theory behind edge detection based on the image gradient. The gradient tends to be more stable than the actual intensity values under different lighting conditions.

Although the image gradient is fairly stable, it can still be challenging to select a threshold for determining if a gradient value should be considered as an edge point. Because the goal of this system is to provide improved reliability, algorithms were examined that relied as little as possible on thresholds, tolerances, or external parameters of any kind. Some of the simpler commercial systems meet these requirements, so initial feature extraction attempts were based on methods derived from observations of how commercial systems operated. These algorithms tend to be optimized for high speed processing on simple hardware.

It was also decided that no pre-processing would be performed on the image (such as changing brightness or contrast). Although these changes can be useful for pattern matching or human interpretation, they are less important when using a gradient approach. They also usually lead to loss of information from the image as upper and lower pixel values are clipped.

3.1.2 Line Extraction

A line feature is a linear grouping of edge features which in turn are areas of high gradient in the image. For feature extraction within a user selected region-of-interest, the feature extractor should select the most prominent line feature in that region.

If the approximate direction of the line is known ahead of time, fairly simple and fast processing methods can be used. As discussed above, these methods were based on observation of commercial systems (specifically the Cognex® system). The line extraction method seemed to be based on use of the gradient image. If the gradient is calculated perpendicular to the expected direction of the line, then it is possible to look for the column of pixels that has the highest total gradient sum (see Figure 3.1). Because the gradient sum can be both positive and negative, it is possible to isolate lines of different polarities. This is sometimes useful in differentiating between two edges that fall within the same search area.

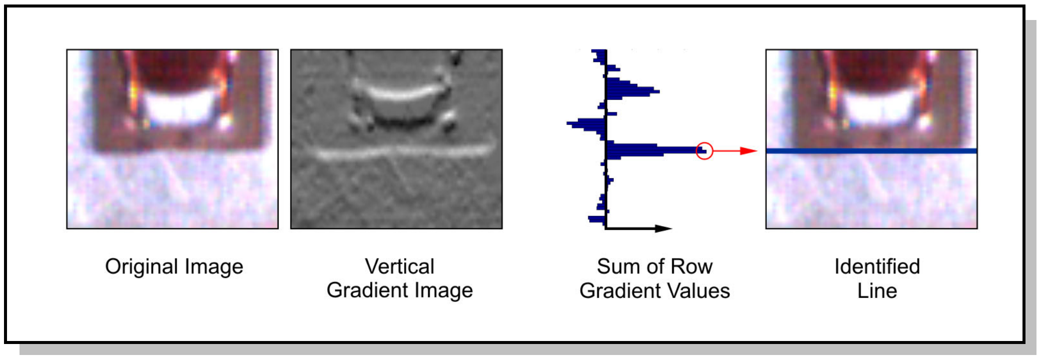


Figure 3.1: The gradient sum method of locating lines in the image.

Because the summations are evaluated by rows, the reported line will always be aligned with the original search box. However, in many cases, the line may have shifted by a few degrees from the expected value. This can be important information to the classifier.

To get a better idea of the true angle of the line, a modification was made to gradient summation method. Instead of finding the strongest column, the strongest point was found in each row. A line was then fit through these strongest points. The fitting could be done with either all points as equal weight, or be weighted by value. This method was able to detect small changes in angle for very simple images, but was sensitive to outlying maximum gradient points in real images (see Figure 3.2).

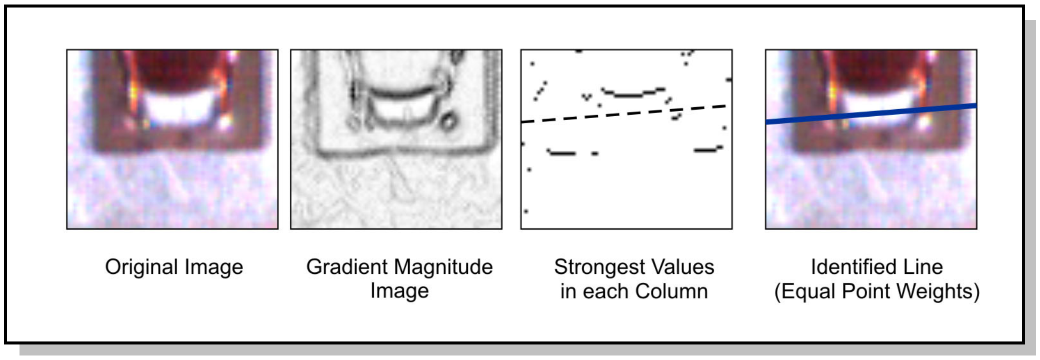

Original Image
Gradient Magnitude Image
Strongest Values in each Column
Identified Line (Equal Point Weights)

Figure 3.2: The gradient maximum method of locating angled lines in the image. Note the effect of ‘noise’ from other strong edges in the image that prevents the correct edge from being extracted properly.

For all of the techniques mentioned so far, if the line was expected to be at an angle, the image had to be rotated before processing began. Rotation of an image by an arbitrary angle is a fairly computationally intensive process and made a notable difference to the total processing time.

To avoid rotation of the image, a Hough-based approach was adopted. The Hough transform method involves mapping of edge pixels in the image into a Hough space where each point represents a particular line in the source image (see section 2.5.2 for details).

Standard Hough transforms act on a binary edge image such as that produced by the Canny edge detection method (see Section 2.5.1). However, this gave equal importance to all edge pixels regardless of their gradient values. It also required that thresholds be selected for the Canny method. Since these thresholds would have to be modified for each situation, a modified Hough transform was used instead. This Hough transform acted directly on the gradient values and added votes into Hough space using the gradient values as weights.

By using a Hough space that is large enough to represent all possible lines that could pass through the source image, it is possible to find the strongest line in any direction. Finding the strongest line consists of running a peak finding algorithm on the resulting Hough space image and correlating that peak to the appropriate line in the original image.

The line feature returns only three attribute values: the angle of the line (*T*) in degrees, the perpendicular distance from the origin (*R*) in pixels, and the *Score*. The score is evaluated as a percentage based on the number of votes the line gets in Hough space compared to the number of votes it would get if the edge was a perfect black to white transition.

For the Hough-based methods, the problem arose of how to restrict the range of angles if the line orientation was known. The solution was to process the entire Hough image, and then before the strongest peak was found, crop the Hough space image so that only the angles of interest are included. Cropping of the Hough space image was fairly simple in that the dimensions of the Hough space used were angle and perpendicular distance from the origin (see Figure 3.3).

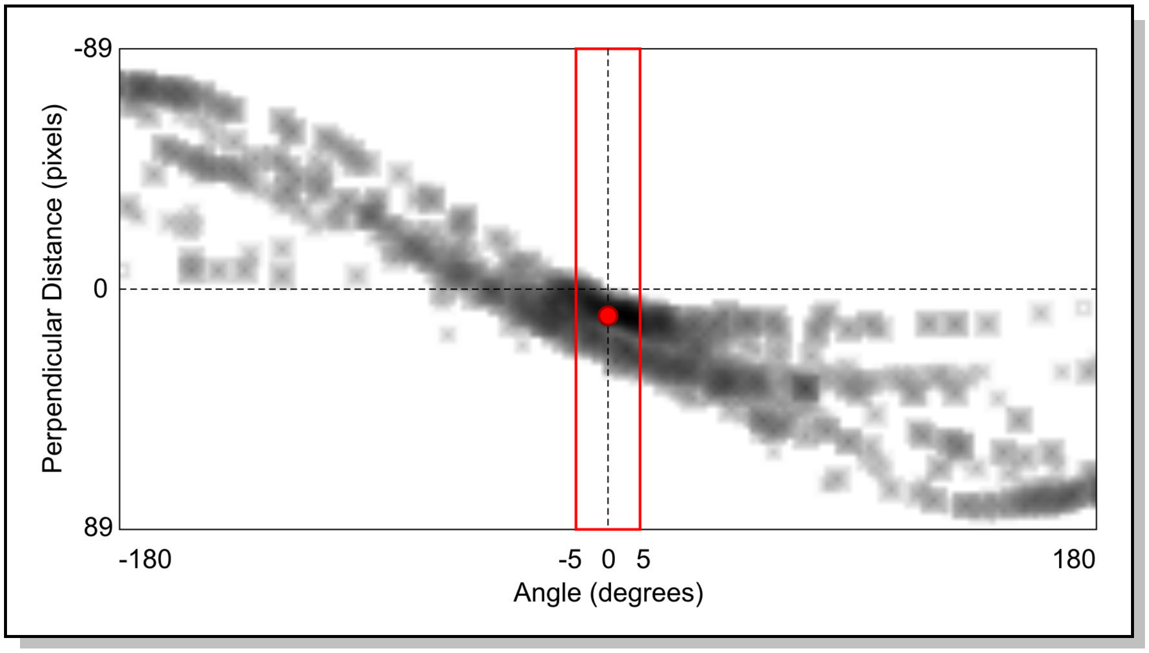


Figure 3.3: Cropping of the Hough Space image limits the allowed angle range of the detected line (in this example, angle is limited to ±5 degrees). The line that would be detected is shown by the location of the dot.

After some experimentation with the line extractors, it was found that in some images, the complex part shapes, reflections, noise, and low contrast caused the extracted lines to sometimes vary greatly for similar part positions. It was found that the problem was that the line was trying to generalize a very large area with several different edge locations. This is one of the problems with feature-based processing, the advantage of having a relatively simple feature representing a large number of pixels can mean that too much information is lost in the process.

One solution would be to reduce the size of the area-of-interest so that several smaller lines are extracted. However, this would be operator intensive. Instead, another feature type was added referred to as a *Squiggle*. Squiggles use the simpler gradient-based line extraction to find short line segments in the area of interest. The area is divided vertically and horizontally by a user defined number of divisions. The strongest line within those rows or columns is then selected. The feature returns a high number of feature attributes (position and strength for each line segment), but works well in regions where normal lines do not represent the image well. Unfortunately, it is harder for the user to interpret the results since the shape of the squiggle often appears to have little to do with the shape of the part. The Squiggle extraction process is shown in Figure 3.4.

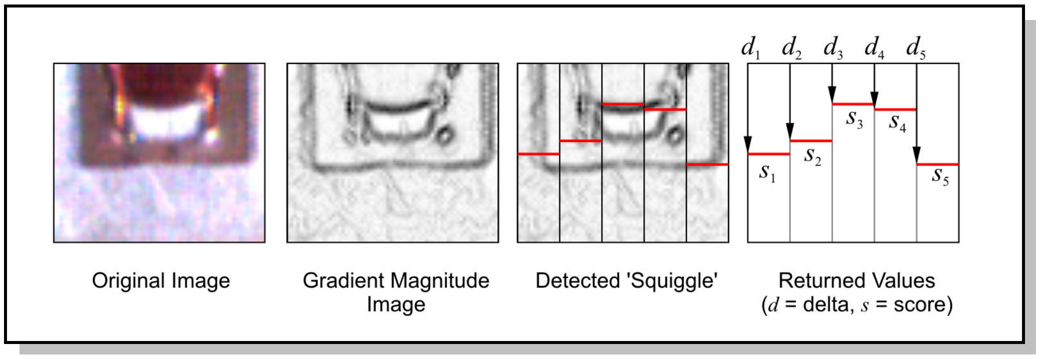

d_1 d_2 d_3 d_4 d_5
s_1 s_2 s_3 s_4 s_5
Original Image
Gradient Magnitude Image
Detected 'Squiggle'
Returned Values (d = delta, s = score)

Figure 3.4: The 'Squiggle' feature consists of a series of line segments extracted for discrete bands of the image. This type of feature tends to work well on smaller objects.

3.1.3 Circle and Hole Extraction

Circles and holes are extracted using a similar method to line extraction. The extractor is still searching for a region with strong edges to define the circle, but now that search must be done in a radial pattern.

Several attempts were made to produce a fast, reliable circle extraction algorithm. Again, the Cognex® system was used as a template. The first approach was to map the pixels in an annular region to a rectangular space using a sampling routine. The resulting distorted pixel image was then simply searched for a line to determine the radius of the circle (see Figure 3.5). This method worked reasonably well, but did not allow for the circle to have a centre anywhere other than the user defined centre location.

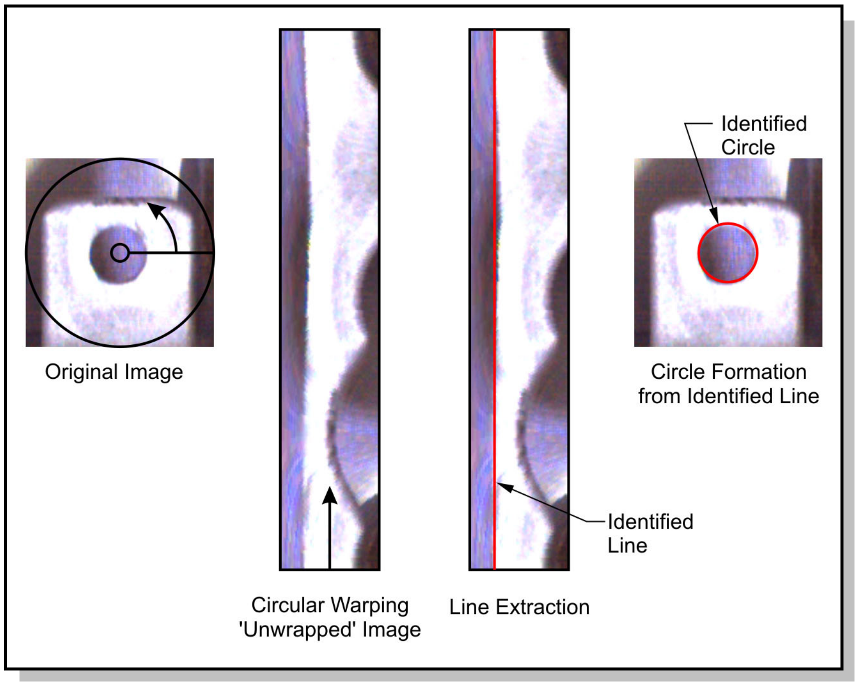

Original Image
Circular Warping 'Unwrapped' Image
Line Extraction
Identified Line
Identified Circle
Circle Formation from Identified Line

Figure 3.5: The process of detecting a circle using circular warping of the original image. By distorting the original image to a linear one, it is possible to use line extraction algorithms to find the strongest linear edge and thus the strongest circle.

A brute force method was used to fix the centre location problem by trying all possible centre locations that fell within the search area. This method correctly located the circle, but took much too long to be of any practical use due to the repeated mapping of the pixels that was required for each centre location.

Again the Hough transform was considered. In some ways a circular Hough transform is actually easier to understand than the linear case. If the radius of the target circle is known, then the process simply draws a circle around each pixel in the edge image. Where these circles intersect are the probable locations of circle centres (see Section 2.5.3 for details).

When the radius of the circle is not known, the usual process is to simply expand the Hough space into three dimensions where each layer represents a different radius. This means that for each pixel in the edge image, a cone must now be created in Hough space which is both memory and computationally intensive.

An alternate method is to use the directional information from the gradient map. Then, instead of drawing circles around each pixel, it is possible to draw lines through them perpendicular to the edge direction. Where all the lines overlap becomes the probable centre of the circle. Working backwards from the centre, it is then possible to determine the radius of the detected circle (see Section 2.5.3 for details).

For this project, two different circular shapes were defined: a “Hole” which has a known radius, and a “circle” which has unknown radius. Holes are extracted using the two dimensional circular Hough transform. A hole feature returns three feature attribute values: the *X* and *Y* location of the hole centre in pixels, and the *Score* of the hole (based on the number of votes in Hough space for that hole compared to what it would be for a perfect white circle on black background). Circles are extracted using the method developed by Peng (2006) and return four feature attributes, three similar to the Hole feature, plus the radius of the detected circle in pixels.

3.1.4 Colour Patch Extraction

Colour patches or 'blobs' are typically of limited use in the inspection of automotive parts due to the monochrome nature of most metallic assemblies. However, in this application, colour was clearly a strong indicator so a colour patch feature was included.

Colour is subjective and hard to define. Though digital imaging can give a numerical breakdown of the red, green, and blue components in the light, it is still challenging to relate these to acceptable colour ranges.

Part of the problem is that humans are very good at correcting for changes in lighting intensity and colour to determine the true colour of an object (Finlayson, 2001). Thus, two objects that may look very different to the camera could still be considered the same colour by a human inspector.

A colour extractor was first designed which averaged the red, green, and blue values over the region of interest and reported this colour. There were several problems with this approach. The first is that there are many strong highlights in the image which hit the maximum pixel value limits (255, 255, 255). These tend to dominate the average and make for very little variation in the colour data. The second problem is that objects in the image that may appear to be fairly uniform colour may actually be made up of pixels of several colours.

To deal with the multiple colour issue, the image was first blurred slightly using a Gaussian filter. Because only colour data is desired, not having a crisp edge to the objects is not a problem.

To better separate the major colour, a histogram approach was tried. A histogram summarizes an image by counting all pixels that fall into a particular category. In the case of a colour image, three histograms are produced, one for each colour band. Again, due to the strong highlights, the 255 value bin in all bands was highly populated. By ignoring this last bin, it was possible to neglect most of the highlights. However, it was found that it was difficult to determine a shift in the histogram values for the clip being present or not. Although there was some visible change, it would have been hard to get the algorithm to recognize this shift using simple characteristics such as the peak location.

Part of the problem was thought to be that the change was spread out over all three colour channels. To try to consolidate the change to one layer, the colour space of the image was changed to the Hue, Saturation, Value (HSV) System. The HSV system collects all of the colour data into the hue channel and so responds more directly to colour shifts.

The other advantage of converting to HSV space is that grey values can be ignored in the colour processing by only looking at pixels with saturation values above a certain threshold.

One of the problems with the HSV space is a tendency for the hue channel to be fairly noisy. In order to reduce the effect of noise in hue, the hue space was broken into very coarse segments. The user is able to select how many segments are used and whether a specific hue is being sought. When no specific hue segment is specified, the algorithm will report the hue segment that has the largest number of pixels in the image. As shown in Figure 3.6, pixels which have a high enough colour saturation are assigned to one of the hue segments. Pixels that belong to the same segment are grouped together to form regions (or *blobs*). The area of each blob is determined and the blob with the largest area is used to generate four feature attributes: *X* and *Y* location of the centroid in pixels, the mean value of the *Hue* segment in degrees, and the *Area* in square pixels.

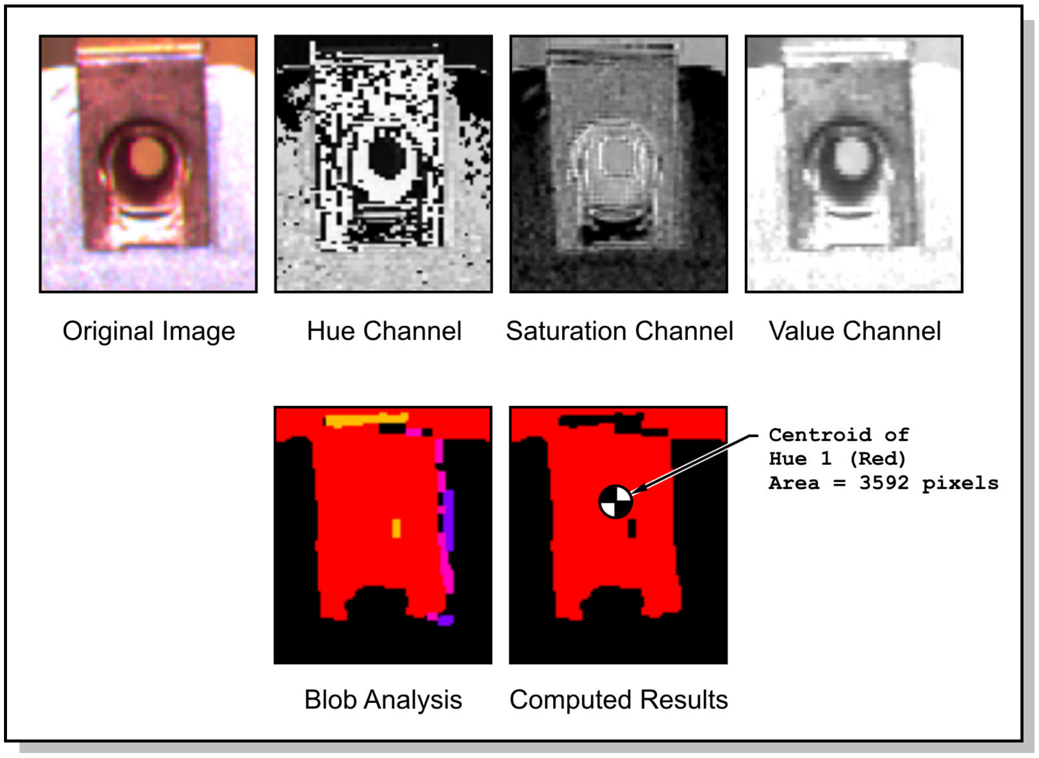


Figure 3.6: The process by which colour patches are extracted from the image. Pixels which reach the threshold for saturation are considered ‘colourful’ enough to be included when determining the area for each hue segment.

3.1.5 Locating the Feature Search Region

It was quickly found that one of the main problems with successfully extracting the features from the image was proper location of the search area. If the part in the image moved a significant distance, then the feature extractor would be acting on an area of the image that may not even contain the feature of interest.

This problem was partially solved using a software-based fixturing approach. Fixturing is a way of finding harder to locate features by first locating easier or more defined points on the same part. For instance, with the part shown in Figure 3.7, the edge of the metal clip may move outside the narrow area-of-interest window, but to locate the pin on the right side of the part, a much larger search area can be used because it is one of the only circular parts in the image. Therefore, the pin becomes the *fixture* for the edge feature.

After locating the pin, the position of the search area can corrected so that it covers the correct region of the part. If more features are used for fixturing, then it is possible to adjust search areas both in location and orientation.

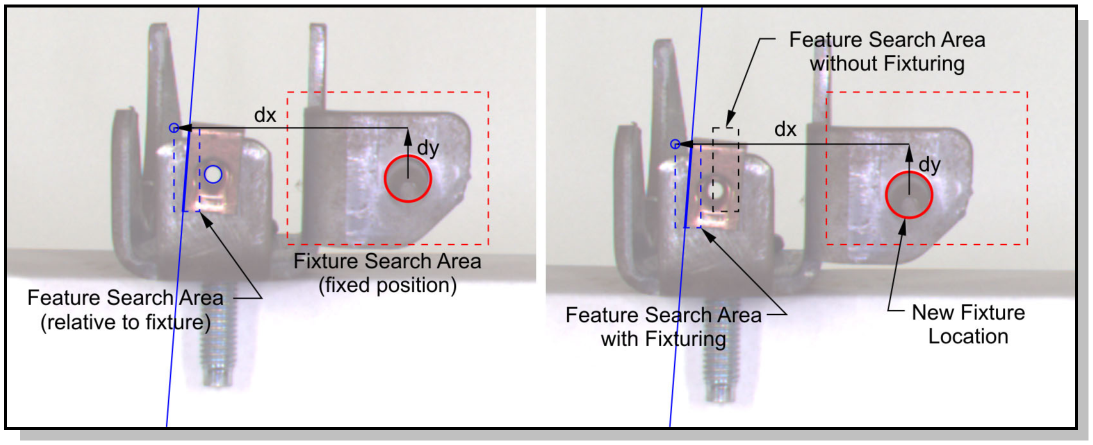


Figure 3.7: The fixturing process involves defining one or more feature search areas relative to fixture objects (left). When the fixture object moves, the feature search region will also be moved (right). In this example, if fixturing was not used, the edge of the clip would fall outside the feature search area.

One of the dangers of using a fixturing technique is that if the fixture is not correctly located, all other features that rely on it for alignment will not be extracted either. This was dealt with by using more than one fixture feature and taking the average position of those that have were found.

Later on in the project, it was found that many industrial applications have sufficient physical fixturing that the software-based fixture method was not required. However, for other cases such as inspection of parts on a conveyor, the random orientation requires that software fixturing be used.

3.1.6 Classification Methods

From the very beginning of this project, a neuro-fuzzy approach was selected as the classification method. The intent was to determine if the advantages of neuro-fuzzy systems would be beneficial in the area of AVI. In addition, the ANFIS structure and training methods were used for their relative ease of implementation. However, there were still several areas that needed careful consideration. Before the ANFIS structure can be trained, a prototype structure has to be developed. In order to give good results, the prototype should have an appropriate number of membership functions for each input. There should also be a rule structure in place that has the necessary links between rule nodes and membership function nodes. Finally, all of the operating parameters of the neuro-fuzzy structure need to be defined such as: the type of membership functions to use, the union and intersection operators, and the defuzzification method.

Since there were no other guidelines to use, these parameters were initially left at the default settings. Automatic structure generation is handled by MATLAB® if the ANFIS function is called without a prototype structure. The default settings are: subtractive clustering method to form the membership functions and define rules; outputs set to linear mode (outputs are linear combinations of the input values); product operator for combining the antecedent parts; and maximum value operator for combining consequent parts.

After much experimentation, it was found that these were not the ideal parameters for this problem. Some of the problems were due to performance issues, but most were because the resulting network was not transparent enough to the user.

The first problem was that using linear output functions made the training of the network very slow. It also meant that it was difficult to tell what each rule was doing. This is especially true in the case of a pass/fail problem where it makes more sense for the output of each rule to be contributing a constant value which drives the output either closer to a pass or closer to a fail outcome. By making the outputs of the rules constant values, it is also possible to look at the antecedent values for those rules and get some idea of what input configurations are leading to specific pass decisions.

The second problem arose once a confidence value was required on the output. As will be discussed further in Section 3.6.1, the confidence value is derived from the firing strengths of the rules. If the rules are combined using a product operator, it is more difficult to determine which antecedent of which rule is causing the low value. By changing the union operator to the minimum value instead, the offending antecedent can be quickly found and remedied.

The final problem was that the input membership functions were often too overlapped and did not clearly define one area of the data. This caused rules to be formed that were essentially the same, adding to network complexity, and reducing clarity of the process. The fix to this problem was to modify the clustering method. The changes made to the clustering, membership and rule formation processes are described in the following sections.

3.2 Data Clustering Methods

The feature extractors generate a feature vector for the image which contains all of the relevant feature parameters from the image. When all of the images are processed a feature matrix is formed. The goal of the clustering process is to create membership functions which are located such that related data points will fall into the same cluster.

Clustering methods assume that the data is grouped around meaningful relationships. For example, data points corresponding to pass images should have similar values to each other. However, they should also be distinct from data points corresponding to fail images. It is expected that each feature attribute will have some distribution about a mean value for each case. Not all quality control problems have this property. Consider the case of identifying if a hole is the correct diameter. If the manufacturing process is operating correctly, the expected distribution would be a single cluster about the correct mean value. At some point, the value would drop below a certain threshold that makes it no longer acceptable but this would be on the boundary of the central cluster (see Figure 3.8). In this case, clustering is probably not the best solution as both pass and fail points would be included in the one central cluster. The solution presented here is focused on problems which show multiple data clusters.

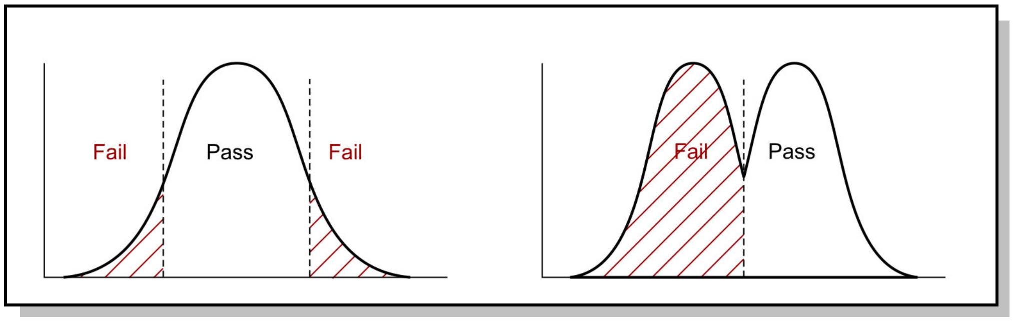


Figure 3.8: Uni-modal distribution (left) is common in dimensional inspection whereas bi-modal or multi-modal occurs in assembly inspection (right).

There are many different methods for creating the membership functions. Grid clustering, discrete incremental clustering (DIC), and subtractive clustering were investigated for this project.

3.2.1 Grid Clustering

Grid clustering is the simplest method as it divides all dimensions into an equal number of uniformly spaced functions. The amount of overlap and the number of members for each dimension is decided by the user. Grid clustering has the advantage that it maintains the clearest membership functions. For example, if the dimension being divided represented a length, then a human may break it into evenly distributed classes of long, average, and short. Such clear categories make it easier to develop rules and also to understand the operation of the final network. It is this relevance that is most difficult to maintain with the more complex clustering algorithms.

However, because there is no consideration for the actual distribution of the data, grid clustering often leads to either poor segmentation of the data of interest, or over-segmentation of regions of the input space that do not have any relation to the problem (see Figure 3.9). In addition, there is no way to determine which clusters should be connected by rules since they are not related to the data. Therefore, the common approach is to create all possible rules using combinations of the membership functions. The resulting network then has so many rules that the ANFIS algorithm has trouble training the network. It also becomes much harder for a user to interpret the final network.

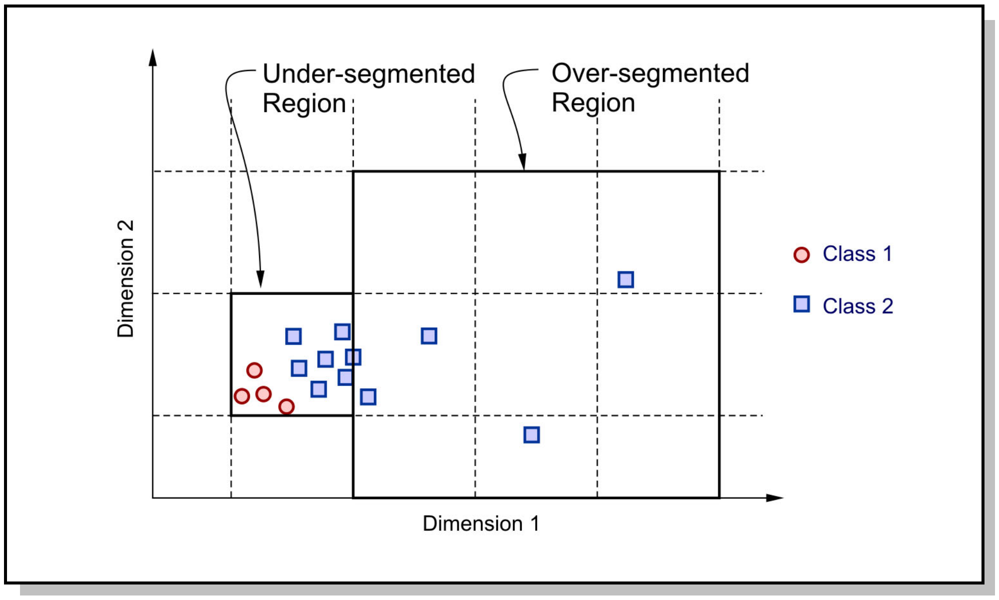


Figure 3.9: An example where grid clustering fails to capture the nature of the data distribution. The left-hand region does not contain enough divisions to separate the two classes while the right-hand region has many divisions but only contains one class.

3.2.2 Discrete Incremental Clustering

The DIC method is driven entirely by the data itself. The data grows the membership functions be either supporting an existing member, or starting a new cluster. Restrictions on growth help to keep functions from overlapping excessively and becoming indistinct. On the surface, this appears to satisfy the ideal requirements by maintaining clear functions while covering the input space well. The main problem with the DIC method however, is that it is very sensitive to the order in which the data is presented. For example, consider a uniform distribution of data over the input space. If the data is presented in ascending order, then a single membership function will be grown and expanded by each subsequent point. However, if they are presented in a more random order, multiple members will be created. Although some of this problem can be reduced by merging membership functions at later stages, this method was considered too variable for this application.

3.2.3 Subtractive Clustering

Subtractive clustering is the default clustering method used by MATLAB® to develop the neuro-fuzzy structure. The advantage of this method is that it uses the data to develop the clusters, but does so in a more global way than the DIC method. The result is that the clusters are more consistent and more representative of actual groupings in the data.

As mentioned previously, the main problem with subtractive clustering is that when a cluster is identified, a new membership function is created for all dimensions. Consider the case of a simple 2D data set containing two clusters, as illustrated in Figure 3.10. The members created along the first dimension are a good representation of the data. However, the second dimension now contains two clusters which cover the same input space. The training process will have difficulty with this network because it will be hard to determine which membership function should be used for data-points in the second dimension.

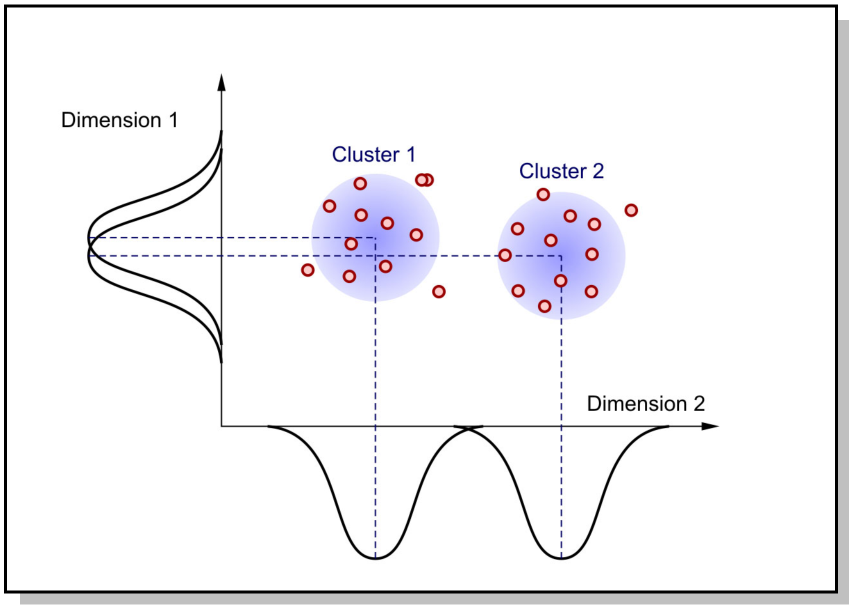


Figure 3.10: Subtractive clustering can create multiple membership functions that are redundant since it creates a new membership function for each identified cluster.

To make the network resulting from subtractive clustering more relevant, the clusters can be post-processed by a conditioning algorithm. The conditioning algorithm sequentially tests the clusters for each dimension. If the value of an adjacent membership function evaluated at the current member centre point exceeds a threshold, then the functions are merged to a new function at a location midway between the existing members. Typically, a threshold of 0.5 keeps the functions distinct while maintaining the original clustering pattern. Figure 3.11 shows the result of running the reduction algorithm on one of the dimensions of a neuro-fuzzy network.

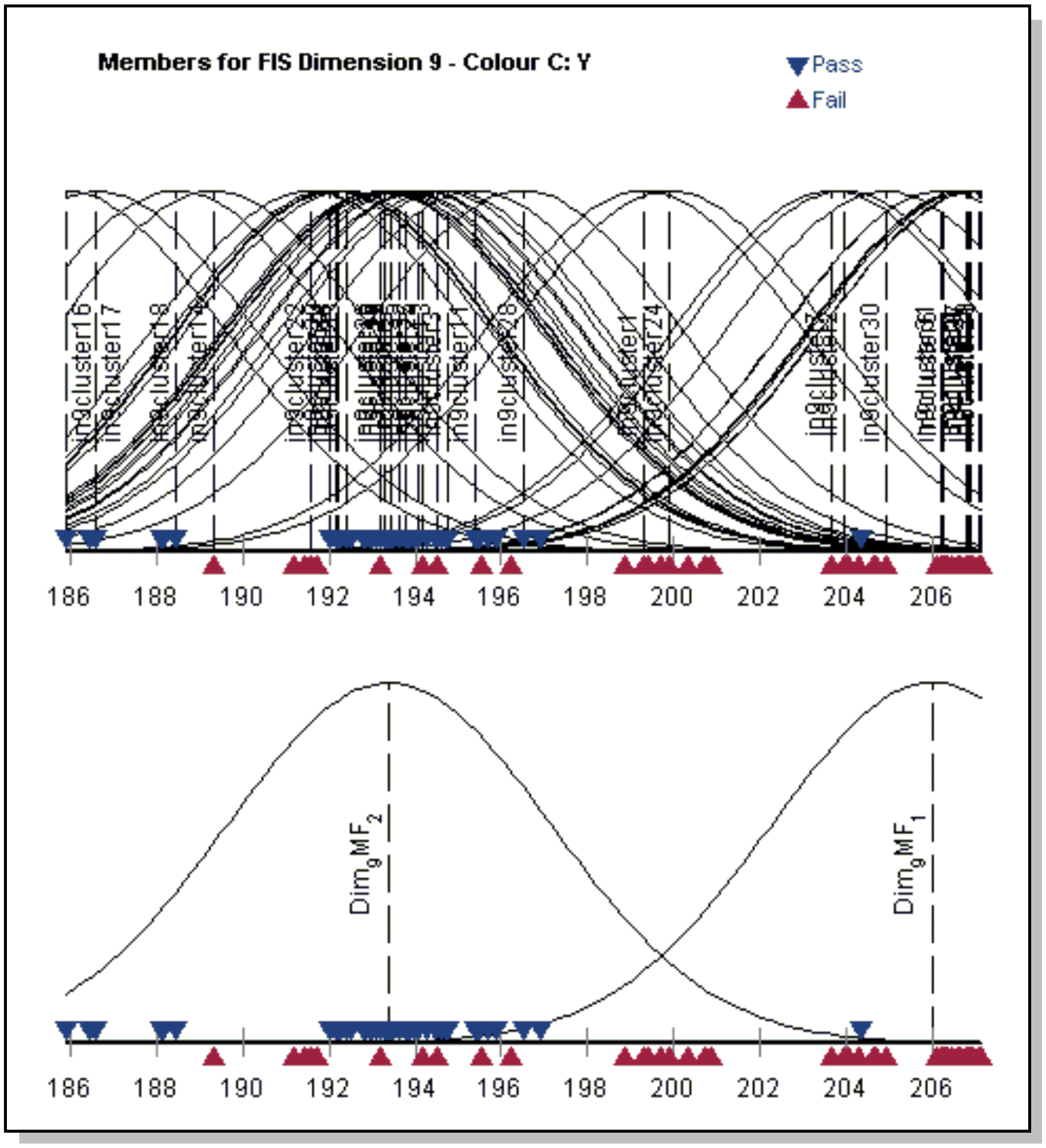


Figure 3.11: Membership function complexity is reduced using grouping methods. In this case, the original 32 membership functions generated during subtractive clustering (top) can be reduced to only 2 (bottom) while still capturing the underlying data distribution.

3.2.4 Formation of Fuzzy Rules

Using the subtractive clustering technique makes it easy to form appropriate rules. Each time a cluster is identified, a rule is created for that cluster. As discussed in the previous section, subtractive clustering creates a new membership function in each input dimension for each cluster identified in the data. The rule for that cluster then has antecedent parts consisting of the newly created membership functions.

Rules can also be formed through another technique which uses the training data. This process requires that a structure has already been formed with membership functions for the input dimensions. The input membership functions are then 'activated' by fuzzifying the input data and observing which membership functions have the highest degree of association. The strongest functions in each input dimension are then connected to the antecedent side of the rule.

The consequent side of the rule is created by following a similar process but with the output data and membership functions. If singleton values are being used as the outputs, the singleton value is set to the target output value from the training data set.

This method of generating rules was adapted from the work by Wang et al. (2004). It is particularly useful for re-forming rules when the clustering has already been completed, for example, when optimizing the network as described in Section 3.4.

3.3 Training the Neuro-Fuzzy Network

The training process is designed to manipulate the location of the membership functions and the value of the rule outputs in order to minimize the classification error. The training algorithm is given training data consisting of both good and bad data which is labelled by assigning an output of zero or one to each data point. The training method used for this project is the ANFIS method as implemented in MATLAB®. This method uses a combination of traditional backpropagation and a *least squares* technique.

Because the training method is iterative, it requires multiple passes through the training data. One pass through the training data is called an epoch. In order to find the best solution, enough epochs must be performed to converge to the minimum point in the error surface. However, if too many epochs are performed on the same data, then the network will become over-trained on that particular dataset and perform poorly on new data.

In order to avoid over-training, the training method is given two datasets. One is the training dataset and is used to compute the required adjustments to the network. The other set is the checking set and is used to evaluate the performance of the network after each epoch. Typically, the training dataset is about 60 percent of the total dataset and 30 percent is used for checking. Since the checking dataset is used to evaluate the success of the training process, it has some influence over the results. To observe the performance of the network on data it has never seen, the remaining 10 percent of the data is withheld for testing purposes.

When the error of the network based on the checking set is reduced to a minimum value, the network is getting over-trained, and training should be stopped. The ANFIS method in MATLAB® automatically handles the checking procedure, but it still requires enough epochs to be specified to find the minimum checking error. For this project, the number of epochs was found experimentally by observing the error over the training process. From error plots similar to Figure 3.12 it can be seen that after an initial period of improvement, the error settles down to a steady-state value. For the data sets encountered in this project, this point was always achieved within 500 epochs.

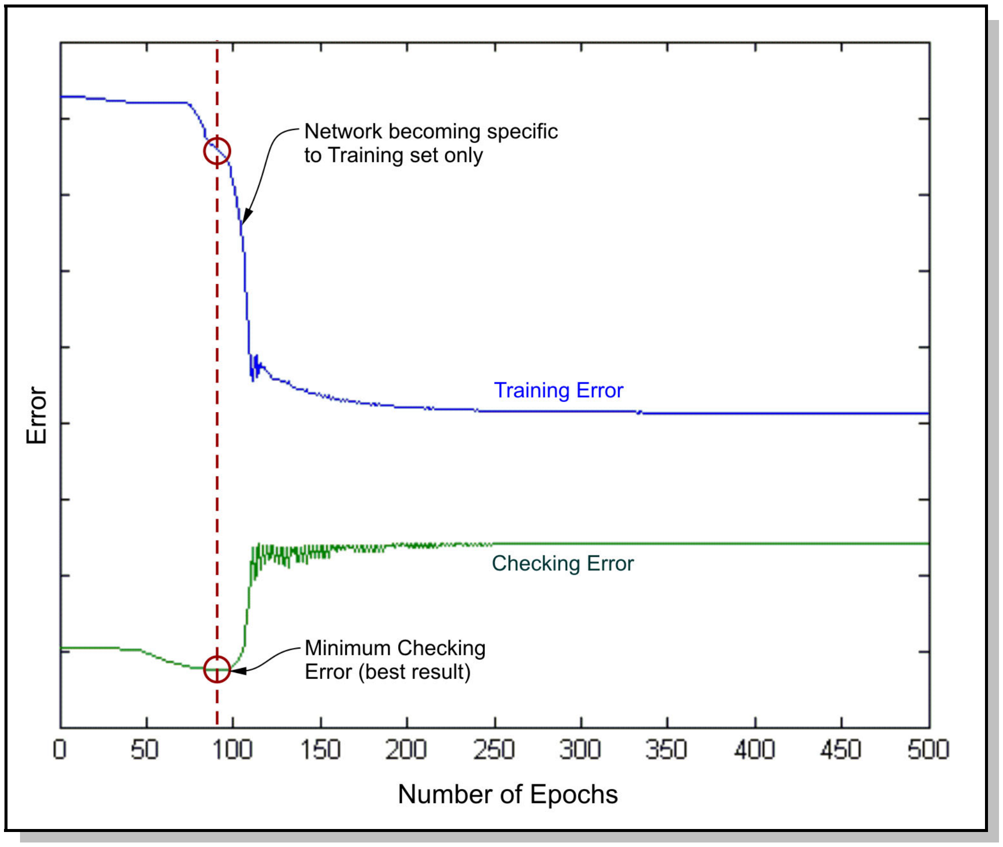


Figure 3.12: Training of a neural network can proceed too far so that the network is too specific to the training set. By holding some data back to use for checking, training can be stopped when the network stops improving the checking result.

3.4 Optimizing the Neuro-Fuzzy Network

The training process proceeds by finding the minimum error possible with the given set of feature attributes. However, in many cases, much of the data does not help in the decision making process. By including irrelevant data in the training set, the network may not achieve the same performance as with a smaller, more relevant dataset. In addition, removing feature attributes from the problem reduces the processing required both in the network and in the original feature extraction process.

Feature selection is the process of deciding which feature attributes to include in a subset in order to achieve the best performance (Kohavi and John, 1997). There are two main approaches to the problem.

The first method analyzes the feature attributes and tries to make some statistical determination of how important that feature attribute is to computing the correct outcome. The main advantage of the method is that it only takes one iteration. However, computing the relevance of the feature attribute without re-computing the network parameters can lead to a false impression of the importance of a particular feature attribute. In addition, feature attributes which are equally important may be removed as they show up as redundant information (Levi and Ullman, 2006).

The second method is referred to as the wrapper method because the optimization algorithm forms a 'wrapper' around the training algorithm. The principle is to change the feature attributes included in the dataset, retrain the network, evaluate the performance, and then decide on the feature attribute subset to try next. The process of optimization then becomes one of searching for the minimum network error across the space of all feature attribute combinations. This method is much more computationally intensive than the statistics-based approach since there are potentially $2^n - 1$ combinations to try.

An exhaustive search tries all of the possible combinations and is guaranteed to find the minimum error. However, it also takes the longest amount of time. Other search strategies can be used to reduce the amount of time taken for the search, but with the possibility that they will not find the true minimum error or best feature attribute subset.

For this project, a forward search was used because of the simplicity of implementation. A forward search starts with an empty set and first works through the set of all single feature attributes. The best feature attribute is then selected and combinations of two features are tried. This process continues until no further improvement is possible, or all features are used. Because the best features are kept from the previous stage, there are only $n - s$ combinations to try at each stage (where s is the stage number and n is the number of feature attributes) so the maximum number of iterations is $n(n + 1)/2$.

The forward search method was compared to the exhaustive search for one of the example data sets. It was found that both methods were able to arrive at similar optimized solutions. The exhaustive search found a better feature attribute set, but the improvement in overall accuracy was only 0.5% above the solution found by the forward search. The exhaustive search took 3472 iterations to reach the same solution as the forward search found in 70 iterations.

3.5 Rule Re-formation

Once the optimal set of feature attributes have been found, the network needs to be updated for the new number of inputs. During the optimization process, unused inputs are just suppressed but not actually removed from the network. For the final configuration, however, only the needed feature attributes should be included so no unnecessary feature extraction is performed.

All unused inputs to the network are first removed along with any membership functions. At first, all rules were then removed and then re-formed using the data-driven rule formation process discussed in Section 3.5. However, it was found that equally good results could be achieved by simply modifying the existing rules.

First, antecedents referring to the unused input membership functions are removed. Then all the rules are checked to make sure that no duplicate rules exist. Any redundant rules are merged into one along with their associated output function or singleton.

3.6 Online Operation

A network that has been developed based on the data, optimized, and re-trained, is ready for implementation into an on-line system. The on-line system evaluates which features need to be extracted from the image and which feature attributes should be passed on to the neuro-fuzzy network. These values are then processed by the neuro-fuzzy network in order to decide on a classification for the image (pass or fail). The classification decision is made based on whether the output of the network is closest to one or closest to zero, which were the training values for pass or fail respectively.

3.6.1 Confidence Evaluation

It would be tempting to base the confidence of the decision on how close the output was to the limits. However, there are problems with this approach.

Recall that the output of the network is based on a weighted average of the singleton outputs for each active rule. The weight of each singleton is based on the firing strength of the associated rule. However, consider the case where two rules are firing – one with a singleton output of one and the other with an output of zero. If the first rule is strongly fired, and the second weakly, the output will be driven strongly towards a value of one. Conversely, a weak first rule and strong second rule drives the output to zero. If the rules are equally fired, then the output would be 0.5 and it would be valid to consider that the decision is uncertain.

However, consider the case where both rules are very weakly activated. This means that the condition is unusual and is not well covered by either rule. Since both weights are small, it is easy for minor changes in either one to strongly affect the output and make it appear that the decision is confident when in fact it is not.

To get a better picture of the confidence of the decision, another measure is required. It was decided that inputs that are far from any trained pattern should be considered low confidence. Although the training set is not explicitly kept, the information from the training processes is summarized by the membership functions. If no data was found in a particular region of the input space, then no membership functions will cover that area and/or no rules will have been formed to cover that situation. Therefore, it is possible to determine how close a set of input data is to previously trained data by looking at the firing strength of the rules. Thus, the confidence value reported during online operation is the maximum rule firing strength in the network.

3.6.2 Detection of Unusual Data

Using the rule firing strength as the confidence value also allows a threshold to be set on the minimum rule firing strength. If the rule strength is below the threshold then the current input pattern can be considered significantly different than the trained pattern.

Being able to detect unusual patterns is very useful to the classification system because it allows for a third type of decision to be made – uncertain. With traditional systems, the image will be classified according to the existing rules even if those rules were developed based on very different data. The neuro-fuzzy system allows the operator to know when something has changed from the training data and so either the processes should be checked or the network should be modified to deal with the new data.

3.6.3 Network Modification

Modifying the network to deal with new data requires a balance of accommodating the new data without destroying the network's ability to correctly process the old data.

Note that this assumes that the old data is still relevant. If the process has changed completely, then presumably the network would be re-formed from scratch with a new example set. It also assumes that the correct classification for the new data is available from the operator.

One method of retraining an existing network is to maintain a database of all of the images that have been seen by the system so far. When new data is obtained, the database, along with the new data examples, can be used to retrain the system. Because there is a finite amount of memory/storage that can be used for such a process, it becomes necessary to decide which criteria will be used to discard images from the set. Some systems used a fixed buffer size such that only the last *n* images are kept. Other methods use more complex criteria such as maintaining as diverse a population as possible within the dataset.

In a neuro-fuzzy-based system it is possible to maintain some history without explicitly retaining any images. As described in Section 3.6.1, a generalization of the training data is stored in the input membership functions. Thus, retraining with new images then becomes a matter of defining how much the new data is allowed to modify the existing network.

There are four types of network modification that are possible: limit modification, changes to membership functions, addition of rules, and changes to the output singletons. The last is not included in this work as it requires more complex processing to make sure that old decisions are still valid.

Limit modification is the simplest change to make. Each dimension in the input space has an associated set of limits that are a literal record of the range of values that were seen in training. This ensures that data outside the training set is not processed. Often, the data seen in the real images is just off one end or the other of the training data range. Small modifications to the limits can be made safely to allow new data to be processed.

Changes to the membership functions are the most difficult to make without disturbing the balance of the network. Changes to the membership functions are made based on the antecedent part of the strongest firing rule. Since the rule firing strength is based on the minimum degree of association of the membership functions in the antecedent part, it is possible to determine which membership function is 'responsible' for a low rule firing strength (see Figure 3.13).

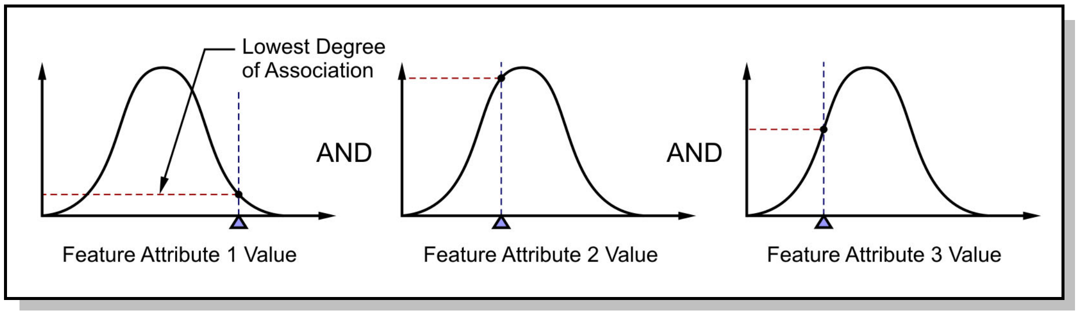

Lowest Degree of Association
AND
AND
Feature Attribute 1 Value
Feature Attribute 2 Value
Feature Attribute 3 Value

Figure 3.13: Assigning blame to a particular membership function is simply a matter of determining the antecedent member with the lowest degree of association.

This membership function can then be adjusted so that for the current input, the degree-of-association is sufficient to fire the rule above the minimum confidence threshold as shown in Figure 3.14.

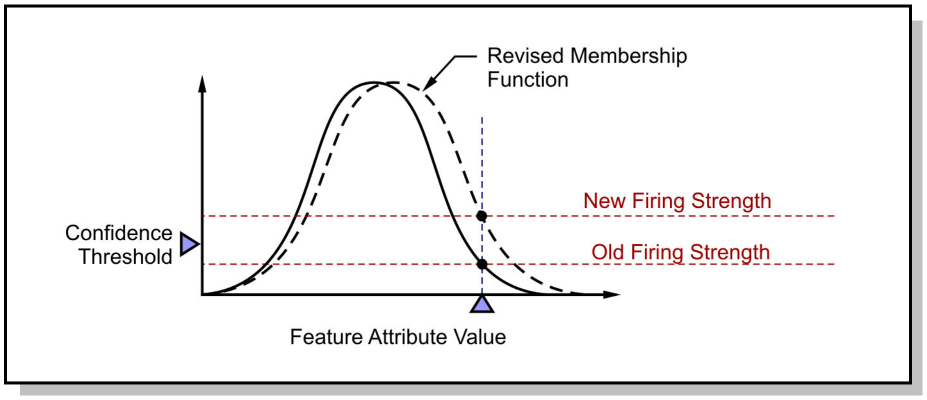


Figure 3.14: Updating of a membership function with new data is accomplished by moving the function slightly so that the firing strength requirement is satisfied.

In the current design, membership function changes are only made if the original degree of association is above a fixed limit. This ensures that the new point is not too far away from the membership function that is being changed.

Adding rules is the final modification that can be made. If the membership functions with the highest degree of association to the current input are not part of the antecedent of a rule, then a new rule can be formed. The new rule is created with a consequent value equal either one or zero, depending on the classification entered by the operator.

In order to provide some level of protection against incorrect operator classification, the strongest firing rule is first checked to see if the output singleton is closest to one or zero. If the output of the strongest rule is in conflict with the classification submitted by the operator, then a warning is issued to double check the decision.

3.7 Final Design

The final algorithm is compiled into a single software package given the name “QVision”. Within the QVision system, there are three main modules as illustrated in Figure 3.15.

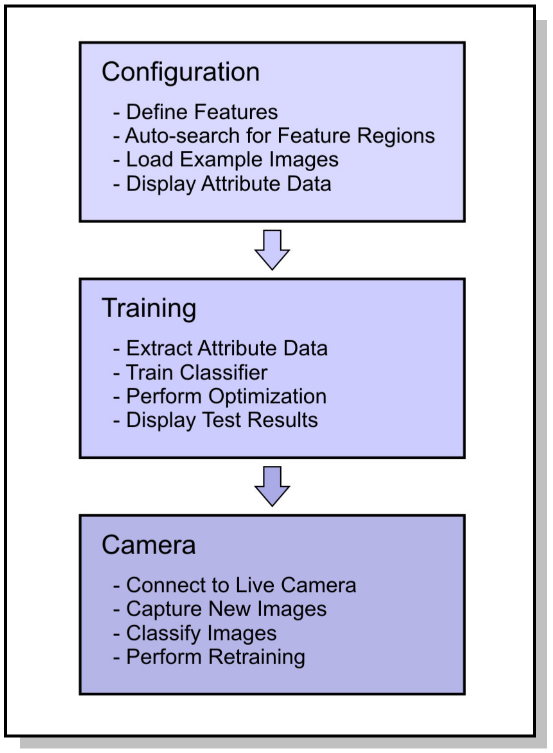


Figure 3.15: The three major modules that make up the QVision system

A configuration module allows the user to select feature extractors and define the area-of-interest for each feature. Once all the feature extraction parameters are set, a training module creates clusters based on the data, forms the neuro-fuzzy structure, and trains the network using the training images. Optimization can also be performed in this module in order to speed processing and possibly improve the performance of the classifier. The final module is the online operation module where new images are classified. The confidence of the decision is reported and if low confidence values occur, the operator has the option of updating the network by providing the correct classification.

Each part of the algorithm is written into a separate subroutine, so it is possible to reconfigure the design of the system at a later time, improve the feature extraction methods, or include a new classifier. Appendix A gives a summary of the methods that were selected for each of the major algorithm components. Also listed are the MATLAB® data structures and subroutines (m-files) developed for this project along with a brief description and the calling syntax.

3.8 Summary

In this chapter, the rationale for using a software solution to AVI problems was discussed. Software solutions allow smaller companies to be able to use AVI without numerous iterations of hardware design. The need for good feature extraction systems was recognized as a key factor for good algorithm performance and suitable extraction algorithms were selected accordingly.

For the classification system, the details of implementing the neuro-fuzzy classifier using the ANFIS system were presented along with the methods that will be used to create, train, and re-train the neuro-fuzzy network. A summary of the ANFIS approach to the problem is given in Killing et al. (2006).

Finally, all of these components were brought together into a unifying structure which, together with a camera and lighting, forms a complete AVI system.

Chapter 4

User Interface

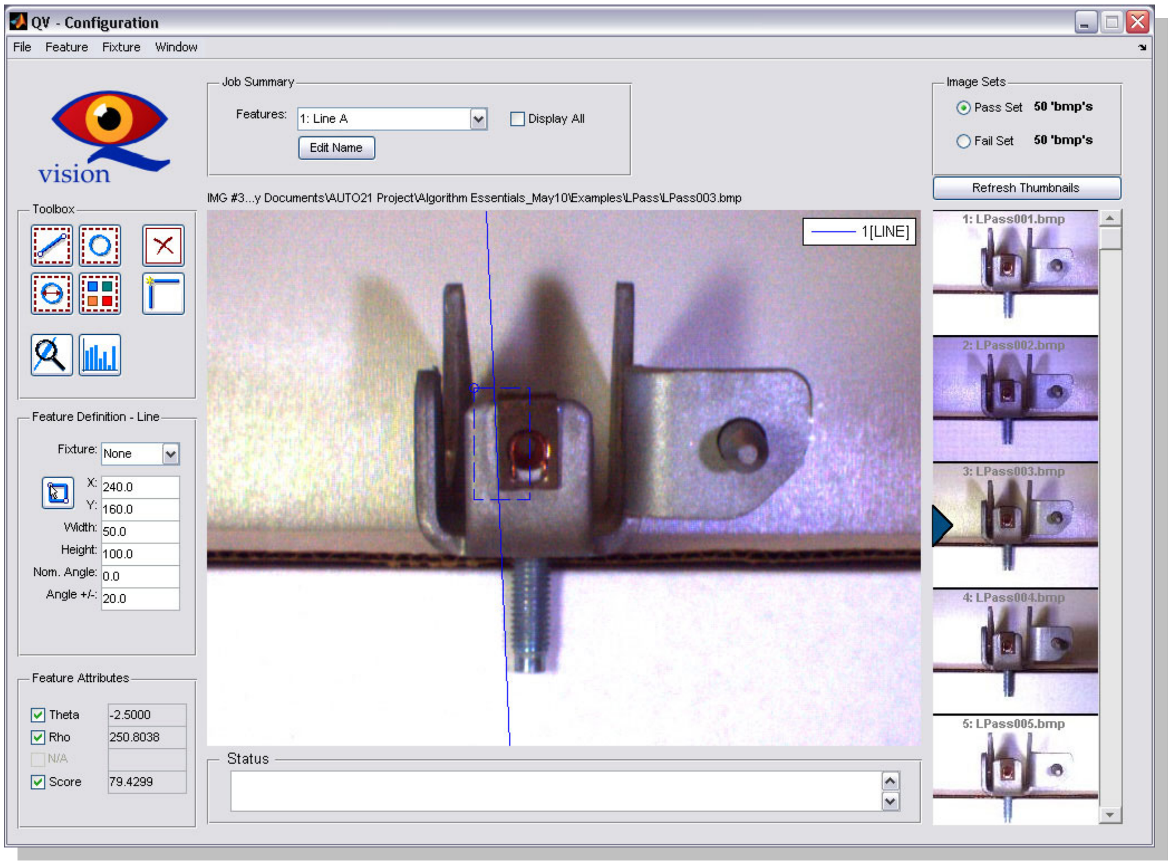
In this work, it was very important that the logic of the algorithm was intuitive for the user. During implementation and operation, the user must be comfortable with the process used to analyze the image, in order to trust the results.

The user interface was divided into three parts to correspond to the three modules of the algorithm: configuration, training, and online operation. The configuration stage allows the designer to indicate which features are relevant; the training stage develops the neuro-fuzzy system that will be used for decision making; and the online operation module makes decisions on a stream of real-time images based on the neuro-fuzzy output.

A complete description of all of the functions available in the user interface is included in Appendix B, the QVision User's Guide.

4.1 Configuration Window

The configuration window shown in Figure 4.1 allows the user to define the problem.



Job Summary
Features: 1: Line A

IMG #3...y Documents\AUTO21 Project\Algorithm Essentials_May10\Examples\LPass\LPass003.bmp

Image Sets
Pass Set 50 'bmp's
Fail Set 50 'bmp's

Feature Definition - Line
Fixture: None
X: 240.0
Y: 160.0
Width: 50.0
Height: 100.0
Nom. Angle: 0.0
Angle +/-: 20.0

Feature Attributes
Theta -2.5000
Rho 250.8038
N/A
Score 79.4299

Figure 4.1: The Configuration Window gives access to functions for setting up the inspection problem and loading example images.

The first step is to load pass and fail images using File -> Open Pass/Fail Set. Pass and fail images are loaded by selecting one directory of images for each. The directory will be scanned for all image files and an index will be created so the program can reference any of the images in the directory later on.

Once the images are loaded, the user must decide on which features to use. Adding a large number of features to the specification is permitted as they can be reduced later on by the optimization process. A toolbar gives quick access to the four major feature types: lines, circles, holes, and colour patches.

Defining a feature involves specifying the search region using numerical values in the sidebar or by interactive selection. The extraction parameters (such as minimum and maximum radius, colour type etc.) are then set. Each type of feature has different specifications that must be set for proper extraction.

The feature attributes used for each feature can also be restricted by checking or un-checking the appropriate boxes in the *attributes* section of the side bar.

As each feature is defined, the search region is shown on the current image along with the extracted feature for the current image. The attributes section shows the attribute value for the selected feature in the current window.

Once all of the desired features have been defined, a number of other tools provide assistance in checking and/or improving the setup of the problem.

4.1.1 Statistics Tool

The statistics tool (see Figure 4.2) displays a plot of the feature attribute values for all of the images in the dataset. This is a good way to quickly identify images that may be unusual or are not extracting the correct feature. It also gives some insight into the nature of the data. Well segregated data may be able to be processed using a simple threshold technique whereas overlapped data will need a neuro-fuzzy approach. Feature attributes which show no clustering at all (such as Line A: T in Figure 4.2) are probably poor indicators and perhaps an alternative feature should be used.

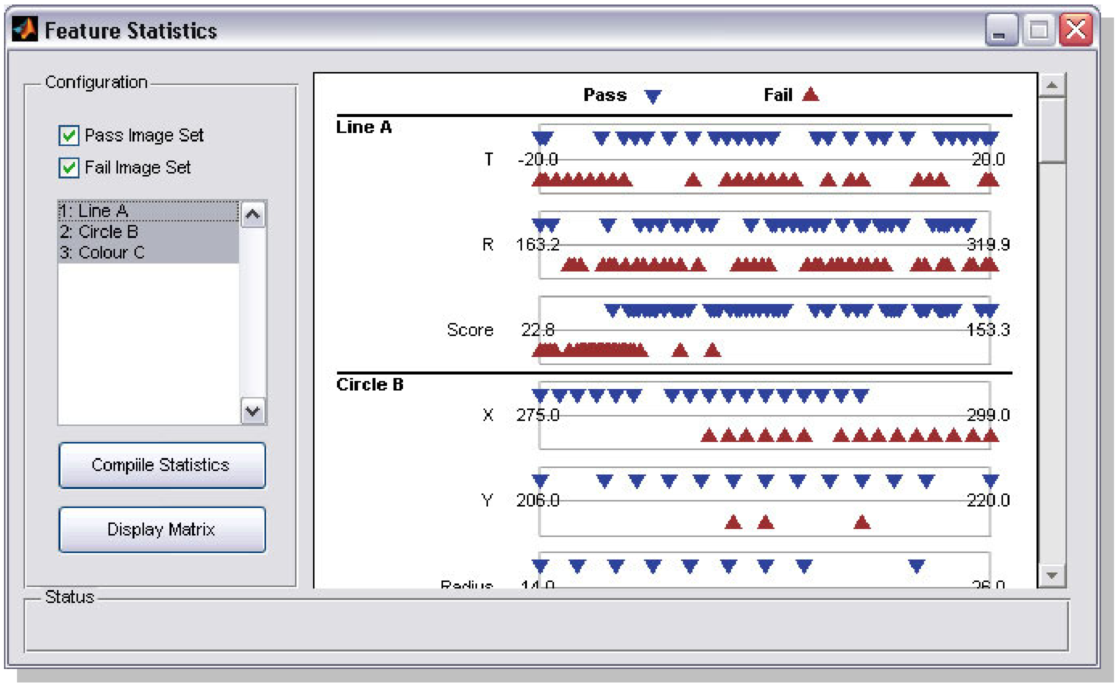


Figure 4.2: The Feature Statistics screen shows a plot of the feature attribute values separated into pass (inverted triangles) and fail (upright triangles). A strong left/right separation of the two data sets (for example the Line A: Score values) indicates a feature attribute that differentiates well between pass and fail images.

4.1.2 Tracking Tool

It is often difficult to decide how big the search region should be for a particular feature in order to cover the variation in position of that feature through the entire image set. The tracking tool automatically goes through the image set and attempts to determine the best search window.

As shown in Figure 4.3, the algorithm operates by first using the SIFT method developed by Lowe (2004) to locate reference keys in the current image. Likely matches are then identified in the other images and a mapping is created for the search region. The new search area is taken as the union of the original and the transformed region. Repeating this process for all the images in the data set gives a search region that should encompass all locations of the feature of interest. In order to avoid over-expanding the region, the original image is checked at each step in order to ensure the same feature is still being found.

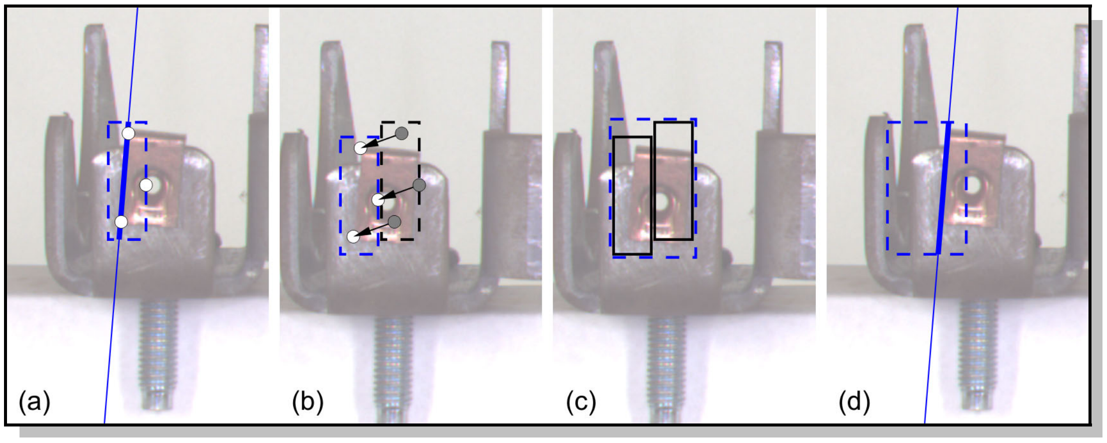
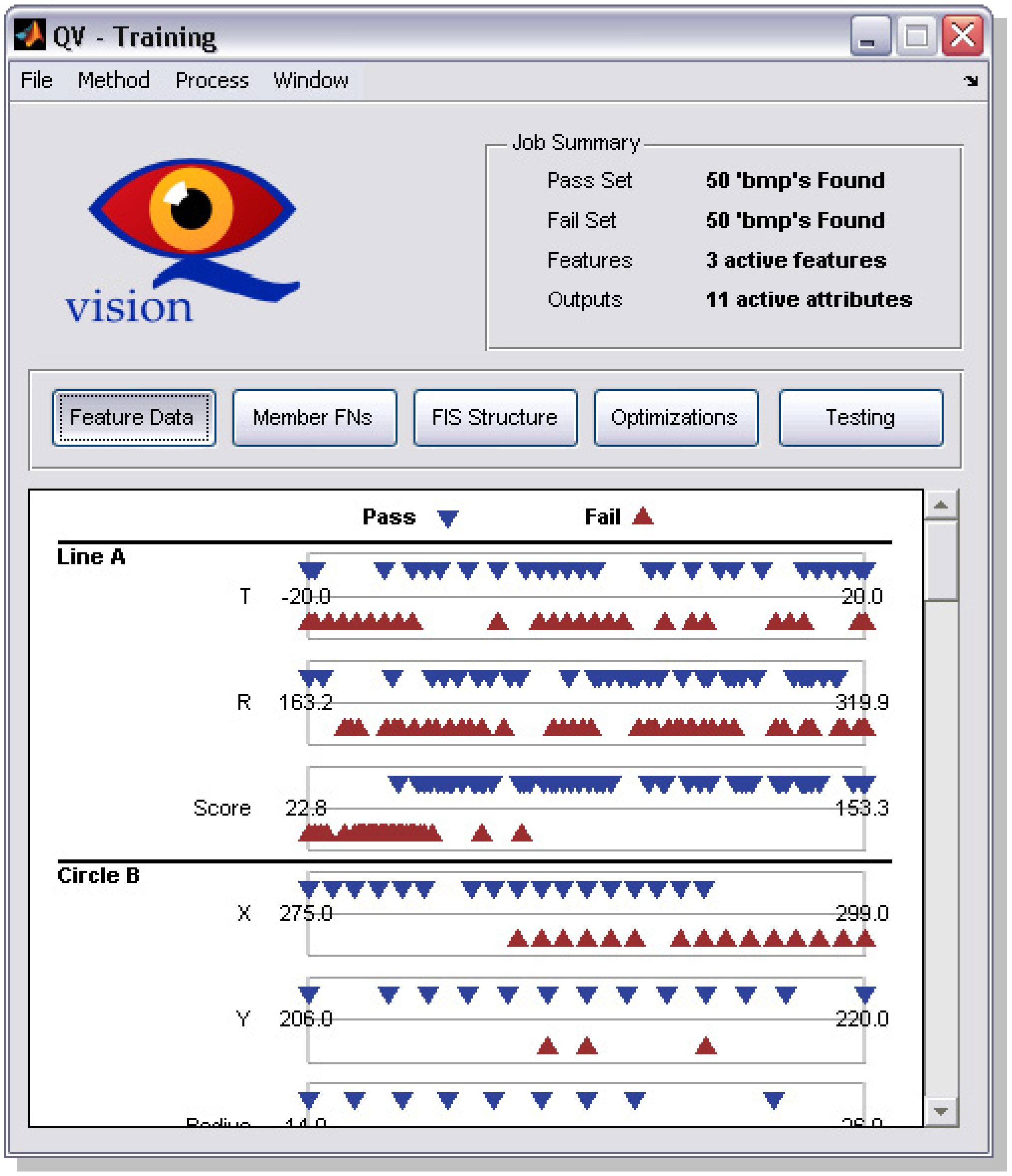


Figure 4.3: The Tracking tool first locates SIFT keypoints in the source image (a), then locates matching keypoints in the second image and determines a mapping (b). The search region is then transformed by the same mapping, and merged with the original region (c). The final result is checked by extracting the feature from the original image.

4.2 Training Window

Training is the core of the algorithm, but requires the least input from the user. Once the problem is defined in the configuration window, the problem definition and image sets can be directly transferred to the training window. The training window (see Figure 4.4) guides the user through the process of training the network.



Job Summary

Pass Set	50 'bmp's Found
Fail Set	50 'bmp's Found
Features	3 active features
Outputs	11 active attributes

Figure 4.4: The training window gives access to tools for the development of the classification system.

The feature values are first extracted for each image in the set using the *Process -> Extract Data* menu option. The extracted data is again displayed so that the data distribution can be verified.

Process -> Divide Data Set is used to randomly sort the dataset and divide it into training, checking, and testing datasets. The data can now be used to train the classification method. After training is complete, the membership functions will be shown and the final Root-Mean-Square (RMS) error reported. With the neuro-fuzzy method, the network structure is also available by selecting the Fuzzy Inference System (FIS) Structure tab (see Figure 4.5).

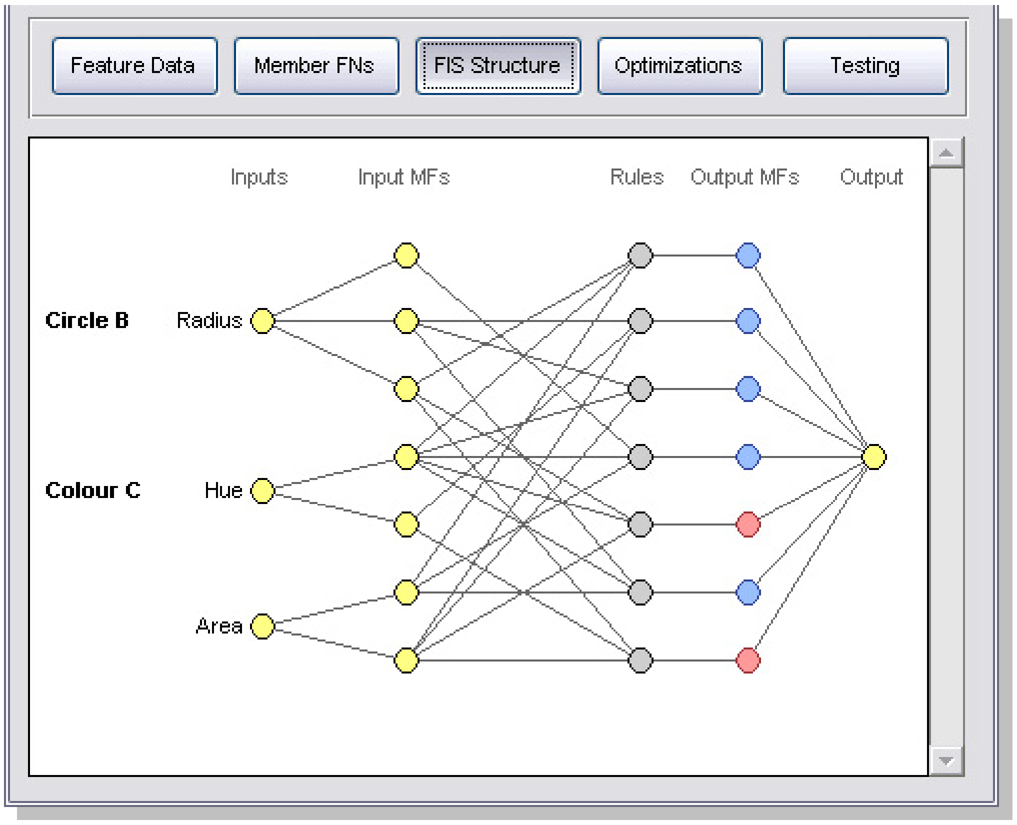
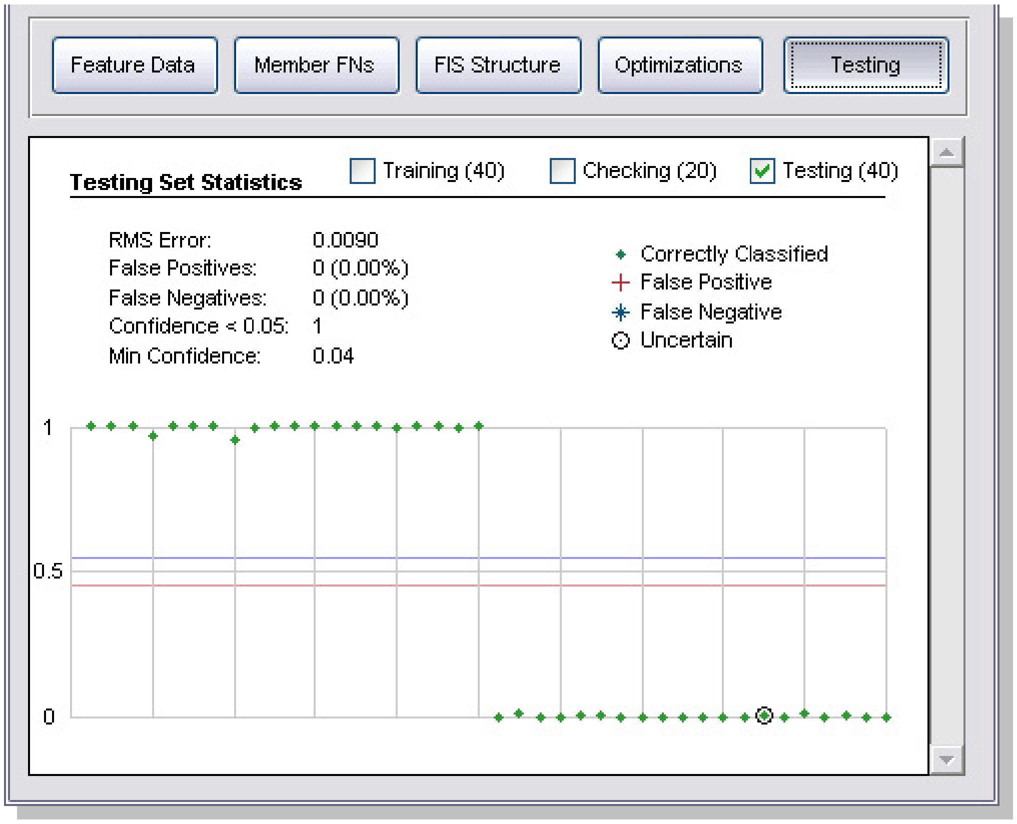


Figure 4.5: The FIS Structure part of the Training Screen shows the network connections between inputs, membership functions, rules, and outputs.

The structure that is formed directly after training is usually complex due to the high number of feature attributes included in the classification process. Optimization reduces the number of feature attributes and can also improve performance. Selecting *Process -> Optimize Solution* begins the optimization process. The results of the optimization are shown on the *Optimizations* tab, listed in order of decreasing RMS error. Selecting one of the rows will reconfigure the network to use only those feature attributes indicated.

A lower RMS error does not necessarily mean better performance since the goal is to correctly classify images with a high degree of confidence. The Testing tab shows the performance visually with low confidence decisions circled (see Figure 4.6). The plot is for the entire dataset (training, checking, and testing data) and relevant statistics are displayed.



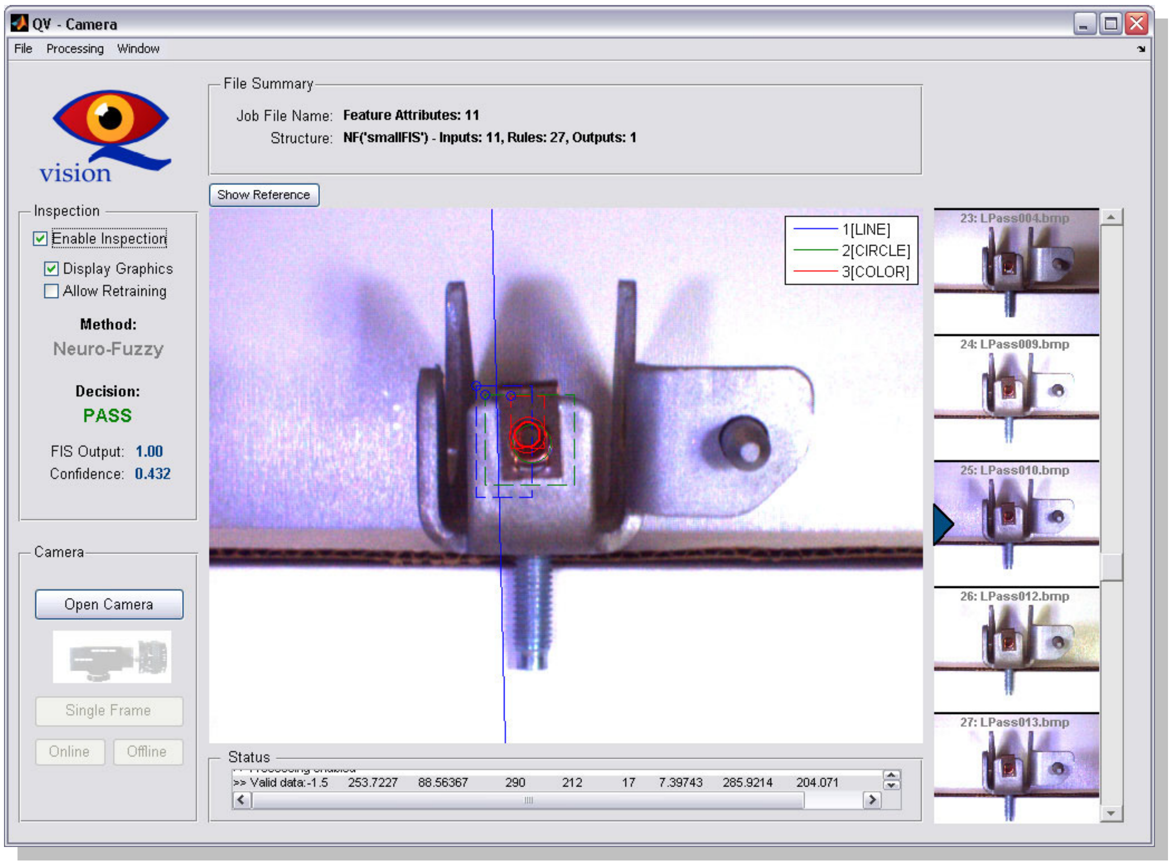
Testing Set Statistics

RMS Error:	0.0090
False Positives:	0 (0.00%)
False Negatives:	0 (0.00%)
Confidence < 0.05:	1
Min Confidence:	0.04

Figure 4.6: The Testing tab of the Training Screen. The plotted points indicate the output value from the classifier for each image. A value of 1 is a pass image and 0 is a fail images. If the decision is uncertain, the point is circled to indicate an unusual image.

4.3 Real Time Operation

Once the network is developed and trained, the Camera window (see Figure 4.7) is the screen that the operator would see on a daily basis. It shows the current image from the camera, the features that are extracted, and the classification made by the network.



File Summary
Job File Name: **Feature Attributes: 11**
Structure: **NF('smallFIS') - Inputs: 11, Rules: 27, Outputs: 1**

Inspection
[✓] Enable Inspection
[✓] Display Graphics
[ ] Allow Retraining

Method: Neuro-Fuzzy

Decision: PASS

FIS Output: 1.00
Confidence: 0.432

Camera

Status
```
>> Valid data:-1.5   253.7227   88.56367   290   212   17   7.39743   285.9214   204.071
```

Figure 4.7: The Camera window shows live images from the camera and the classifier output for the current image. In this case the image has been identified as a Pass with a confidence of 0.43. Stored images can also be loaded (right-hand column of images).

For each new image, the right-hand bar shows the FIS output, classification of the image, and the confidence value (see Section 3.6.1 for details on how the confidence value is calculated). If the confidence falls below a preset threshold, then the classification will change to Uncertain, indicating that the network needs to be adjusted to handle the new data. The threshold at which an image becomes uncertain was determined experimentally in order to catch unusual images and could be adjusted to give more conservative results.

Checking the Allow Retraining box will present the operator with a dialog box for images with an uncertain classification. The operator can then manually classify the image and the system will attempt to update the network to reflect the new information. If no changes are possible, the operator will be notified to retrain using new example images.

For the purposes of off-line testing, image sets may also be loaded from disk and run as if they were coming from the camera.

4.4 Summary

This chapter outlines the user interface which gives access to the features of the algorithm. The interface consists of three main parts which are used for: set up of the problem, development of the classification system, and online testing. The graphical displays allow the user to interact with the system in an intuitive way. By using a modular approach, changes can easily be made to the operation of the algorithm without changing the way the user interacts with the software.

Chapter 5

Experimental Method

Testing of the neuro-fuzzy algorithm is based on a series of images captured from a test cell in the Intelligent Automation Laboratory at Queen's University, and two sets of images from the Van Rob production facility. The neuro-fuzzy system is compared to a threshold-based system. The purpose of these tests is primarily to investigate the performance of the algorithm in processing data to arrive at the correct outcome.

5.1 Hardware Design

The real production environment has a number of factors that influence the image captured by an AVI system. In automotive production lines where welding, grinding, and other metalworking processes are common, there is more chance that lighting conditions will vary, dust will accumulate on sensor and lights, and vibration will be transferred to the inspection area. In addition, shops with natural lighting (which are becoming more common in an attempt to improve the employee environment (NREL, 2002) make it difficult for AVI systems, since they work best under controlled lighting conditions.

In the image set from Van Rob, all of these factors are acting simultaneously on the image. To test the algorithm in a more controlled environment, a test cell was constructed which could replicate and isolate some of the conditions in the real inspection environment. An additional smaller cell was also constructed for imaging smaller parts and for demonstration purposes.

5.1.1 Overall design

The part that was available from Van Rob is quite large, measuring about 2m in length. It was decided that making a test cell to fit the entire beam was not practical. Instead, the beam was cut in half since the camera on the real system only sees a small portion of the beam at a time.

The overall cell dimensions were set by the size of the part, the focal length of the camera, the effective throw of the lights, and the space required for the lighting to diffuse and not give localized hotspots on the part. Final dimensions are 1.2m in width, 1.5m in length, and 0.9m in height. The cell frame is extruded aluminium with hardboard panels providing the side walls. A sliding door at the front of the unit allows the operator to access the camera, lighting, and part when needed, but to keep out ambient light when capturing images (see Figure 5.1). The top panel is translucent plastic so that external lighting can be allowed into the cell. This can be blocked out by an opaque black cloth when required.

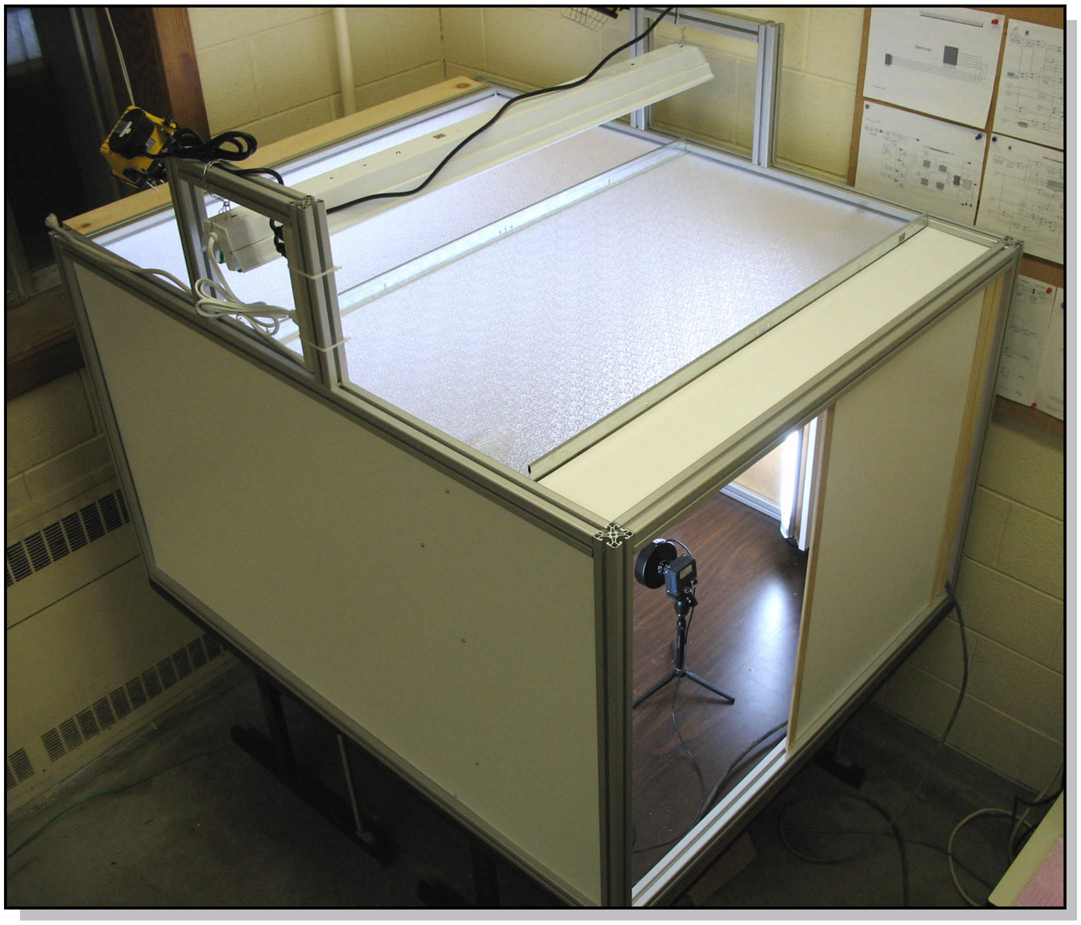


Figure 5.1: The exterior of the laboratory test cell showing sliding door at the front. Exterior lighting is visible above the top panel.

5.1.2 Lighting

One of the biggest issues first identified by Van Rob was lighting. The original inspection station at Van Rob had an open top. The overhead lighting in the factory was low frequency which means that the flicker from the lighting was longer than the time to capture one image frame in the camera. With an open top to the cell, the camera would occasionally capture the dark portion of the flicker which would change the light levels and interfere with the processing.

The cell was also located near a loading door. In the warmer months, the door was often opened to allow fresh air into the building. However, this also influenced the ambient light level in the inspection cell and again caused problems.

In order to replicate these effects, the laboratory test cell is constructed with a translucent panel in the top (see Figure 5.2). Above the panel, there is a low frequency fluorescent light to provide a flickering light source. Two tungsten-halogen lights are also mounted by the fluorescent fixture and can be used to shift the colour of the incoming light to a more red hue. The cell is located near a large outdoor window in the building which can be used to allow sunlight into the cell. Tests with only interior lighting can also be performed by covering the translucent panel with a black cloth that blocks exterior light sources.

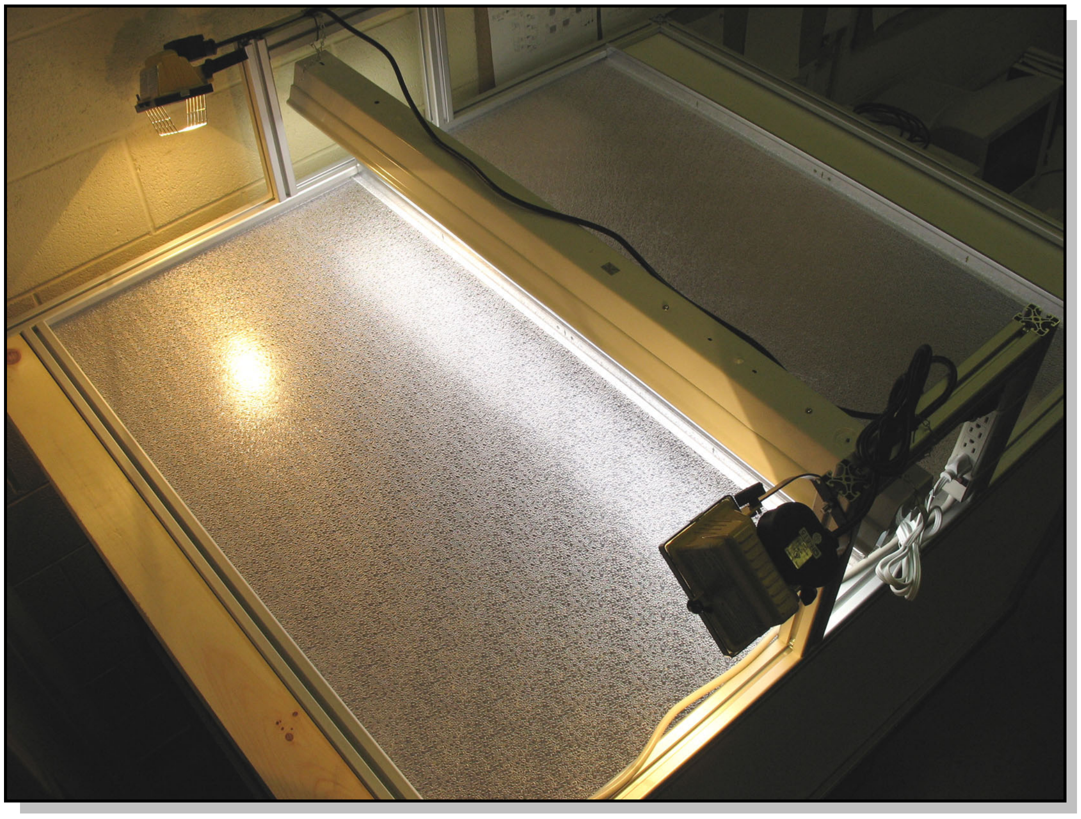


Figure 5.2: Exterior lighting on the test cell consists of two incandescent halogen sources, and one 4 foot fluorescent source.

The interior lighting consists of two of the most popular AVI lighting sources: high frequency fluorescent and LED.

High frequency fluorescent lights run at several thousand Hertz which means that during one image frame, the light level will change from light to dark many times. The sensor in the camera will average out these flickers and the result is consistent light from frame to frame. Fluorescent lighting provides good general lighting over a broad area. As shown in Figure 5.3, two of these fixtures are located approximately 45 degrees from centre, to either side of the camera. The location of the fixtures is designed to reduce the amount of direct glare from the flat surfaces of the parts.

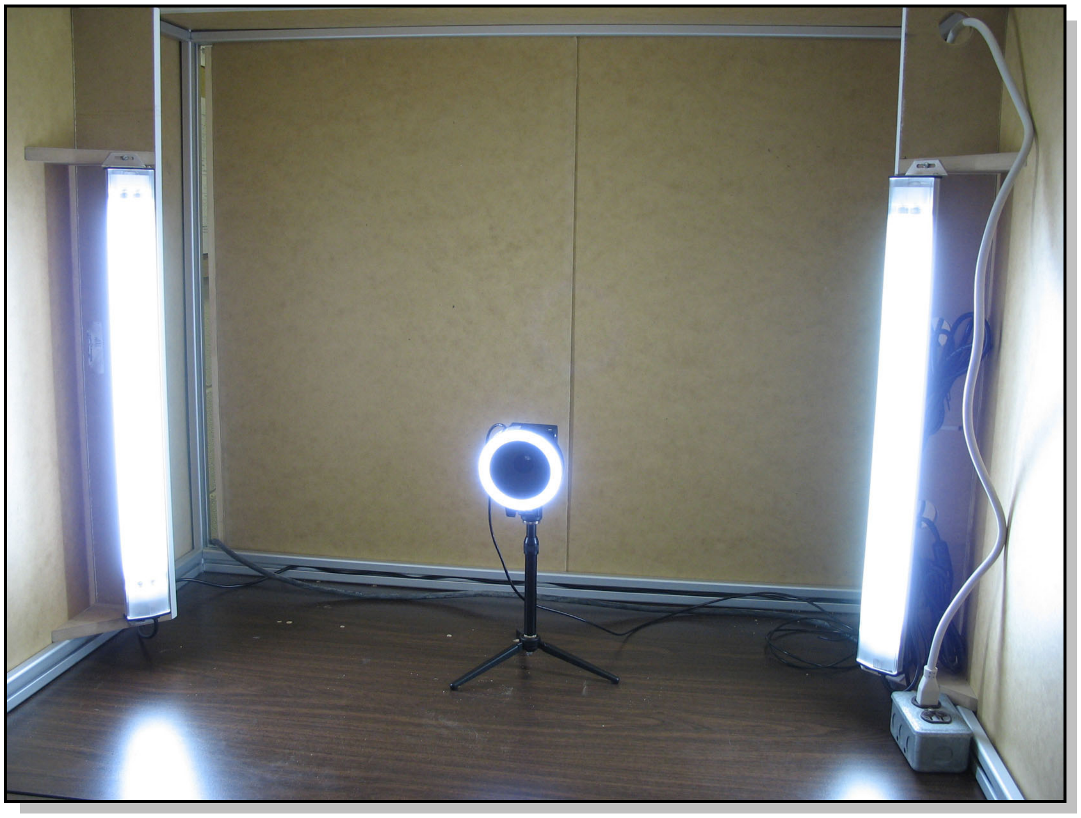


Figure 5.3: Interior lighting in the test cell consists of two 2 foot long, high frequency fluorescent lights, and one LED ring light mounted to the camera.

The LED fixture consists of an LED ring-light. The ring-light attaches directly to the front of the camera lens and provides an even light directly from the viewpoint of the camera. This direction of lighting often has problems when imaging flat metallic parts due to direct reflections, but can sometimes provide helpful highlights and shadows. LED lights are driven by direct current and so do not flicker. They also tend to provide more a more consistent spectrum of light over the life of the fixture (Zuech, 2003).

By using different combinations of lighting, the appearance of the image can be changed quite drastically as shown in Figure 5.4. This is much larger than the range of lighting that would be experienced in an actual inspection station.

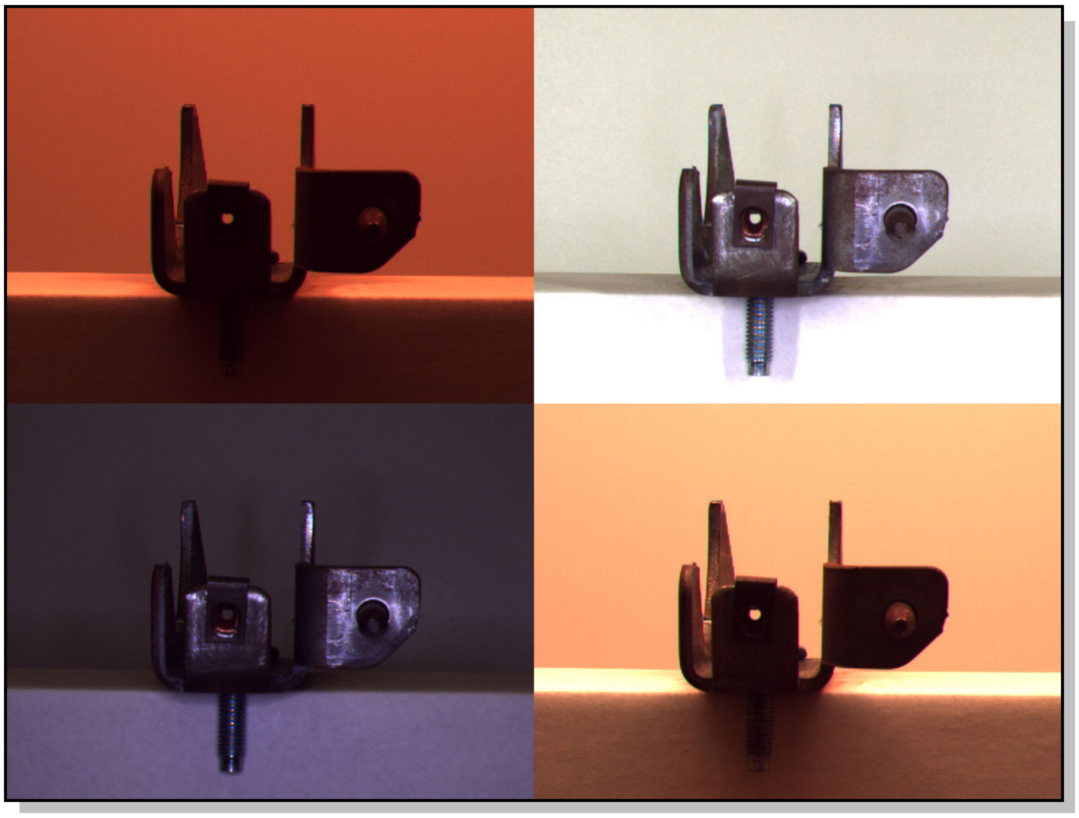


Figure 5.4: Four examples of the range of lighting that is possible in the test cell. Clockwise from top left: external incandescent, all internal lights, external fluorescent and incandescent, LED.

5.1.3 Camera

Although it was desirable to duplicate the Van Rob system as much as possible, the selected camera is not the same camera. The main reason for this is that the system at Van Rob uses a Cognex® In-Sight™ 5400C smart camera (see Appendix C for specifications of all cameras used for this project). Smart cameras contain onboard processing devices that run standard image processing algorithms. The user can set up the camera to perform these processing steps on various parts of the picture and also determine how the resulting information will be processed to make a decision about the image. This is handy for industrial settings as there is no need for an external computer to run the image processing software. However, it is challenging to work with this system for research and development.

For the laboratory-based system, a colour camera was selected since the Cognex® camera was also colour and colour information can often be useful to identification of parts during inspection. A Basler® A312fc was selected as it gave similar performance to the Cognex® camera in terms of resolution and speed but was able to connect to a PC using an IEEE-1394 (FireWire) connection. The images are read into the MATLAB® environment using the Image Acquisition Toolbox and the Carnegie Mellon University IEEE1394 driver.

For the smaller test cell, an imi tech IMC-1080FT camera was purchased. This camera is slightly higher resolution than the Basler® camera, but is also FireWire based.

5.1.4 Background and Environment

The cross-car beam from Van Rob contains many holes and is a very open structure, so it is possible to see through the beam to other parts of the cell and the factory itself. This means there can be significant variation in the images due to the position of other machinery. To simulate background motion in the laboratory test cell, a large space was allowed behind the part and the back wall such that clutter objects could be added or removed. The back wall itself is normally white but can be easily removed to change it for a different surface.

Although dust build-up, smoke, and other particulate can be a fairly major influence on the image quality over time, no attempt was made to specifically model these effects in the laboratory test cell. It was decided that a system that is able to deal with changes in lighting conditions, would also be able to handle any intensity changes due to gradual dust build-up or environmental effects.

5.2 Test set

The laboratory-generated test images are views of a part of the cross-car beam under investigation. The part consists of a single stamped bracket with one J-type clip as shown in Figure 5.5.

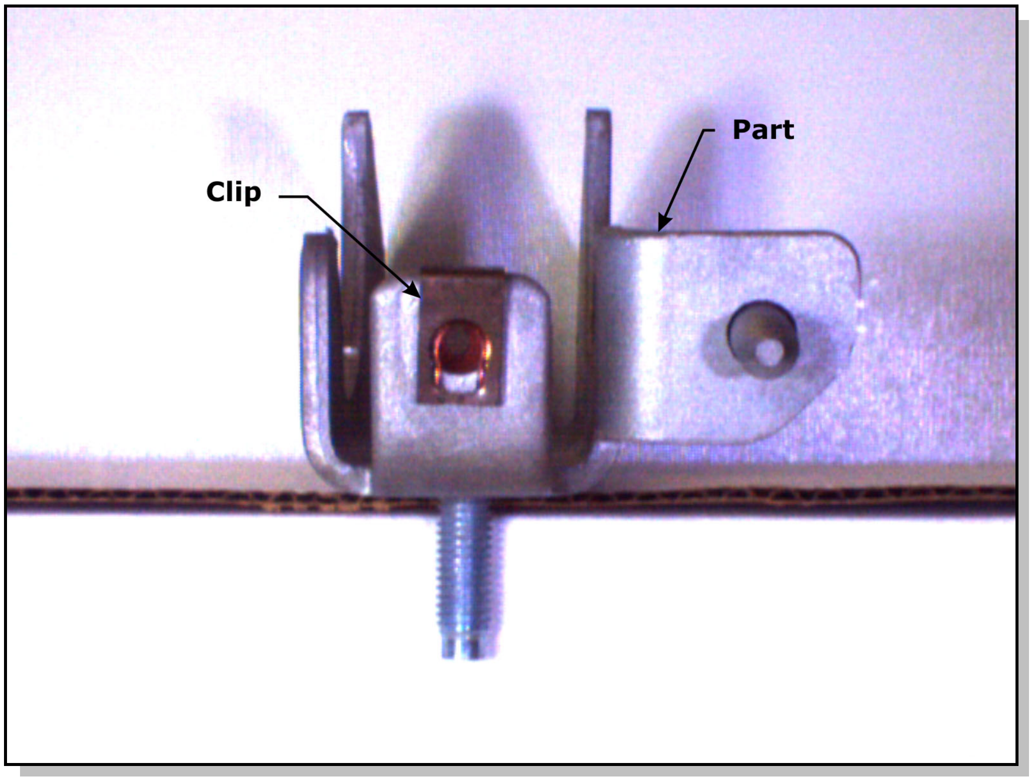


Figure 5.5: An example of one of the images used for training and testing the classification system.

The image set represents the types of lighting, part, and environmental variations that occur in the production environment. Variations in the images include exterior lighting, small changes to the position of the part and clip, changes in the background position and colour, and the addition of clutter objects. Interior illumination and the camera settings (gain, white balance, shutter speed, and iris) are the same for all images.

A total of 50 pass and 50 fail images were generated for testing. The pass and fail images have similar conditions, but in the fail images the clip is removed. The objective is to determine if the clip is present.

For a more realistic idea of the variation that can be expected in real images, the image database from Van Rob provides over 3,000 images of the real part. Two databases are available from two different periods in the system's development (see Figure 5.6). The first contains only a single view which is used to inspect two clips. The second set contains three different views ("View 2", "View 8", and "View 9") with multiple clips per view. The older dataset is actually easier to classify as the intensity of the lighting in the new set means much of the colour information is lost. A detailed description of the datasets is given in Appendix D.

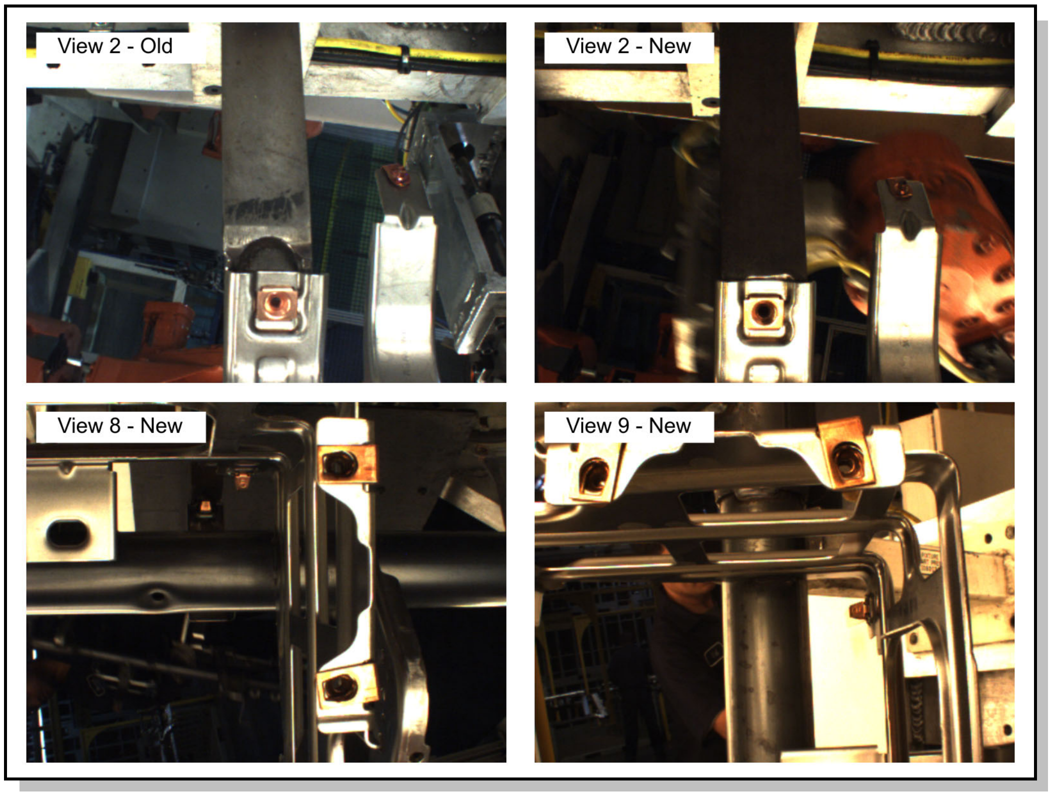


Figure 5.6: Sample of the images from the Van Rob image set. The first image is from the early development of the system and has more colour information. The latter three images are from the newer images and provide additional views.

Both of the image sets from Van Rob are biased toward passed parts. This is because the image sets were generated during production runs and so failed parts were only captured at their natural rate of occurrence. For most tests of the algorithm, an equal number of pass and fail images are selected from the overall dataset in order to provide balanced training.

5.3 Testing of Feature Extractors

Although feature extraction methods are not the focus of this study, the custom feature extractors need to be tested to ensure they produce reliable data. In addition, it is desirable to confirm that feature-based methods help to reduce the influence of lighting conditions on the extracted feature.

The initial tests of the feature extractors are based on a simple test pattern target which includes linear edges and circles (see Figure 5.7). This target is placed in the test cell, and used during the development of the feature extraction algorithms. As it is very simple, it allows quick verification that the algorithm is operating correctly without the influence of other factors.

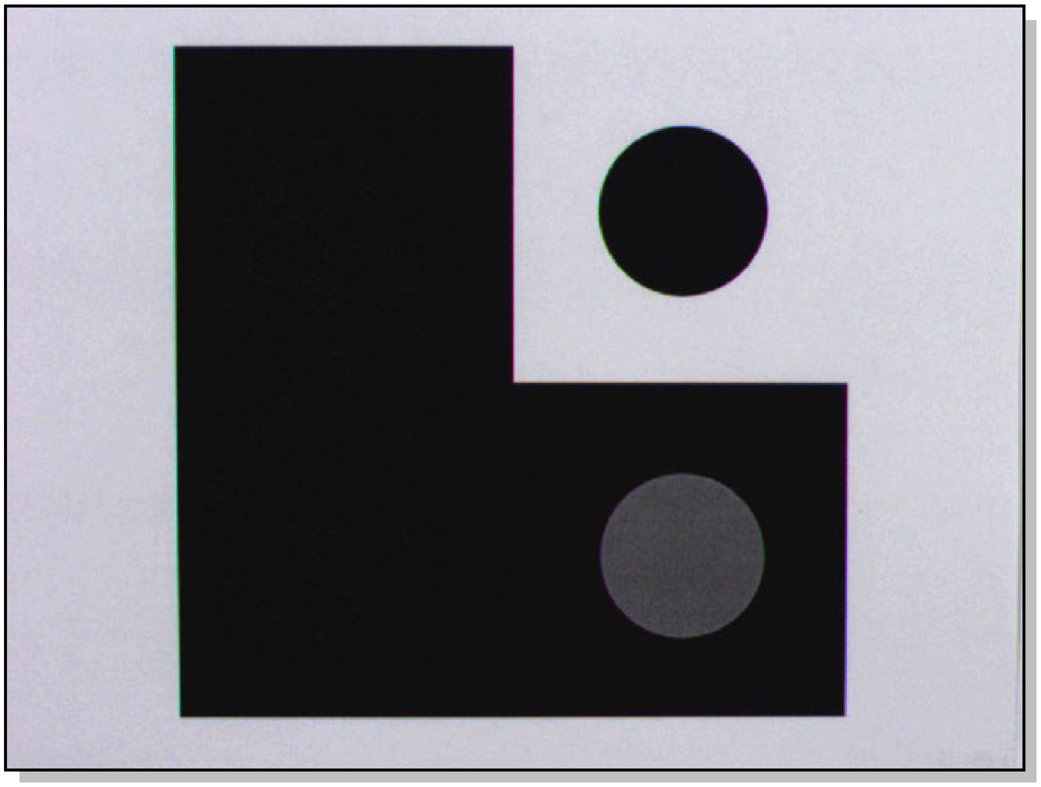


Figure 5.7: Testing target used for early development of the algorithm. Linear edges and round circles allowed easy verification of the feature extraction results.

Such a simple target does not really test the robustness of the extractor under different lighting conditions and lower contrast scenarios. Further tests focus on more realistic images by using the real part in the test cell.

The real part contains several lines, circles, and colour regions. For the test, one of each type of feature was selected as shown in Figure 5.8.

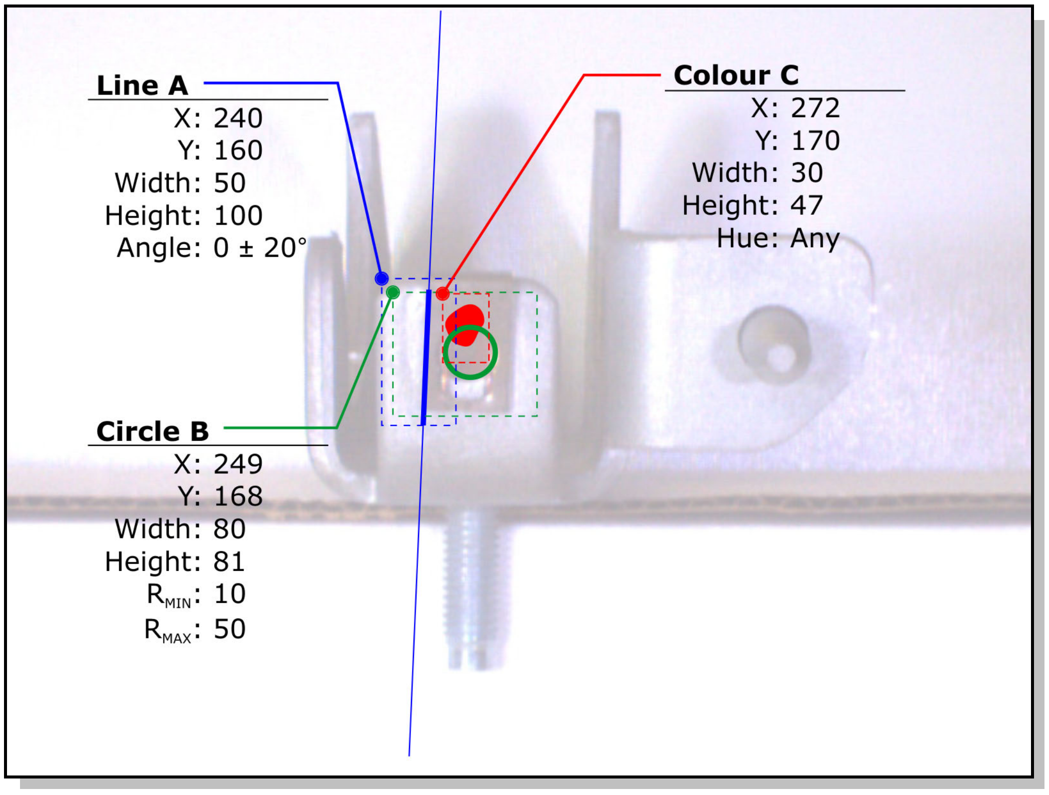


Figure 5.8: The features used for testing with the laboratory-generated images.

By looking at how these features are extracted over a sample of input images, it is possible to see that the correct object in the image is detected even though there are large changes in overall lighting intensity and direction. For comparison, similar features were defined in the commercial Cognex® Insight Explorer system. For the same sample of images, the trend in output values are generally the same, but the Cognex® system is able to return more stable values. This was expected as commercial image extraction systems have been fine-tuned over many years to perform extraction as consistently as possible. However, having less stable feature extractors means the classification system needs to be better and so provides a more challenging test for the neuro-fuzzy system.

5.4 Testing of Auto-Search method

The idea of the auto-search algorithm is for the user to be able to define a search region in one image without worrying about where that feature might be positioned in subsequent images. The auto-search algorithm attempts to locate the feature in the other images and expands the search region accordingly. Successful search region expansion yields a region that finds the desired feature in all images.

To test the auto-search algorithm, a line feature is used. Specifically, the left side of the clip was selected. This edge is fairly difficult to track as it both moves laterally due to the position of the part, and also rotates depending on how the clip was inserted. The initial search region is defined using the first pass image (LPass001.bmp) as shown in Figure 5.9. The initial permitted angle range was the default of 0 ± 5 degrees. At the end of the expansion process, the search region has been expanded to a larger area and has an angle range of -4.9 ± 18.4 degrees.

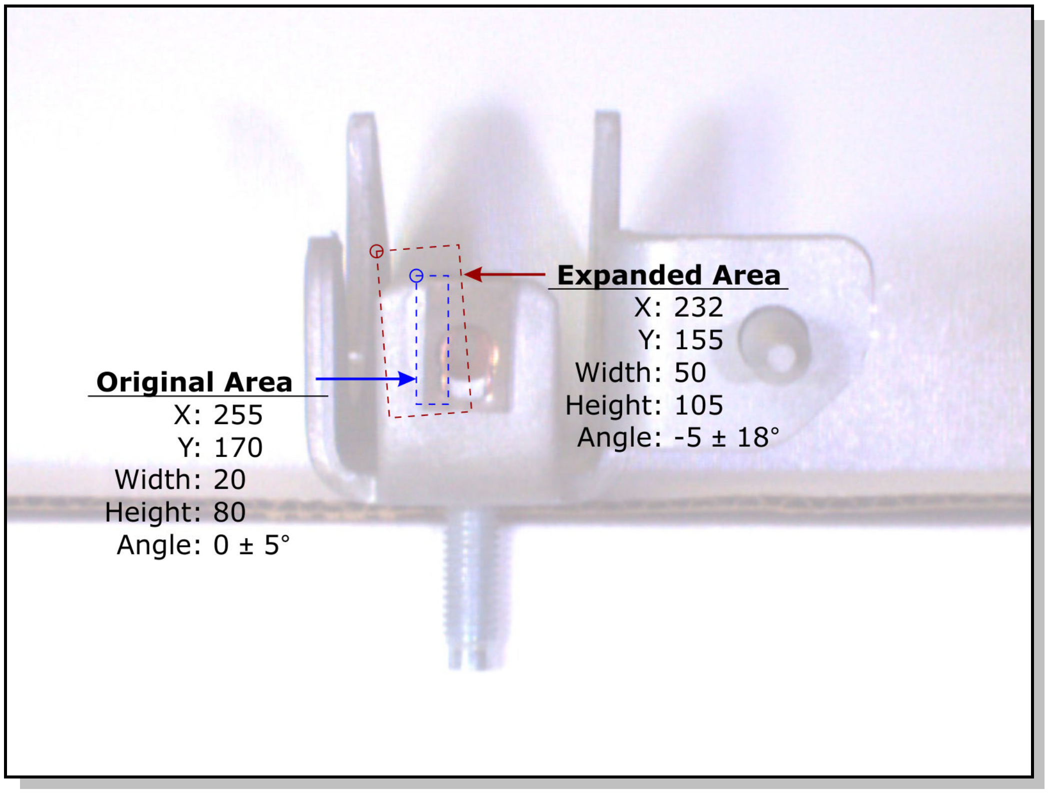


Figure 5.9: The original and automatically expanded regions for the line feature on the edge of the J-clip.

The results of using the expanded region to extract lines are shown on three images in Figure 5.10. The left image shows the correct extraction on an image similar to the reference image. This shows that the algorithm is able to maintain the original intent. The second image shows a rotated clip that would have been outside the original search region, but is now correctly identified. In the third image, the feature is not correctly identified because the expanded search region also includes part of the background which has a strong edge to over-ride the true clip edge.

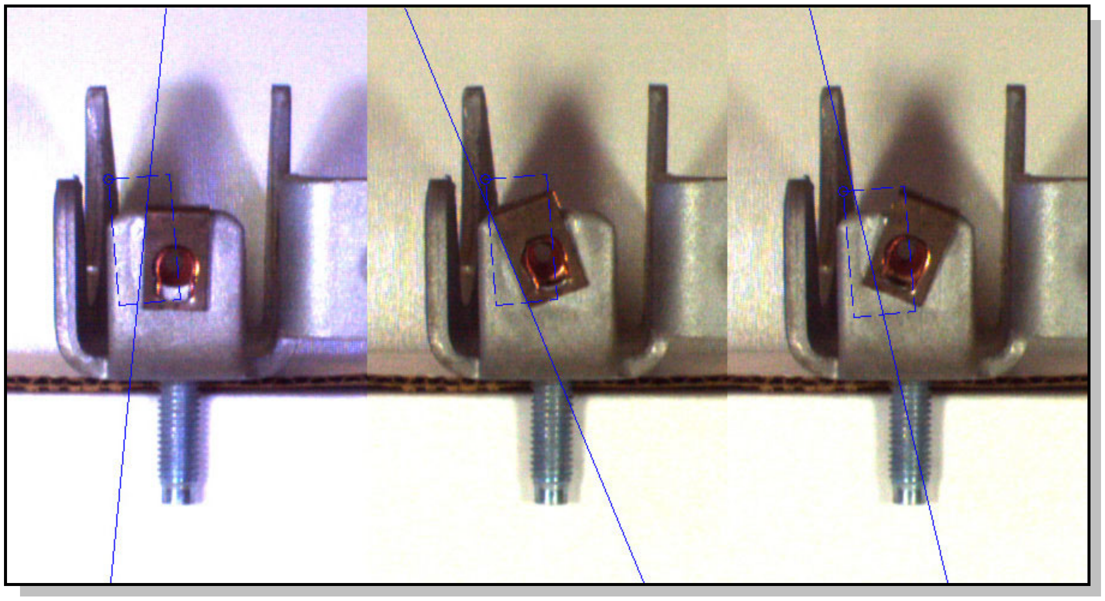


Figure 5.10: Results from extracting features using the auto-expanded search region on three images from the training set. In the left two images, the clip edge is correctly identified, but in the right image, an edge the background causes incorrect extraction.

By manual adjusting the size of the search region so that it no longer includes the background, all of the images can be processed correctly. Since this solution is not much different from the parameters of the automatically expanded region, slight adjustments to the auto-search method may solve the problem.

5.5 Development of Baseline System

For the testing of the neuro-fuzzy algorithm performance, it is necessary to provide a baseline system for comparison. Since the neuro-fuzzy algorithm uses feature data derived from custom feature extractors, it would not be consistent to directly compare it to commercial systems.

Most existing systems use some form of threshold-based decision making. That is, the user is required to set hard limits for the allowable values of various feature attributes. If there are multiple features or attributes, then the operator is also required to decide how the information should be combined to arrive at a final decision.

In dimension checking applications, this process is relatively straight forward. Each feature attribute must pass within the required tolerance. If any of the feature attributes fail, the whole part fails. However, in assembly type inspection, each feature attribute is used as evidence to support the overall decision of correctly or incorrectly assembled.

One of the other problems with threshold-based systems is a lack of a systematic method for determining the threshold values. Usually it is up to the operator to decide on the value to use. In this case, consistency is ensured by using an algorithm to set the threshold values.

The threshold-setting algorithm looks for clusters of points in the training data. Two input parameters determine how large the clusters must be before they are accepted, both in number of points, and in range compared to the total span. The algorithm proceeds by taking a single run through the data, collecting groups of adjacent pass or fail points. Only those groups which are large enough (based on the input parameters) are stored as regions.

When region generation is complete, the regions are separated by threshold points. Each feature attribute can have multiple thresholds if they are supported by the data. This is slightly more advanced than most manually configured systems as typically only one threshold will be used per feature attribute. The locations of the thresholds set by the algorithm generally fit well with the locations selected by a human user. Some examples are shown in Figure 5.11.

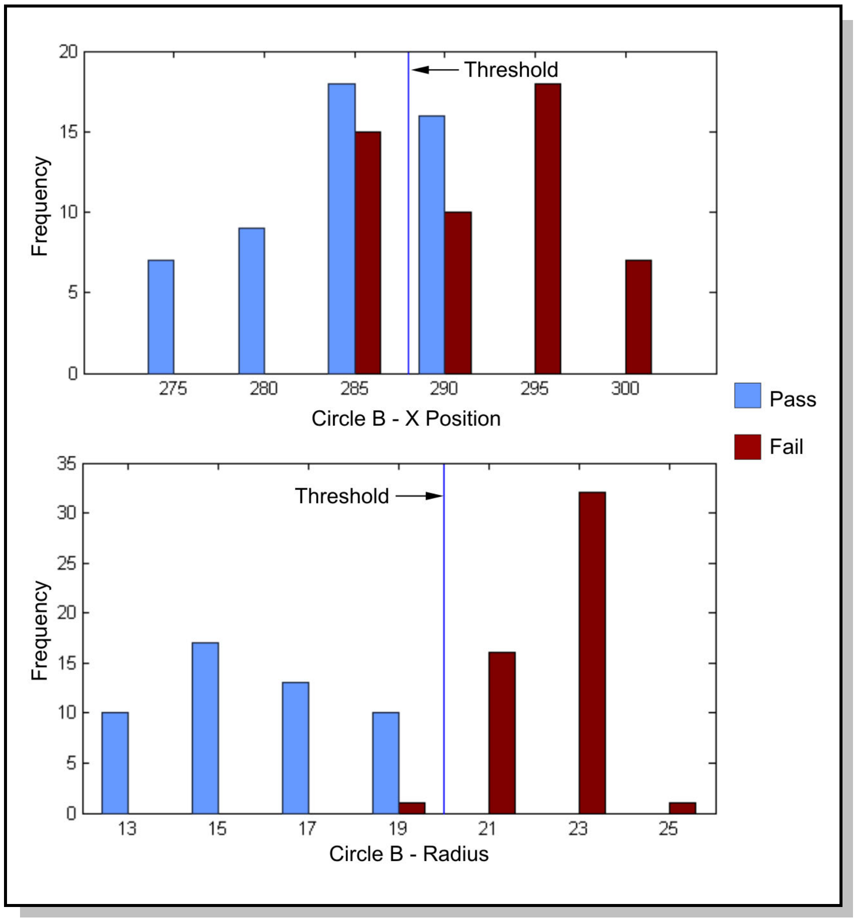


Figure 5.11: Two examples of the calculated threshold position in relation to histograms of the pass (right) and fail (left) feature attribute data. The lower diagram shows a case where the threshold can separate the data fairly well. However, in the upper diagram, many images will be misclassified due to the overlap of the pass and fail data.

To evaluate new data points, they are compared to the defined regions. If a data point falls within a region designated as pass, it is assigned a classification of one, otherwise it is given a classification of zero. As discussed above, in inspection of assembly, each feature attribute can contribute to the overall decision. To achieve this effect, a novel approach was developed whereby the outputs from each feature attribute are averaged to produce a single output value. This makes the output a continuous variable between zero and one with values closer to one indicating more evidence in the image for a pass classification.

In a more advanced system, each attribute could also be given a different weight in the average depending on its importance. However, this is outside the normal procedure in traditional systems and starts to approximate some of the features of the neuro-fuzzy system.

An optimization method is also implemented for the threshold system which is similar to the neuro-fuzzy optimization process (see Section 3.4). Since the output of the threshold classifier is the average of the outputs for each feature attribute, optimization is simply a matter of determining the feature attributes that should contribute to the average to achieve the lowest overall classification error. The average can be calculated very quickly so an exhaustive search method is feasible.

5.6 Testing Results and Discussion

5.6.1 Ease of Operation

Improvement of time it takes an operator to learn and set up the system was one of the major goals of this project. However, this is difficult to quantify as it would require an operator who was not experienced with any vision system to set up the QVision software and a traditional commercial package. Even then, things learned about the images on the first system would reduce the time required for the second system.

Therefore, explicit testing of the set up time was not done. What follows is an examination of the steps required to configure the system and how these might increase or reduce the total time required.

Although the threshold-based system described in the previous section provides a good benchmark for accuracy performance tests, it is much more automated than traditional systems and so does not give a good comparison for training time.

In both the QVision system, and with traditional methods, the user must define a set of features that will be used for inspection. This is how the system is told what is important to the inspection process. Since the QVision system uses a graphical interface to allow the user to draw the search regions, this part of the process should take the same amount of time as a commercial graphical system.

Once the features are defined however, the QVision system allows the user to extract the feature attribute data for all of the images in the training set in order to observe the range and overlap of the data. This makes it much easier to find any outlier images. These are images that have unusual conditions which cause features to be extracted improperly.

Perhaps what takes the longest time in setting up a traditional system is defining how the feature attribute values will be combined to create the decision output. This is where the example-based training is truly superior. By presenting images that contain the expected variation in part appearance and position, the algorithm can determine the range of values to allow or disallow. Through the optimization process, the algorithm also eliminates any features that are not required for inspection.

Traditionally this is a trial-and-error process that can take weeks or months for the user to reach the right combination. With the QVision system, the user can generate a result within a few minutes. This result can then be evaluated and new features added if performance is not satisfactory.

Small changes to the workflow can also make a big difference. The QVision system allows the user to view the performance of the system on example images through an intuitive plot showing performance on a number of indices. Traditionally, this had to be done be looking at each image individually and seeing if the correct result was achieved.

5.6.2 Performance Measures

Testing of performance is based on four indices. The *RMS error* is the root-mean-square of the error between the classifier output and the desired classification as calculated by:

$$RMS\ Error = \sqrt{\frac{\sum_{i=1}^{n}(c_i - x_i)^2}{n}}$$

Where x_i is the classifier output for each image, c_i is corresponding correct classification (zero or one), and n is the number of images (Mehrotra et al., 2000).

Number of *false positives* is the number of images with clips installed incorrectly which are classified as passed. Number of *false negatives* is the number of images with clips installed properly which are classified as faulty. Finally, the number of *uncertain images* is the number of images where the classifier output is either weak (no fuzzy rules satisfied strongly) or close to the decision threshold of 0.5.

5.6.3 Lighting Change Robustness

The first test for lighting change robustness uses the images generated from the laboratory-based test cell. These are the simplest images in terms of clarity, resolution, and lack of clutter. However, each image is taken under different lighting conditions and the part and clip both move slightly.

Three features are defined as shown in Figure 5.8 and repeated here as Figure 5.12. Both the neuro-fuzzy and threshold-based methods were trained on the data from these features.

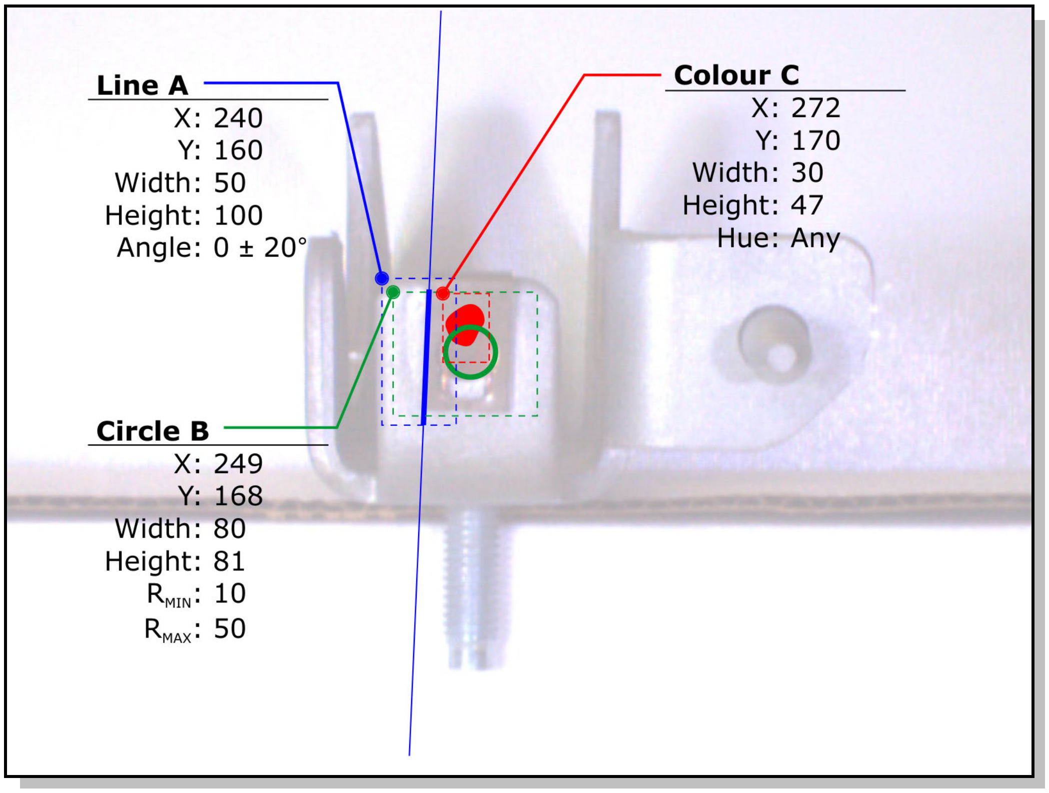


Figure 5.12: Features and feature search parameters used for testing of robustness to lighting change.

For these three features, there are a total of 11 feature attributes. Without optimization, many of the feature attributes do not help in performing correct classification. However, after training with all 11 attributes, a neuro-fuzzy network is created with 27 rules. Optimization selects only two features as the most relevant: the Score of Line A, and the Radius of Circle B. The resulting network is reduced to only 5 rules.

Using a threshold-based classifier on the same data discards all but 4 feature attributes during the grouping process (all other attributes are too evenly dispersed for the grouping algorithm to define a threshold point). Optimization selects the Radius of Circle B as the best feature attribute.

All four classifiers are run with a testing set consisting of 40 of the 100 input images. This is higher than the typical ratio of 10 percent training data (outlined in Section 3.3) because the total dataset size is quite small. The performance is shown in Table 5.1.

Table 5.1: Results of testing Neuro-Fuzzy and Threshold algorithms on images with differences in lighting

Method	**RMS Error**	**Number of False Positives**	**Number of False Negatives**	**Number of Uncertain**
Neuro-Fuzzy	0.1381	0	0	7
Neuro-Fuzzy Optimized	0.0635	0	0	0
Threshold	0.1723	0	0	3
Threshold Optimized	0.0000	0	0	0

In this case, both the Neuro-Fuzzy and Threshold classifiers are able to achieve perfect classification on the testing data. The lower RMS error of the threshold-based system is due to the fact that only two outputs are possible with one feature attribute selected (0 or 1) so if all the features are correctly classified, the error will be exactly zero. For the neuro-fuzzy method, the output is continuous so there is some scatter of the data about the desired values of 0 and 1.

The one thing that these results hide is that the threshold system actually misclassifies one of the images included in the checking dataset. If the original data plot is examined for the feature used by the threshold method (see Figure 5.13), it is clear that there is no exact division point between pass and fail. It just happens that the data point that is part of the overlap was not included in the testing set. Since only one feature is being used by the threshold classifier, the output for this one image switches all the way to the incorrect value. If the confidence in the decision is being taken as the distance of the output from 0.5, then the decision would appear to be very confident.

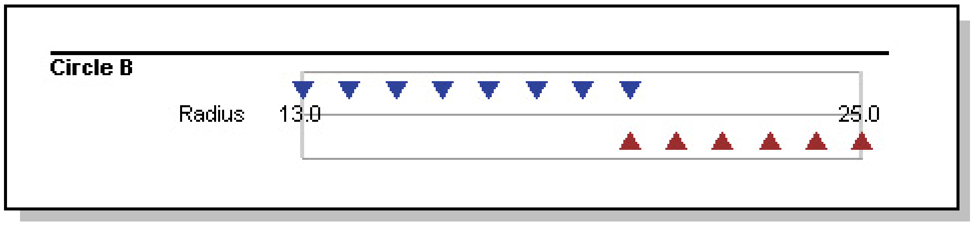


Figure 5.13: Data plot for the feature attribute used by the threshold method. Although the data is well segmented, the two centre points share the same value making perfect separation by a threshold impossible.

By contrast, the neuro-fuzzy classifier is able to correctly classify the same image using additional information from the Line A Score attribute. However, since it is a weaker classification, the confidence is low and the image is marked as ‘uncertain’. Still, that is a better outcome than misclassifying the image with no indication that the results might be wrong.

5.6.4 Real World Variation

The next test involves the dataset provided by Van Rob. These datasets give a more realistic example of the kind of variation that can be expected in lighting and part appearance. In addition, these images are fairly wide angle such that the clips are a smaller part of the total image compared to the laboratory-generated images.

The clearest view of a clip in the set is taken from position 'View 2' and consists of a push type clip which is inserted into a square hole in the part. It is very uncommon for these types of clips to be only partially inserted as they would fall out if not fully inserted. Therefore, the task is only to observe if the clip is present or not.

The database contained 45 images in which the clip was missing and 1597 images where the clip was present. If the system was trained with a set containing this large disparity in examples, it would be fairly poor at correctly classifying fail images due to the lack of examples. Instead, 45 pass images were selected to provide a balanced set. For this task, only two features are defined as shown in Figure 5.14.

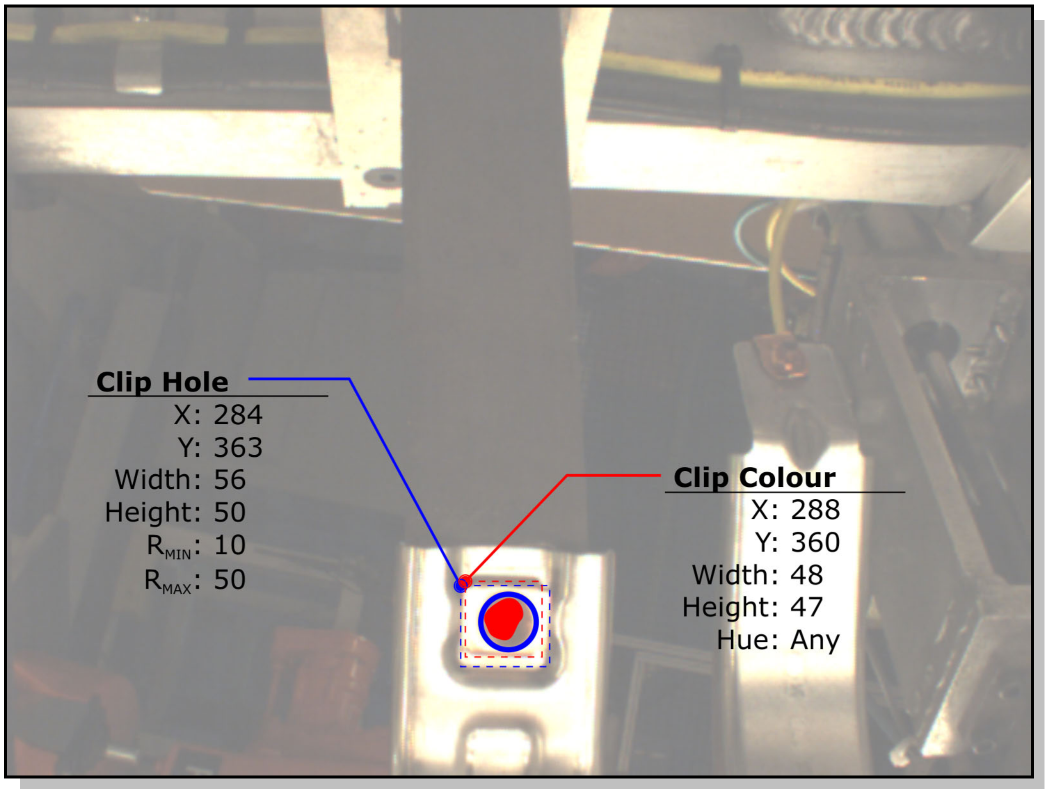


Figure 5.14: Features used for the inspection of a push clip in View 2.

Even with only two features, there are a total of eight feature attributes which can be used. Somewhat surprisingly, there is actually more separation of the pass and fail data for some of the attributes in this set as compared to the laboratory-based test done in the previous section. This may be due to the more consistent lighting found in the real images as compared to the fairly drastic changes which were artificially imposed on the laboratory images. Because of the complete data segmentation on the Score attribute of the Clip Hole, both classifiers selected that attribute during optimization and were able to produce perfect classification as shown in Table 5.2.

Table 5.2: Results on Testing Neuro-Fuzzy and Threshold Algorithms on classification of 'View 2' push clip images.

Method	**RMS Error**	**Number of False Positives**	**Number of False Negatives**	**Number of Uncertain**
Neuro-Fuzzy	0.2595	1	0	12
Neuro-Fuzzy Optimized	0.0192	0	0	0
Threshold	0.2732	1	1	6
Threshold Optimized	0.0000	0	0	0

In this case, there is a distinctive break between the pass and fail data so both methods are able to correctly classify all of the images in the data set. For this type of situation the extra complexity of the neuro-fuzzy system may not be warranted. However, the use of example-based training is still of benefit as it is able to detect the feature attribute that will be the most reliable for classifying the images.

Though the initial results show that the neuro-fuzzy system is not required, it may still be useful if the data starts to 'drift' due to lighting change or dirt build-up on the camera or lights. In that case, the neuro-fuzzy system would output values closer to 0.5 as a warning that things were changing. The threshold-based system, however, would fail suddenly and without warning.

5.6.5 Ability to deal with Poor Viewpoints

The push clip in View 2 is arguably the easiest of the clips to analyse. However, there is another clip in View 2 which is much harder to correctly classify – the J-type clip in the background. It is more challenging for several reasons. The first is that the lighting is poor in that region of the beam because the part is further away and is angled away from the camera. It is also a region where one of the other robots often enters the scene. Also, the clip itself is inserted from the side of the part and needs to be captured in a stamped hole. The part can fail if the clip is missing, but also if the tang of the clip does not engage the hole properly. Occasionally, the beam itself will be bent which fails the part regardless of the clip status.

For the first test, the only goal is to determine if the clip is present or not. Thus in all 52 fail images, the clip was missing completely. To match the fail images, 52 pass images are selected. As this is a harder clip to identify, it is not immediately clear to the user what features should be used. The view is actually of the back of the clip so there are few geometrical features that are unique to the clip. In the end, four features provide the algorithm with plenty of options during the optimization phase (see Figure 5.15).

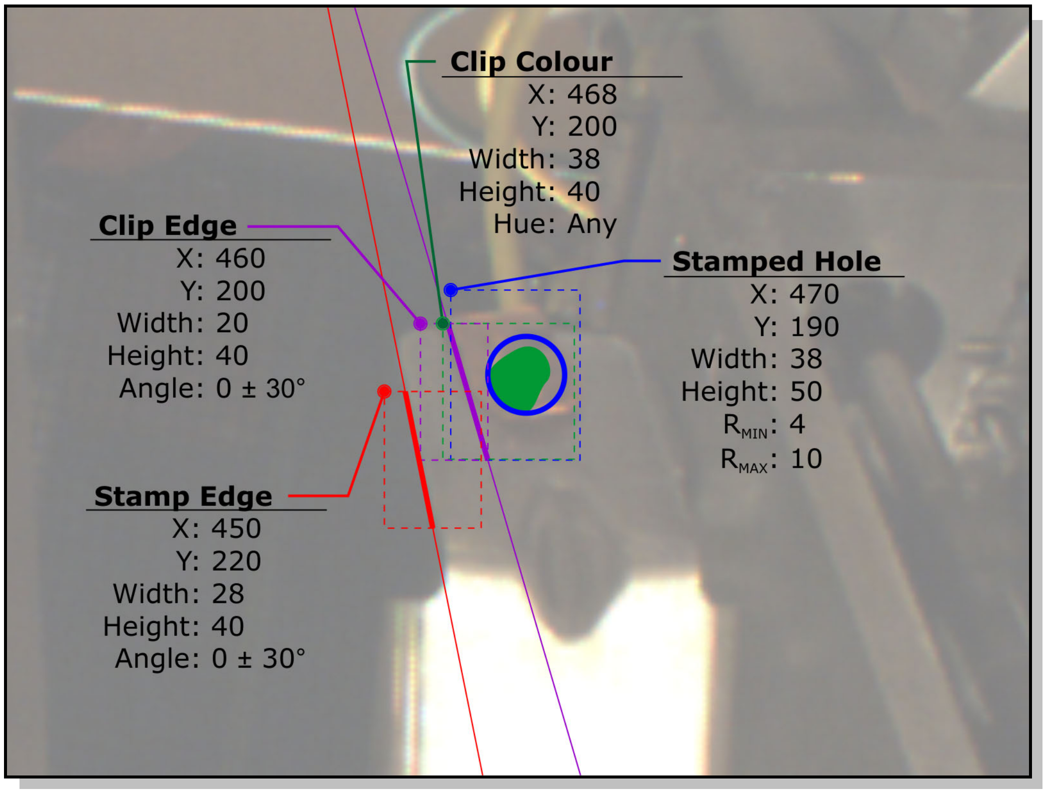


Figure 5.15: A close-up view of the features used for identification of the J-Clip in View 2. Parts of a robot gripper are visible in the background.

Even given the difficulties described above, several of the feature attributes showed good segregation of pass and fail data. The Score of the Stamped Hole feature was the best with the pass and fail data completely separate. Again, both methods optimized to this one feature. The results are shown in Table 5.3.

Table 5.3: Results from identification of clip presence for the J-clip in View 2.

Method	**RMS Error**	**Number of False Positives**	**Number of False Negatives**	**Number of Uncertain**
Neuro-Fuzzy	0.1683	2	0	8
Neuro-Fuzzy Optimized	0.0389	0	0	0
Threshold	0.3032	2	0	1
Threshold Optimized	0.0000	0	0	0

Again, the strength of the algorithm in finding the best feature attributes is displayed by finding a highly segregated attribute that reliably distinguishes pass from fail.

It is a more challenging test if the clip is incorrectly inserted or if the beam itself is bent. Nine images were located in the image set that contain this type of fault and were added to the original fail set. Some examples are shown in Figure 5.16.

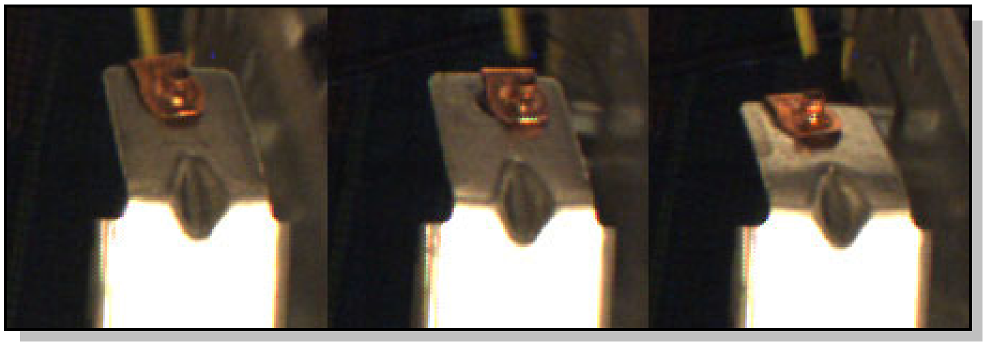


Figure 5.16: Close-up of the J-clip in View 2 showing the challenge in identifying faults (brightness and contrast have been increased for clarity). The left clip is inserted properly, the centre clip tang is not engaged, and the right image shows a bent beam.

The same features are used from the earlier test, but looking the data distribution shows that now none of the attributes show the nice segregation seen earlier (see Figure 5.17)

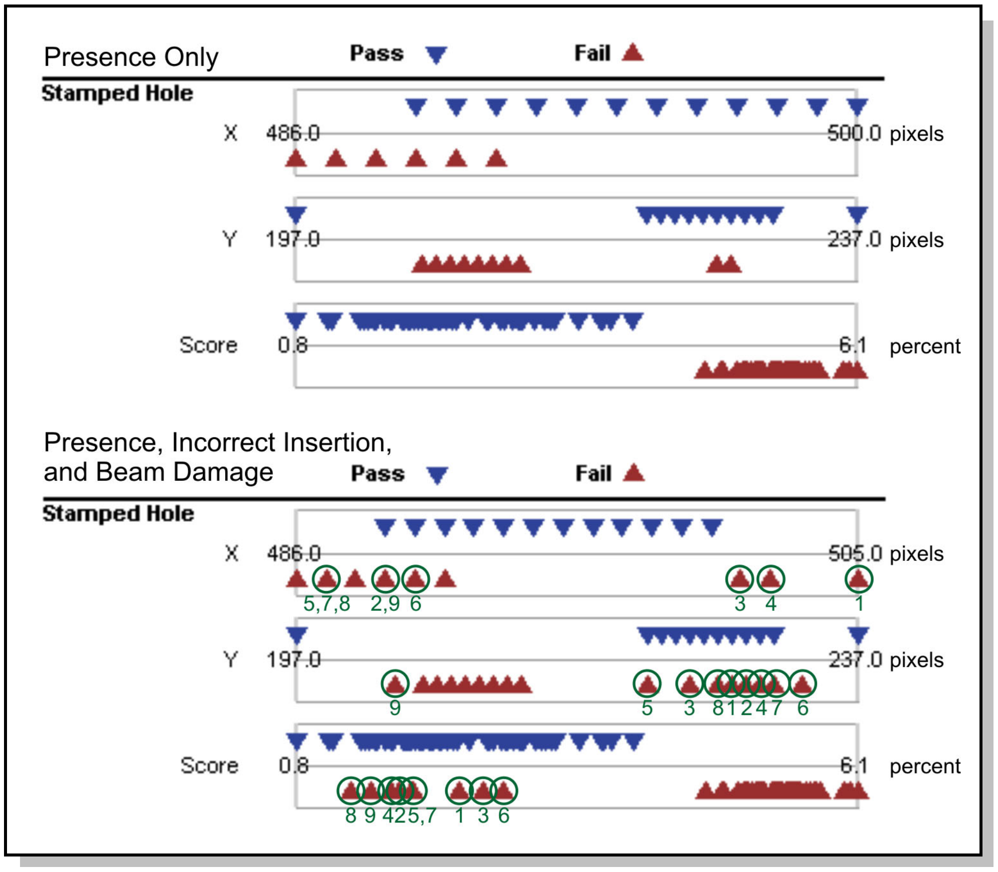


Figure 5.17: Plot of the pass (inverted triangles) and fail (upright triangles) data for the first three attributes for J-clip inspection. The nine additional fail images (circled) make the Hole-Score attribute no longer perfectly segregated.

With all attribute values highly overlapped, both methods struggle to extract enough data to produce the correct classification. Through optimization, both methods select the X-position of the Stamped Hole as the best feature. However, the neuro-fuzzy system is able to improve the result slightly by also considering the Radius of the Stamped Hole feature. The final performance of the methods is shown in Table 5.4.

Table 5.4: Performance of the classifiers on inspection of the J-clip in View 2

Method	**RMS Error**	**Number of False Positives**	**Number of False Negatives**	**Number of Uncertain**
Neuro-Fuzzy	0.2468	3	1	9
Neuro-Fuzzy Optimized	0.3024	1	3	1
Threshold	0.3170	2	1	0
Threshold Optimized	0.3464	2	4	0

Looking at which images are misclassified, the threshold-based classifier is only able to correctly classify 2 of the 9 added images. The neuro-fuzzy system was able to correctly classify 8 of the added images at the expense of misclassifying 3 of the original images. However, the 3 images that are misclassified from the original set are all false negatives (the preferable type of misclassification).

The poor performance of both methods on this dataset suggests that there is not enough information in the feature attribute data in order to correctly classify all of the images. The user could then take this information and attempt to add additional features to help the classification process. If the situation of a bent beam was a common problem, it would also help to provide more examples of this type of failure so they represent a greater proportion of the total training set.

5.6.6 Retraining of the Neuro-Fuzzy Network

The retraining process allows information from the operator to adjust the network parameters when an image is classified as uncertain. The adjusted network is then used for subsequent decisions.

Retraining should result in the image used for retraining to be correctly classified, and to not impact the decisions on already correctly classified images. To test this, the set of laboratory images is divided into two parts. The first part contains 88 images under varying lighting conditions not including LED light alone. The second part contains 12 images taken under only LED lighting. This produces a much more blue tone to the image. The network is initially trained using the first part of the dataset which results in all images being classified properly and low RMS error of 0.0279.

After the network is trained, the second part of the dataset is classified using the network. Since these images are different than the training images, it is expected that several will be classed as ‘uncertain’. In fact, the network is robust enough to correctly classify 4 of the 6 fail images, and 3 of the 6 pass images. The remaining images are classified as uncertain. By manually classifying the uncertain images, the network parameters were adjusted automatically to incorporate the new information. After retraining, all 12 images in the second set are classified properly. Performance on the first dataset shows that all images are still classified correctly and the RMS error remains unchanged.

For comparison, if the threshold system is trained on the first dataset, it also produces no misclassifications. However, when the second set is presented, 4 of the 6 fail images are incorrectly passed.

Through experimentation with the number of features attributes used for the neuro-fuzzy network, it can be determined that retraining works better when more attributes are included. This is probably because there are more possibilities for adjusting the network without affecting previous results. The use of additional attributes also means that it is easier for the network to detect uncertain images in the first place.

5.7 Summary

This chapter starts by outlining the test apparatus that was constructed in a laboratory setting in order to simulate some of the effects that were observed in industrial environments. Once the basic algorithm performance was confirmed, more complex images were used from Van Rob Stampings.

The tests showed that the algorithm excelled in locating the most relevant feature attributes which is often a slow task for human operators. When the neuro-fuzzy system was compared to a threshold-based system, the neuro-fuzzy system performed as well or better and was also able to recognize unusual conditions. A summary of the performance of neuro-fuzzy versus threshold-based classifiers, including applications to other AVI problems is given in Killing et al. (2007).

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

6.1.1 Graphical User Interface

The graphical user interface described in Chapter 4 has proven to be very valuable both in the development and testing of the algorithm, but also in demonstrating the software to industrial partners. A graphical interface is considered essential to any automated visual inspection system due to the visual nature of the data.

6.1.2 Neuro-Fuzzy System vs. Threshold-Based

The tests described in Chapter 5 show that the neuro-fuzzy system can help to improve performance in situations where the input data is not well segmented. The threshold-based system performs as well or better than the neuro-fuzzy system when the data is more separated.

However, there are some other factors to consider when making the choice between the two classifiers. The main one is the way that system degrades. Although the threshold system performs well on the training data, it will not provide any indication that conditions are changing in the input images. When the attribute values cross the threshold, they will instantaneously change to the opposite and incorrect decision. The use of multiple attributes will help as the output is an average of the attribute values and they are not likely to all change at the same time.

Another factor to consider is the ability to retrain the system. The neuro-fuzzy system provides a convenient structure to implicitly store information about the images used in the training set. For a threshold-based system, a more explicit storage method would be required in order to know how far the threshold can be adjusted without affecting previous decisions.

The main advantage to the threshold-based system is that it is very simple. This means it is easy to understand and also fast to compute. However, the neuro-fuzzy classifier developed in this thesis was deliberately kept simple to satisfy those same criteria. Therefore, it is not a ‘black box’ system. It is possible to inspect the network and understand how it makes decisions.

6.1.3 Example-based learning

One of the most useful findings of this work is the advantage of using example-based learning to train AVI systems. Example-based learning allows quick setup and verification of the classifier without relying on the user to guess at the expected variation in the images.

One potential issue is that it may be difficult to obtain a realistic set of example images if the system is not already set up. However, in most cases, the camera system should be able to be installed and run as a 'dummy' inspection station until the system is trained. If human operators are present at another point in the line, then their inspection data can be used to attach the correct classification to the training images.

Another possible source for images would be to set up a lab-based simulation of the real environment. Images could be captured and the network trained to get a basic idea of what features will be important. This would speed development once the real system is in place.

Whichever method is used, the real advantage of using an automated training system is that after a short period of time, a classification system will be developed and performance on the training images will highlight any weaknesses. By providing this information quickly, it removes the necessity for the user to try different combinations of feature attributes and run through tests of various example images. The user is now free to concentrate on the higher level issues such as what features might provide good sources for information. This should speed the development process and reduce the frustration felt during the training of current systems.

6.2 Recommendations

One of the biggest areas for potential improvement of the system as currently developed is in the feature extractors. Many commercial and academic sources are available for optimized feature extractors that will have higher performance than the custom ones written for this project. More reliable feature extractors will make it easier for the network to classify the data correctly.

The strong performance of the threshold-based system also suggests that this may be an avenue for future development. However, as the threshold-based system becomes more complex, it begins to resemble a simple neuro-fuzzy system, so there is a point when more refining of the threshold-based system is no longer warranted.

One issue to consider in terms of industrial implementation is the user acceptance of a neuro-fuzzy system. Although all attempts have been made to keep the system as simple as possible, there still seems to be some fear of the neuro-fuzzy system. Further testing and validation of the system on real production examples may provide enough confidence that this is no longer an issue. A good starting point would be tests with more challenging images than those in the existing databases. For example, images which contain more clip positions, worse lighting, or are out of focus. Van Rob has indicated that this type of image set could be made available to Queen's University for further work.

Another possibility for industrial implementation is to first introduce the example-based training into an existing AVI system. This would increase acceptance of the method and still give many of the advantages of the neuro-fuzzy system (easy training, automatic optimization of feature attributes, automatic threshold setting). Then, as the users become comfortable with the logic behind the algorithm, a complete neuro-fuzzy system could be implemented where conditions warrant its use.

Ultimately, the test of the system will be reliable performance on a real production line for an extended period of time. That is something that is hard to simulate as the problems that arise are often not issues that have been seen in the past. The performance of this system in testing has shown strong potential to be able to solve some of the most common issues in current implementation of AVI systems.

References

Basler Vision Technologies. (2007), "Area scan cameras", *Company Website*, http://www.baslerweb.com/beitraege/beitrag_en_17693.html

Brown & Sharpe. (2007), *Company Website*, http://www.brownandsharpe.com

Chen, B., and Hoberock, L.L. (1996), "Machine vision recognition of fuzzy objects using a new fuzzy neural network", *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, April, pp.1596-1601

Chung, Y. K. and Kim, K. (2006), "Image processing based automatic inspection for assembly line of automobiles", *Key Engineering Materials*, Vol. 321-323 II, pp.1288-1292.

Davies, E.R. (2005), *Machine vision: theory, algorithms, practicalities*, 3rd edition, Morgan Kaufmann, New York, NY

Fiala, M. (2004), "ARTag Revision 1.A fiducial marker system using digital techniques", *National Research Council*, Technical Report #NRC-47419/ERB-1117, November

Finlayson, G.D., and Hordley, S.D. (2001a), "Color constancy at a pixel", *Journal of the Optical Society of America*, Vol. 18, No. 2, Februrary, pp. 253-264

Finlayson, G.D., and Schaefer, G. (2001b), "Solving for colour constancy using a constrained dichromatic reflection model", *International Journal of Computer Vision*, Vol. 42, No. 3, pp. 127-144

Finlayson, G.D., Hordley, S.D., and Hubel, P.M. (2002), "Illuminant estimation for object recognition", *COLOR Research and Application*, Vol. 27, No. 4, August, pp.260-270

Garcia, H. C., Villalobos, J. R., and Runger, G. C. (2006), "An automated feature selection method for visual inspection systems", *IEEE Transactions on Automation Science and Engineering*, Vol. 3, No. 4, pp. 394-406

Gonzalez, R.C, Woods, R.E. and Eddins, S.L. (2004), *Digital Image Processing Using MATLAB*, Prentice Hall, Saddle River, NJ

Ho, S-Y, Lee, K-C, Chen, S-S, and Ho, S-J. (2002), “Accurate modeling and prediction of surface roughness by computer vision in turning operations using an adaptive neuro-fuzzy inference system”, *International Journal of Machine Tools & Manufacture*, Vol. 42, pp. 1441-1446

Hunter, J. J., Graham, J. and Taylor, C. J. (1995), “User programmable visual inspection”, *Image and Vision Computing*, Vol. 13, No.8, pp.623-628

imi Technology Co., Ltd. (2007), “HAN series”, *Company Website*, http://www.imi-tech.com/han.html

Jang, R.J-S. (1993), “ANFIS: Adaptive-network-based fuzzy inference system”, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 3, May/June, pp. 665-685

Kecman, V. (2001), *Learning and Soft Computing*, MIT Press, Cambridge, Massachusetts

Killing, J., Surgenor, B.W., Norman, T. and Mechefske, C.K. (2006), “Flexible machine vision-based parts inspection with neuro fuzzy systems”, *Proc 16th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM)*, Limerick, Ireland, June 26 to 28, pp. 439-446

Killing, J., Surgenor, B.W., and Mechefske, C.K. (2007), “Robust machine vision-based parts inspection: intelligent neuro-fuzzy versus threshold-based classification”, *Proceedings of the 2007 International Manufacturing Science and Engineering Conference*, ASME Paper MSEC2007-31196, Atlanta, Georgia, USA, October 15-17

König, A., Genther, H., Glesner, M., Korn, A., Quint, F., Waleschkowski, N., Henrich, W., Decker, M., and Schahn, M. (1993), “A generic dynamic inspection system for visual object inspection and industrial quality control”, *Proceedings of 1993 International Joint Conference on Neural Networks*, Vol. 2, October, pp. 1243-1246

Lee, K-J, and Bien, Z. (1997), “A model-based machine vision system using fuzzy logic”, *International Journal of Approximate Reasoning*, Vol. 16, pp. 119-135

Levi, D. and Ullman, S. (2006), “Learning to classify by ongoing feature selection”, *Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV’06)*

Lowe, D. G. (2004), “Distinctive image features from scale invariant keypoints”, *International Journal of Computer Vision*, Vol. 60 No. 2, pp.91-110

Luko, S.N. (2000) “Zero defects, statistically considered”. *SAE Technical Paper Series*, Paper 2000-01-2605

Marino, P., Siguenza, C.A., Pastoriza, V., Santamaria, M., and Martinez, E. (2001), "Modelling vision inspection system for fuzzy logic", *Proceedings of The 27th Annual Conference of the IEEE Industrial Electronics Society*, pp. 387-392

Mehrotra, K., Mohan, C.K. and Ranka, S. (2000) *Elements of Artifical Neural Networks*, MIT Press, Cambridge, Massachusetts

Nguyen T.H., and Walker, A.E. (2006), "A first course in fuzzy logic", *Chapman & Hall/CRC*, New York

Norman, T. (2005) "Clip detection using machine vision", *Queen's University Technical Report* #QCL-05-02

Norman, T., Surgenor, B.W., Killing, J., Mechefske, C.K., Bone, G., Demirli, K., Sun, Q., and Xi, F. (2006), "A neuro-fuzzy approach to a machine vision-based parts inspection problem", *Proceedings of the SAE 2006 World Congress*, SAE Paper 06CONG-40, Detroit, MI, USA, April 3 to 6

Novini, A. (1993), "Fundamentals of machine vision lighting", *WESCON'93 Conference Record*, September, pp. 44-52

NREL, "A literature review of the effects of natural light on building occupants", *NREL Technical Report* #550-30769, July

Peng, T. (2006) "Detect circular shapes in a grayscale image", *Matlab Algorithm*, Department of Mechanical Engineering, University of Maryland

Pfragner, M. (2004) "Fast implementation of circular Hough transform", Matlab Algorithm from the *MatLab Exchange Forum*

Pham, D.T. and Bayro-Corrochano, E.J. (1995), "Neural networks for classifying surface defects on automotive valve stem seals", *International Journal of Machine Tools & Manufacture*, Vol. 35, No. 8, pp.1115-1124

Samhouri, M.S. (2005) "A neuro-fuzzy approach to the prediction and control of surface roughness during grinding", *Doctor of Philosophy Thesis*, Department of Mechanical Engineering, Queen's University, Kingston, Ontario, Canada

Shafi, A. (2004), "Machine vision in automotive manufacturing", *Sensor Review*, Vol. 24, No. 4, pp.337-342

Sure Controls. (2007), "Machine vision products and tools (COGNEX)", *Company Website*, http://www.sure.ca/mv_technologies.htm

Tarabanis, K.A., Allen, P.K., and Tsai, R.Y. (1995), "A survey of sensor planning in computer vision", *IEEE Transactions on Robotics and Automation,* Vol. 11, No. 1, February, pp. 86-104

The MathWorks, Inc. (2004), *Image Processing Toolbox: User's Guide*, Version 5, Massachusetts, USA, p.13-24

Thomas, A.W. (2006), "Software leads $3.4B spending", *Quality*, December, pp.42-47

Wang, D, Chai, Q., and Geok, S.N. (2004), "MS-TSKfnn: novel Takagi-Sugeno-Kang fuzzy neural network using ART like clustering", *Proceedings of 2004 IEEE International Joint Conference on Neural* Networks, Vol. 3, July, pp.2361-2366

Wilson, A., (2006) "3-D tracking ensures engine integrity", *Vision Systems Design*, Vol. 11, No. 9, September

Yager, R.R., and Dimitar P.F. (1994), "Approximate clustering via the mountain method", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 24, No. 8, August, pp. 1279-1284

Zuech, N. (2003), "Machine vision and lighting", *Machine Vision Online*, January, http://www.machinevisiononline.org/public/articles/archivedetails.cfm?id=1430

Appendix A

Algorithm Components

A.1 Major Methods used by the Algorithm

The following is a summary of the methods selected for the various components of the algorithm. The primary m-file is also given. Further information about each m-file can be found in Section A.3. Routines that are not part of a built-in MATLAB® library are indicated by ** if they are completely unique or * if they are third party algorithms.

Neuro-Fuzzy Classifier

Creation	Subtractive clustering	*genfis2.m*
Simplification	Custom simplification method	*redSubFis.m***
Training	Backpropagation + least squares	*anfis.m*
Evaluation	Neuro-fuzzy network evaluation	*evalfis.m*
Optimization	Forward search	*optimizeForward.m***

(Running the optimization routine will create the 'neuro-fuzzy optimized' classifier instead of the 'neuro-fuzzy' classifier.)

Threshold Classifier

Creation	Custom grouping method	*thresholdclusters.m***
Evaluation	Average of outputs	*evalThres.m***
Optimization	Exhaustive search	*optimizeThreshold.m***

(Running the optimization routine will create the 'threshold optimized' classifier instead of the 'threshold' classifier.)

Feature Extraction

Lines	Hough line detection based on gradient image (Peng, 2004)	*Hough_Grd_Mod4.m**
Circles	Hough based using gradient value and direction (Peng, 2007)	*CircularHough_Grd_Mod.m**
Holes	Hough based using gradient value only (Pfragner, 2004)	*houghcircle2.m**
Colour	Custom method using reduced HSV colour space	*SegHSV.m***
Squiggle	Custom method based on simple gradient methods (part of *QVision*, but not fully implemented)	*Squiggle.m***

Tracking Tool

Key-point identification	SIFT method	*sift.m**
Key-point matching	Modified version of SIFT match algorithm	*transRegion.m**

Methods Investigated but not Used

Clustering	Discrete incremental clustering, Grid clustering
Rule Formation	Data driven rule generation
Edge Detection	Canny operator, Fixed threshold
Line Extraction	Gradient sum, Best-fit to gradient maximums, Binary based Hough transform
Circle Extraction	Circular unwrapping (warping)
Colour Extraction	Segmentation based on RGB pixel values

A.2 Data Storage Structures

In order to maintain unified data storage between all of the subroutines, standard structures were defined to hold common information. The overall structure is *userData* and its contents are listed below along with the contents of selected substructures.

userData – Overall storage structure for all information related to a project. This is the structure that is saved or loaded with the Project.

```
userData
 .cs             information specific to the configuration window
 .tm             information specific to the training window
 .rt             information specific to the camera window
 .windows        the handles of currently open windows
 .project        name and location of the current project
 .imagesets      location and size of the pass and fail image sets
 .dataMatrix     matrix of all feature attribute values for all images
 .labels         names of the feature attributes in dataMatrix
 .limits         expected range of the attributes in dataMatrix
 .activeMethod   currently selected classifier
 .Methods        information specific to each classifier
 .defFeatures    feature extraction information
 .defFixtures    fixture definition information
 .defAttributes  attribute output information
 .minConf        limit for decision to be considered uncertain
```

.project – project location and identification

```
project
 .name           english name
 .root           full path location for the project root
 .file           full path and file name of the project
 .jobfile        full path and file name of any imported job files
```

.imagesets – example image data information (location and size)

```
imagesets
 .passthumb      full path and name of the pass thumbnail images
 .failthumb      full path and name of the fail thumbnail images
 .imgext         file extension for the images (.bmp, .jpg, or .png)
 .passdir        full path of the location of the pass images
 .faildir        full path of the location of the fail images
 .testdir        full path of the location of the testing images
 .numpass        number of images in the passdir location
 .numfail        number of images in the faildir location
 .numtest       number of images in the testdir location
```

.Methods – Contains information about the classification methods. The content of each subsection varies depending on the classifier.

```
Methods(activeMethod #)
 .name           identifier for the method
 .fullStruct     FIS for neuro-fuzzy, or thresArray for Threshold-based
 .activeStruct   feduction of the fullStruct based on the optimization
 .optimization   currently selected optimization
```

.defFeatures – Holds information about each feature in the current inspection. Parts of the structure that are specific to a particular feature are indicated.

```
defFeatures(Feat #)
 .Type          [feature type]
 .Descrip       english description
 .Fix           fixture number (0 for no fixture)
 .Polarity      polarity of feature (0 for both, 1 for pos, -1 for neg)
 .cX            X location of search area (Lines & Colours = corner,
 .cY            Y location of search area  Circles & Holes = centre)
 .Width         Search box width
 .Height        Search box height
 .AMin          Minimum angle to search (Lines)
 .AMax          Maximum angle to search (Lines)
 .Rin           Inner circle search area radius (Circles)
 .Rout          Outer circle search area radius (Circles)
 .RFind         Radius of hole to search for (Holes)
 .Res           Hue segmentation for color blob finding (Colours)
 .ColorID       Color ID of blob to find (Colours)
```

***.defFixtures* –** Contains information about the fixture definitions. Each fixture is based on the nominal position of one or more features. These features must already be defined in the section above. Features which are based on fixtures cannot be part of that same fixture.

```
defFixtures(Fix #) -> .Feature(fIdx) -> .FNum (matches feature number)
                      .Descrip          .NomR (nominal roe)
                                        .NomT (nominal theta
```

.defAttributes (or defOutputs*) – A matrix that defines which attribute values are active. Each row refers to a particular feature and each column to a feature attribute. The matrix is always four columns wide.

```
defOuputs(feature #, dim #)
dim #s: 1 = X or Theta
        2 = Y or Roe
        3 = Radius for circles, n/a for lines, Color ID for colors
        4 = Score, Area for colors
```

*defOutputs was a term used in early development of the algorithm. The name was later changed to defAttributes to better reflect what this truly represents. However, the old name can still be found in some of the functions.

A.3 Subroutines (m-files)

A.3.1 AutoSearch Directory

Files in this directory are related to the process of automatically expanding search regions by looking at the position of features in a series of images.

autoExpandRegion.m

Calling Syntax:

```
[Xexp, Yexp, Wexp, Hexp, A1exp, A2exp, resMatrix] =
autoExpandRegion(fType, imgList, Xnom, Ynom, Wnom, Hnom, A1nom, A2nom,
R1nom, R2nom);
```

Description:

Attempts to expand the boundaries of a search region by looking for matching SIFT keys between images. FTYPE identifies the type of feature being sought. IMGLIST is a cell array containing the file names of the images through which to search. The first image in IMGLIST is assumed to be the reference image to which the boundary (XNOM, YNOM, WNOM, HNOM, A1NOM, A2NOM) applies. The boundary parameters are those of a standard feature search specification (*x* and *y* of the top left corner, *width*, *height*, and *minimum* and *maximum angle*). R1NOM and R2NOM are used if circles are being tracked. The algorithm will extract the specified feature from the search region in the first image, then locate SIFT key-points near the search window. These will then be matched to SIFT key-points in the other images, and the best-fit mapping found. The mapping is applied to the original search window and then a bounding box formed around the original and transformed regions to define the new search area.

The returned values are the expanded search region (XEXP, YEXP, WEXP, HEXP, A1EXP, A2EXP), and the feature attributes of the feature extracted from each image RESMATRIX.

sift.m

Calling Syntax:

```
[image, descriptors, locs] = sift(imageFile)
```

Description:

This is the SIFT key-point generation routine (written by Lowe). It handles transfer of data to and from the siftWin32.exe which is where the processing takes place. This file must reside in the same directory as the executable.

Takes the name of the image to process (IMAGEFILE), and returns a copy of the IMAGE, a matrix of the local DESCRIPTORS for each key-point, and the locations (LOCS) of those points in the image.

transRegion.m

Calling Syntax:

```
[TRMat, At] = transRegion(Xs, Ys, Ws, Hs, As, des1, locs1, file2)
```

Description:

Takes a search region (XS, YX, WS, HS, AS) and a list of SIFT key-point descriptions (DES1) and locations (LOCS1) and attempts to locate matching points in the image contained in FILE2. FILE2 must be the name of the file, not the image contents. Only points that are near the search region are considered. Once a match is found, the transformation between LOCS1 and the matching points in FILE2 is returned as a transformation matrix (TRMAT). The change in angle between the two frames is also returned as AT. Alternatively, AT can be derived from the elements in TRMAT.

A.3.2 Clustering Directory

Files in this directory are used to generate parts of the neuro-fuzzy network such as rules and membership functions. Functions are also provided to reduce the network size.

formRulesSugeno.m

Calling Syntax:

```
[ffis, mfIdx] = formRulesSugeno(pfis, inData, outData)
```

Description:

This function takes a prototype fuzzy structure (PFIS), and a set of training data pairs (INDATA, OUTDATA), and uses them to generate a new set of rules. The prototype structure is used to obtain the membership function and inputs definitions. To form rules, the input membership functions are activated using the INDATA points. The strongest degree-of-associated members are made the antecedent of a new fuzzy rule unless an existing rule has the same parameters. The output of the new rule is set to the value in OUTDATA. The new rules replace any rules that existed in PFIS, and the structure is returned as FFIS. Optional return MFIDX is a matrix of the antecedent parts of the new rules.

optimizeDims.m

Calling Syntax:

```
[bestFeat, globalMin, bestLabels] = optimizeDims(inFis, trainMatrix,
checkMatrix, threshold, labels)
```

Description:
This is the function for performing an exhaustive search of all of the possible feature attribute combinations to arrive at the best result. INFIS is the fuzzy structure which must be fully complete. TRAINMATRIX contains training points in the first *n* columns and matching output points in the last column. CHECKMATRIX is of a similar format, but is data that will only be used for checking during ANFIS training. THRESHOLD is the checking error at which optimization should stop. If THRESHOLD is set to zero, optimization will continue until all combinations of attributes have been tried. The list of best features is output to BESTFEAT, and the associated checking error as GLOBALMIN. BESTLABELS is a subset of LABELS which contains a cell array of only the names of inputs included in BESTFEAT.

optimizeForward.m

Calling Syntax:
```
[bestFeat, globalMin, bestLabels] = optimizeForward(inFis, trainMatrix,
checkMatrix, threshold, labels)
```

Description:
optimizeForward has the same calling syntax as optimizeDims. However, this function does a forward search of possibilities instead of trying all possible combinations. This is faster, but may miss the global minimum solution. For the forward search, feature attributes are tried independently to determine the best attribute. After the best is selected, the remaining attributes are added to the set one by one to find the best combination of two features. This process continues until: THRESHOLD is reached by the checking error, no further improvement is possible by adding more attributes, or all attributes have been tried.

optimizeThreshold.m

Calling Syntax:
```
[bestFeat, globalMin, bestLabels] = optimizeThreshold(thresArray,
dataMatrix, labels)
```

Description:
This function is the last of the optimization set and performs optimization for threshold-based classifiers. Because the process is much faster to evaluate, only the exhaustive method is provided. THRESARRAY is a cell array of threshold values for each dimension. The first element in each cell has to be a zero or one to represent the class of the first region. For example, if THRESARRAY contains [1 3] for a particular dimension, then a pass region will be defined on –INF to 3, a fail region for 3 to INF.

redSubFis.m

CallingSyntax:
```
[smallFis] = redSubFis(bigFis, mergeThres)
```

Description:
This function takes a FIS structure generated by subtractive clustering (genfis2) and reduces the number of membership functions and rules by merging membership functions with similar ranges and adjusting the rules accordingly. Rules which become identical after this process are merged. BIGFIS is the input fuzzy structure which was generated by genfis2. MERGETHRES is the threshold for which membership functions should be combined. The threshold is evaluated based on the value of the nearest neighbour, evaluated at the centre-point of the current function. The resulting fuzzy structure is output as SMALLFIS.

reduceFIS.m
Calling Syntax:
```
[ffis] = reduceFIS(pfis, usedF)
```

Description:
Reducefis takes an input fuzzy structure (PFIS) and a vector of feature attributes (USEDF) which should be kept. The remaining inputs of PFIS are eliminated along with any membership functions, rules, and outputs that refer to those inputs. Rules which are redundant after eliminating the unused inputs are merged. The final structure is returned as FFIS.

reformFIS.m
Calling Syntax:
```
[bestFis] = reformFIS(inFis, trainMatrix, checkMatrix, usedF)
```

Description:
ReformFIS calls reduceFIS so it does all of the things mentioned under reduceFIS. However, it also takes the reduced fuzzy structure and retrains it with the training data in TRAINMATRIX and CHECKMATRIX. The reduced, trained structure is returned as BESTFIS.

thresholdclusters.m
Calling Syntax:
```
[thresArray] = thresholdclusters(inputdata, minpoints, minsize)
```

Description:
This is the threshold equivalent of genfis2. It takes input of INPUTDATA, a matrix of feature vectors with the last column containing a clustering indicator (in this case 0 for fail and 1 for pass). Data is clustered by looking at each column of INPUTDATA and finding continuous regions with the same classification which contain at least MINPOINTS and have a span of MINSIZE $\times$ the range of that column. Once the regions

are identified, the boundaries are taken as the average between the locations of the boundary points in each region.

These boundaries are returned as the cell array THRESARRAY. Each cell of THRESARRAY contains an initial element which is the classification of the first region. The following points are the locations of the boundary points. Classification is assumed to switch across each boundary. Elements of THRESARRAY which are empty indicate no threshold setting is possible for that dimension with the current settings.

A.3.3 Feature Extractors Directory

This directory contains functions related to finding features in the image. It also has some functions for generating data sets from a series of images.

CircularHough_Grd_Mod.m

Calling Syntax:

```
[varargout] = CircularHough_Grd(img, radrange, varargin)
```

Description:

This is the method written by Peng (2006) for extraction of circles from greyscale images using the gradient direction to reduce the complexity of the task. It has been modified slightly for this project in order to only return the strongest circle in IMG and also to use the built-in Matlab peak finding routine. IMG should be a greyscale image of only the region-of-interest. RADRANGE is a two element vector containing the minimum and maximum radii to search for. The remaining parameters are optional and modify the internal settings of the algorithm. More details are available by typing: `help CircularHough_Grd`.

The first output is the centre of the circle (*x*, and *y*), the second output is the detected radius, and the last is the score of that circle. All outputs are optional, and reducing the number of outputs will reduce the processing time.

ColorBlob.m

Calling Syntax:

```
[fVect, limits] = ColorBlob(srcImage, X, Y, Width, Height, Resolution,
ColorID)
```

Description:

ColorBlob is a wrapper for the colour feature extraction. It prepares the image before passing it to the segHSV function. SRCIMAGE is the full size, colour image from the camera or disk. X, Y, WIDTH, and HEIGHT define a box within the SRCIMAGE which is the region-of-interest. Only this part of the image is passed to segHSV. RESOLUTION defines how many segments will be used to divide the hue space. COLORID is the segment number of the hue-of-interest. Setting COLORID to zero extracts the hue with the largest area.

The return value FVECT is a four element vector which contains the elements [Centroid X, Centroid Y, ColorID, and Area]. If COLORID was specified as zero, then the output contains the ID of the hue with the largest Area. Otherwise, the return matches the input. LIMITS is a 4 by 2 matrix containing the expected range of the values in FVECT. These values can be used to scale the output to normalized values.

CreateTraining.m

Calling Syntax:

```
[ANFISMatrix, limits, labels] = CreateTraining(passFileList,
failFileList, defFeatures, defFixtures, defOutputs)
```

Description:

This is a high level function which creates a set of training data by extracting features from a set of images. PASSFILELIST contains the full names of a series of images to be used as examples of good parts. FAILFILELIST is the same but for bad parts. The images must exist in the specified locations when this function is run. DEFFEATURES, DEFFIXTURES, and DEFOUTPUTS contain the description of the features to be extracted. They are defined as described in Section A.1

The output of the function is a matrix (ANFISMATRIX) which contains the feature attribute values (columns) for all of the images (rows). The last column contains the classification (one or zero) based on whether the image was in the pass or fail list. LIMITS is the expected range of the values in each column, and LABELS is the name associated with each column (this is generated from the user input name stored in DEFFEATURES combined with the attribute name).

ExtractFeature.m

Calling Syntax:

```
[fVect, limits] = ExtractFeature(srcImage, fNum, defFeatures, fixDef)
```

Description:

This is the umbrella function for all of the feature extraction routines. It takes the full colour SRCIMAGE, and DEFFEATURES structure, and fixDef (the DEFFIXTURES structure) along with an index to DEFFEATURES which determines which features should be extracted (FNUM). The function analyses the entry for the selected feature in DEFFEATURES, selects the appropriate feature extractor, and passes the necessary information to that extractor. The return value is FVECT and LIMITS which are direct copies of the values returned by the feature extractors.

FeatureFileInput.m
Calling Syntax:
```
[defFeatures, defFixtures, defOutputs] = FeatureFileInput(fileName)
```

Description:
This is a series of three functions which handle input and output to the custom file format used by QVision. Changes to the software have moved to a standard format where all information is stored in a Matlab .mat file. However, the feature file input/output is still useful for importing feature search definitions into the QVision environment.

FeatureFileMod.m
Calling Syntax:
```
FeatureFileMod(defOutputs, fileNameIn, fileNameOut)
```

Description:
Opens the feature file FILENAMEIN, and edits the attribute output portion of the file using the definition in DEFOUTPUTS. The rest of the file is unchanged and the result is stored into FILENAMEOUT. Use to block certain attributes for later processing.

FeatureFileOutput.m
Calling Syntax:
```
FeatureFileOutput(fileNameOut, defFeatures, defFixtures, defOutputs)
```

Description:
Writes a feature file to FILENAMEOUT using the information store in DEFFEATURES, DEFFIXTURES, and DEFOUTPUTS.

FindFixture.m
Calling Syntax:
```
[fix, fMat, lMat] = FindFixture(srcImage, defFeatures, defFixtures,
fMat)
```

Description:
The function is used to locate fixtures in the image by first extracting the features in DEFFEATURES that make up the fixture definition (as defined in DEFFIXTURES), and then computing the average transformation between the nominal feature positions and the ones found in the image SRCIMAGE. The formation of the transformation is based on the type and number of features included in the fixture. FMAT contains the extracted feature information (one row per feature). If any of the rows of FMAT are not empty, that data is used for locating the fixture instead of having to re-extract the feature from the image.

FIX is a three element vector with the transformation deltas in *x*, *y*, and *theta*. FMAT contains the information from the extracted features, and LMAT contains the expected range (limits) of the values in FMAT.

GetFeatures.m

Calling Syntax:

```
[featVect, limits, labels] = GetFeatures(srcImage, defFeatures,
defFixtures, defOutputs)
```

Description:

Extracts all of the features defined in DEFFEATURES from the SRCIMAGE. If fixtures are needed, FindFixture is called with the information in DEFFIXTURES. Once the features are extracted, the feature vectors from each feature are placed end to end to form a row vector (FEATVECT). LIMITS and LABELS correspond to the expected range and name of each of the entries in featVect respectively.

Hough_Grd_Mod4.m

Calling Syntax:

```
[varargout] = Hough_Grd_Mod4(img, varargin)
```

Description:

This function is a modification of the algorithm develop by Peng (2004) to create the Hough space image. It has been modified to allow the use of a mask to eliminate unwanted portions of the image (IMG). IMG must be a grayscale image of only the region of interest. VARARGIN can be used to give the function a *mask* to block out unwanted portions of the image. The mask must be a binary image the same size as IMG with ones for pixels which are to be kept.

There are three optional outputs: the Hough space image, a vector of the angles represented by each pixel in the horizontal direction, and a vector of the perpendicular distances represented by each pixel in the vertical direction.

houghcircle2.m

Calling Syntax:

```
[Accumulator, xdim, ydim] = houghcircle2(Imbinary,r)
```

Description:

This is a method by Pfragner (2004) which create the Hough space image for circle detection with a fixed radius. It is unmodified from the original. IMBINARY is an edge image and R is the desired radius of circle to detect. The return is the Hough space image ACCUMULATOR, and XDIM, YDIM are the dimensions of ACCUMULATOR.

HoughCircleDetectGrd.m

Calling Syntax:

```
[fVect, limits] = HoughCircleDetectGrd(srcImage, X, Y, Width, Height,
Rin, Rout)
```

Description:

This function is a wrapper method for the detection of circles using the CircularHough_Grd_Mod algorithm. It takes the SRCIMAGE and extracts the portion specified by the region of interest (X, Y, WIDTH, HEIGHT). This is passed to the Hough method along with the specified radius ranges (RIN and ROUT). The returned values are a vector of feature attribute values (FVECT) containing four elements: [centre X, centre Y, radius, and score], and the expected range of those values contained in LIMITS.

HoughHoleDetect.m

Calling Syntax:

```
[fVect, limits] = HoughHoleDetect(srcImage, X, Y, Width, Height, Rfind,
Polarity)
```

Description:

This function is a wrapper method for the detection of holes using the houghcircle2 algorithm. It takes the SRCIMAGE and extracts the portion specified by the region of interest (X, Y, WIDTH, HEIGHT). This is passed to the Hough method along with the specified radius (RFIND). POLARITY is no longer used but remains for legacy purposes. The returned values are a vector of feature attribute values (FVECT) containing four elements: [centre X, centre Y, radius, and score], where the radius will always equal RFIND. The expected range of the FVECT values is contained in LIMITS.

LineDetectHough.m

Calling Syntax:

```
[fVect, limits] = LineDetectHough(srcImage, X, Y, Width, Height,
AngleMin, AngleMax, Polarity)
```

Description:

This is the main line extraction algorithm. It takes the full image SRCIMAGE, applies the region of interest (X, Y, WIDTH, HEIGHT, ANGLEMIN, ANGLEMAX), and uses the Hough_Grd_Mod4 algorithm to generate the Hough space image. The feature attributes of the line are returned in FVECT as four elements: [Angle, Perpendicular distance, 0, and Score]. The extra zero is used to maintain the feature vector length at four elements. POLARITY is no longer used but is kept for legacy purposes.

segHSV.m

Calling Syntax:

```
[colorMatrix] = segHSV(srcimage, numSect, colorIdx, numRtn, satThres,
minSize)
```

Description:
segHSV is used to extract colour information from the image. It takes SRCIMAGE, a full colour image of only the region of interest and converts it to the HSV colour model. The hue space is then divided into NUMSECT sections and pixels are grouped together into like sections. The largest (by area) NUMRTN groups are returned for all of the sections listed in COLORIDX. For instance calling with NUMSECT = 8, COLORIDX = [1,3], and NUMRTN = 1 will return only the largest colour area in sectors 1 and 3 of the hue space when the hue space is divided 8 ways. SATTHRES determines what is counted as a colour based on the saturation value of the pixel.

COLORMATRIX is NUMRTN by 4 matrix of the properties of the colour regions. The four columns are: [Centroid X, Centroid Y, Hue (0 to 360), Area]. If fewer than NUMRTN areas are identified, the matrix will contain fewer than NUMRTN rows. If NUMRTN = 0, then the properties of all identified regions will be returned.

Squiggle.m

Calling Syntax:
```
[fVect, limits, rtnImage] = Squiggle(srcImage, X, Y, Width, Height,
Segments, Polarity)
```

Description:
Squiggle is used to extract a ‘characteristic’ shape from the image. It takes the full image SRCIMAGE and applies the region of interest (X, Y, WIDTH, HEIGHT). This region is divided into SEGMENTS sections. SEGMENTS is a two element vector [XSEGMENTS, YSEGMENTS]. POLARITY is also two elements and determines which direction of line will be included in the calculations. The elements in POLARITY can have values of 0 (both polarities accepted), -1 or 1.

The returned data is FVECT which lists pairs of displacements of the lines from the edge of the image, and the associated line score. FVECT is four elements wide but the centre two elements are always zero. This is to comply with the standard 4 element size. Each element of the SQUIGGLE will occupy one row of FVECT. SQUIGGLE is the only feature extractor that returns non-standard feature vectors.

A.3.4 siftDemoV4 Directory

This directory primarily contains the original SIFT algorithms as developed by Lowe (2004). These are: appendimages.m, match.m, showkeys.m, and sift.m. More information about these functions can be found by typing `help` followed by the function name.

A.3.5 GUI Directory

This directory can be separated into two parts. M-files associated with the graphical user interface, and m-files which are stand-alone subroutines. The files associated with the user interface consist of .m/.fig pairs.

AutoSearchDialog.m/.fig	The feature searching window
ClassifyDialog.m/.fig	Question box for operator pass/fail decision
ConfigSystem.m/.fig	The configuration window screen
DisplayStatsDialog.m/.fig	The feature statistics window
FixtureEditing.m/.fig	Window used to modify the fixture definitions
HomePage.m/.fig	For future use (unified QVision start screen)
PlotMatrix.m/.fig	Feature search results window
RealTime.m/.fig	Camera window screen
SelectFisDialog.m/.fig	Allows the user to load a fuzzy structure from disk
TrainingModule.m/.fig	Training window screen
QVision.m	Initializes Matlab and starts the QVision program

All of the above functions share the data structure *userData* as described in Section A.1 of this Appendix. The remainder of the functions in this directory are associated with some aspect of the GUI but do not open windows of their own.

CMdisplayfeature.m, CMdisplayImg.m, CMpopulatepanel.m, CMpopulatepopups.m

Calling Syntax:
```
CMdisplayfeature(handles), CMdisplayImg(handles),
CMpopulatepanel(handles), CMpopulatepopups(handles)
```

Description:
These are a family of functions that assist is running the Configuration screen. They do not return any values and can only be called from the configuration window. HANDLES is the Matlab structure containing the handles to the objects on the Configuration screen. CMdisplayfeature extracts the current feature from the current image and shows the results. CMdisplayImg reads the current image from disk and displays it in the main window. CMpopulate panel formats and fills in the left side panel of the Configuration window. CMpopulatepopups fills in the entries in the drop-down lists on the Configuration screen

ProjectNew.m, ProjectOpen.m, ProjectSave.m, ProjectSaveAs.m

Calling Syntax:
```
ProjectNew, ProjectOpen, ProjectSave, ProjectSaveAs
```

Description:
These are another family of helper functions. In this case, any of the three QVision windows can call these functions in order to Open, Save, or create a New project file. The project file is shared between all windows. The data saved to the project file is a copy of the *userData* structure (defined in Section A.2 of this Appendix). For the Save function, the existing file name and path are taken from userData.project.file. If this entry is empty, then the user is prompted for a file name and location. Open, New and SaveAs always prompt the user for a file name and location. There no inputs or outputs as all data is shared using the *userData* structure.

RefreshGUI.m
Calling Syntax:
`RefreshGUI(guihandle)`

Description:
RefreshGUI takes the handle of a currently open window (GUIHANDLE) and refreshes the data in that window. The name property of the window must be recognized as one of the three standard windows: `'QV - Configuration', 'QV - Training'`, or `'QV - Camera'`. The window is refreshed by deleting it and then reloading. This invokes the window startup code which displays the current data in the window.

initBlankUD.m
Calling Syntax:
`initBlankUD`

Description:
This function has no inputs or outputs. It just initializes the *userData* structure so it has all of the parts as listed in Section A.2. All three windows call this function on startup if the *userData* structure doesn't already exist. If it does exist, then the window will use the existing data and not call this function.

ncolor.m
Calling Syntax:
`[tricolor] = ncolor(colorID)`

Description:
In order to unify the colour palette for all windows in the software, ncolor provides a standard set of colours that can be accessed for plotting functions. The function is called with the text name of the colour and the RGB triplet is returned. If the name is not recognized, the colour black will be returned (0,0,0). Currently defined colours are: `'white', 'black', 'blue', 'red', 'green', 'purple', 'yellow', and 'grey'`. Each colour (except white and black) can be modified by a preceding `'lt'` or

`'dk'` for a light or dark tone of the same colour. Note that this function is not fully implemented in all plotting functions so changing the values in ncolor may not change all display colours.

A.3.6 OnlineAdapt Directory

This directory contains two functions which are used to perform neuro-fuzzy network retraining functions.

ReTrainFis.m

Calling Syntax:

```
[modFIS, msgRtn, chkERR] = ReTrainFis(oldFIS, newFVect, newOutput,
scoreMin, chkData)
```

Description:

After the neuro-fuzzy network is trained, it can be retrained by incorporating new information to adjust the network parameters. This function takes the current FIS structure (OLDFIS), and attempts to incorporate the new data NEWFVECT, so that the output matches NEWOUTPUT. SCOREMIN is the minimum confidence value required so that the retrained network will not output in the uncertain region. CHKDATA can be used if performance of the new network on specific data points is desired. CHKDATA is a matrix of feature attribute values with the last column containing the correct classification.

If the adjustment can be made, the new network is output as MODFIS and the rms error on CHKDATA is output as CHKERR. If no adjustment can be made then MODFIS is just a copy of OLDFIS. MSGRTN contains a text string which explains how the network was modified (or why it was not).

getMFResponse.m

Calling Syntax:

```
[firedMF, strenMF] = getMFResponse(FIS, evalVect)
```

Description:

This function takes a FIS structure and a feature attribute vector EVALVECT that matches the number of inputs in FIS. It calculates which membership function in each input dimension has the highest degree of association with the data in EVALVECT. FIREDMF contains the highest scoring membership functions, and STRENMF contains the degrees of association for the membership functions in FIREDMF.

A.3.7 Utilities Directory

This directory contains functions that are not directly related to a particular category but assist in some part of the process. They include plotting and display functions, camera setup functions, and other smaller tools that were missing from the Matlab built-in functions.

addOverlay.m
Calling Syntax:
`addOverlay(fType, pVect)`

Description:
This function adds a graphical display on top of the current image. FTYPE determines the type of graphic to be shown and PVECT contains the data for plotting. A legend is also created and maintained through multiple calls to the function. If the legend is deleted externally, addOverlay will restart the legend on the next call. Allowed FTYPEs are [BOUND] for drawing boundary regions, and [CIRCLE], [LINE], [HOLE], [COLOR], [SQUIGX], [SQUIGY] for plotting feature extraction results. PVECT has different meanings depending on the value of FTYPE but is typically an augmented version of the feature vector.

appendFileName.m
Calling Syntax:
`appendFileName(srcDir, addedend)`

Description:
Appends the text string ADDEDEND to all of the image files contained in SRCDIR but maintains the original extension. Original files are replaced by the newly named ones. The file contents are transferred too so it can take a while for large directories.

BoundedBox.m
Calling Syntax:
`[X1, Y1, X2, Y2] = BoundedBox(ThetaDeg, Roe, Height, Width)`

Description:
BoundedBox intersects the line defined by THETADEG (angle), and ROE (perpendicular distance to origin) with the box (HEIGHT, WIDTH). It is assumed that the box starts at the origin. (X1, Y1) and (X2, Y2) are the intersection points.

BoundedBoxFancy.m
Calling Syntax:
```
[X1, Y1, X2, Y2] = BoundedBoxFancy(ThetaDeg, Roe, X, Y, Width, Height,
Angle)
```

Description:
This is a more advanced version of BoundedBox which allows the box (X, Y, WIDTH, HEIGHT, ANGLE) to be angled and to have an arbitrary origin. Intersection points are returned as (X1, Y1, X2, Y2).

CheckHit.m
Calling Syntax:
```
[locX, locY] = CheckHit(hObject, checkPnt)
```

Description:
This function is called by one of the graphical user interfaces. It looks at the boundaries of the object referenced by HOBJECT and determines if CHECKPNT is within those boundaries. If it is, LOCX and LOCY are the coordinates of CHECKPNT relative to the origin of HOBJECT. If the point is outside the object boundaries, LOCX and LOCY will be empty. This function is useful in determining when the user cursor is over objects on the screen.

classDir.m
Calling Syntax:
```
classDir(srcDir)
```

Description:
This is a useful function for classifying a set of unclassified training images. All of the images in SRCDIR are displayed, one at a time and the user is asked to provide the correct classification. A copy of the classified images is stored in a directory with the name "Class_" followed by the user defined name of the class. If this function is run on a directory that already has directories with that name format, these are suggested as possible class names. The user also has the option to start somewhere in the middle of the set. An undo button allows the user to go back *once* if a mistake is made.

clip.m
Calling Syntax:
```
[clipped] = clip(data, limit1, limit2)
```

Description:
This function restricts the value of DATA to lie between LIMIT1 and LIMIT2. The restricted value is returned as CLIPPED. The order of LIMIT1 and LIMIT2 does not matter.

CloseCamera.m
Calling Syntax:
```
CloseCamera(vid)
```

Description:
Takes the VID structure associated with an open camera and closes the camera. The VID structure is also deleted.

ConfigureCamera.m
Calling Syntax:
```
ConfigureCamera
```

Description:
This is not a function. It acts directly in the current environment. Because it operates in Matlab *cell* mode, parts of it can be run separately. The main purpose of the m-file is to be able to play with the camera settings to get them adjusted for a particular hardware setup. To adjust the values, they must be changed in the code itself.

The last section is a semi-automatic white balance adjustment. Run this segment when the camera is facing a white surface and the iris and gain have been adjusted so that the white surface appears grey (no values are saturated). The routine will then try to adjust the white balance parameters until the red, green, and blue are equal.

Values from this process must be transferred manually into the OpenCamera function.

evalfisRMS.m
Calling Syntax:
```
[RMSErr, MisClass] = evalfisRMS(DataMatrix, FIS)
```

Description:
Evaluates the output of the FIS structure for the values in DATAMATRIX, compares them to the ideal output (last column of DATAMATRIX), and returns the rms error (RMSERR). MISCLASS contains the indices of the rows in DATAMATRIX that were more than 0.5 away from the correct classification.

evalthres.m
Calling Syntax:
```
[passfail, voted, output] = evalthres(fVect, thresArray)
```

Description:
This is the equivalent of *evalfis* for the threshold-based method. FVECT is the vector or matrix of points to evaluate. THRESARRAY is the definition of the threshold transition points.

PASSFAIL is a vector of zeros and ones indicating the results for each row in FVECT. The output is pass if and only if all of the feature attributes fall into a pass region. VOTED is also the output for each row in FVECT but is the average output for all of the attributes. OUTPUT is a matrix of values that gives the output for each individual attribute.

fixdec.m

Calling Syntax:
```
[rndnumber] = fixdec (number, decplaces)
```

Description:
Returns RNDNUMBER, a NUMBER rounded to the specified number of decimals (DECPLACES).

GenerateImageSet.m

Calling Syntax:
```
[]=GenerateImageSet(baseFileName, varargin)
```

Description:
This function assists in capturing a series of images from the camera for use in training. BASEFILENAME will be used as the first part of all file names. The remainder will be a three digit number starting from “000”. Images are always saved in bitmap format (.bmp). The camera is automatically opened, and a preview of each image is shown. Image are saved to the current directory and overwrite will occur automatically if the same file names already exist in that directory. The number of images to acquire can be entered as VARARGIN, or the user will be prompted on start-up.

GetCameraImage.m

Calling Syntax:
```
[rtnImage] = GetCameraImage(vid)
```

Description:
Captures an image from the camera referenced by VID and returns it as RTNIMAGE.

getFisLimits.m

Calling Syntax:
```
[limits] = getFisLimits(FIS)
```

Description:
Reports the upper and lower bounds of each input dimension in FIS and returns as a matrix of values LIMITS.

intersectLines.m

Calling Syntax:
```
[X, Y] = intersectLines(Th1, R1, Th2, R2)
```

Description:
Finds the intersection point (X,Y) between two lines (TH1, R1) and (TH2, R2) defined using the angle/distance from the origin method. If no intersection is found X and Y are NAN.

intline.m
Calling Syntax:
```
[x,y] = intline(x1, x2, y1, y2)
```

Description:
This function was copied from the text by Gonzalez et. al. (2004) and produces two vectors of coordinates which are the integer coordinates of all of the points between (X1, Y1) and (X2, Y2). Used for mapping a line segment onto pixels.

makerotmat.m
Calling Syntax:
```
[rotmat] = makerotmat(Angle)
```

Description:
Makes a rotation matrix (ROTMAT) based on the specified ANGLE.

OpenCamera.m
Calling Syntax:
```
[vid] = OpenCamera
```

Description:
Opens any connected DCAM compliant camera which matches the names of either the Basler or imi tech camera. The camera must be configured to work with Matlab first. To check operation, type "`imaqhwinfo('dcam')`". The DeviceInfo field should be of size 1x1 if the camera is connected and working properly. The output VID is used to identify the camera for later functions.

overlayImages.m
Calling Syntax:
```
compositeimg = overlayImages(baseimg, overlayimg)
```

Description:
Show the edge image of OVERLAYIMG as a red outline overtop of BASEIMG. The composite image is returned as COMPOSITEIMG. This function is used when aligning the camera image with a saved image.

plotColorWheel.m
Calling Syntax:
`plotColorWheel(axishandle, segments, selection)`

Description:
Plots a mini colour wheel to AXISHANDLE with SEGMENTS number of segments evenly spaced in the hue range. If SELECTION is greater than zero, the SELECTION# of SEGMENT is highlighted.

plotcube.m
Calling Syntax:
`plotcube(srcimage, n)`

Description:
Plots the colours in SRCIMAGE in a 3d cube with axis of red, green, and blue components. This function can be used to look at the colour distribution in an image.

plotDataOverMembers.m
Calling Syntax:
`plotDataOverMembers(varargin)`

Description:
This function plots the data values from DATAMATRIX over the membership function display so there is some reference for the membership function locations. Type: `help plotDataOverMembers` for usage.

plotDMat.m
Calling Syntax:
`[varargout] = plotDMat(varargin)`

Description:
This function plots the data values from DATAMATRIX with pass and fail data separated. Outputs a handle to the axes used for plotting. Type: `help plotDMat` for usage.

plotMembers.m
Calling Syntax:
`plotMembers(varargin)`

Description:
This function plots the membership functions for a particular input dimension of the supplied FIS structure. Type: `help plotMembers` for usage.

plotOptimize.m
Calling Syntax:
```
[varargout] = plotOptimize(varargin)
```

Description:
This function plots the results of the optimization runs with the current run highlighted. Outputs a handle to the axes used for plotting. Type: `help plotOptimize` for usage.

plotStructure.m
Calling Syntax:
```
plotStructure(varargin)
```

Description:
This function plots neuro-fuzzy structure for ANFIS-based FIS structures only. Type: `help plotStructure` for usage.

plotTestResults.m
Calling Syntax:
```
[FPMat, FNMat] = plotTestResults(varargin)
```

Description:
This function plots the test results and calculates the statistics for the neuro-fuzzy classifier. Outputs are the indices of the false positives (FPMAT) and false negatives (FNMAT). These are also listed in the main window when the function is run. Type: `help plotTestResults` for usage.

plotTestResultsThreshold.m
Calling Syntax:
```
[FPMat, FNMat] = plotTestResultsThreshold(varargin)
```

Description:
This function plots the test results and calculates the statistics for the threshold classifier. Outputs are the indices of the false positives (FPMAT) and false negatives (FNMAT). These are also listed in the main window when the function is run. Type: `help plotTestResultsThreshold` for usage.

setFisLimits.m
Calling Syntax:
```
[modFIS] = setFisLimits(FIS, limits)
```

Description:
This function sets the input dimension limits of the FIS structure to the values in LIMITS. The updated structure is returned as MODFIS.

Appendix B

QVision User's Guide

B.1 Introduction

QVision is a machine vision inspection system designed to provide the user with access to neuro-fuzzy algorithms. These so-called 'soft computing' methods attempt to mimic a more human inspection method that can accommodate changes in lighting, part orientation, and appearance in order to focus on the important requirements.

Training is accomplished with a simple process whereby examples of good and bad parts are presented to the system. The neuro-fuzzy algorithm then determines which features of the images are important, and how to use that information to correctly classify the images. One of the major advantages of this system is the ability to use multiple images to train the inspection system. This gives a much better idea of the range of values expected in actual operation and can reduce retraining time later.

At the moment, this software is intended for demonstration purposes and represents experimental work. It is not yet intended for production use.

B.1.1 Installation

The algorithm requires Matlab® R2007b or later to be installed on the host system. The software is not compiled and requires the compiler in Matlab® to run. The installation needs to have rights to the *Fuzzy Toolbox, Image Acquisition Toolbox,* and *Image Processing Toolbox*.

Seting up the environment

1. Install the drivers for the IMI camera (see document "Installing the imi-tech camera with Matlab")
2. Copy all of the files and directories to a convenient location on the hard drive.
3. Open the Matlab® program and choose Set Path from the File menu.
4. Choose Add with Subdirectories and navigate to the directory into which you copied the algorithm files. This associates the directory so Matlab® can find the subroutines.

Note The algorithm can technically operate with any dcam compliant IEEE-1394 (Firewire) or USB camera. However, manual modifications are required to the code in order to operate properly with any camera other than the imi tech IMC1080FT or Basler 312fc models.

Startup

To run the algorithm, type `QVision` at the Matlab® command prompt. This will open the first window (the configuration window).

B.2 Inspection Example

The following is a quick introduction to the software's capabilities as illustrated through the example files provided. A more detailed description of each part of the program is found in the next section.

1 Run Matlab®, and start the algorithm by typing `QVision` to start the program and the *Configuration Window* should appear.

Note The live link to the camera is not established until the *Camera* window is opened in Step 14.

2 From the ***File*** menu, select ***Open Project***. Navigate to the *Examples* directory and select *DemoStart.mat* and then click on *Open* to load this project file. The project file contains information about the feature definitions and the location of the pass and fail directories.

3 After reading in the pass and fail images, the program will create a set of thumbnails and display these in the side bar. When finished, the centre panel of the *Configuration* window should show the first fail image.

4 For this example, no changes will be made to the features so proceed directly to the *Training Window* by selecting the ***Window*** menu and choosing ***Training***. All of the data from the configuration window is automatically transferred to the training window.

5 From the ***Process*** menu, select ***Extract Data*** to process the images. A bar will show the progress of the data extraction process for the pass and fail image sets (depending on the computer speed, this may take some time). When complete, click ***OK*** in response to the "Feature extraction complete" message.

6 From the ***Process*** menu select ***Divide Data Set***. Accept the default values by clicking ***OK***.

7 From the ***Process*** menu, select ***Train Method*** to begin training. A series of messages and a status bar will appear during the training process. Click ***OK*** when the checking error is displayed. A plot of the data should be shown in the main window.

8 Click on the ***FIS Structure*** tab to see the network. Click on the ***Member FNs*** tab to return to see the membership functions formed during training. Click on the ***Feature Data*** tab to see the data distribution again.

9 From the ***Process*** menu, select ***Optimize Solution***. Click ***OK*** to accept the default "Error tolerance" of 0.0. A bar will appear indicating the progress of the optimization process. Click ***OK*** in response to the "x features were used to reach error of y" message.

10 Click on the ***Testing*** tab to see the results with full FIS structure. The RMS error for this example is typically on the order of 0.1, with 1 or 2 false positives or negatives.

Note The exact error and number of false negatives/positives may vary when this example is run. This is due to a randomization that occurs during the *Divide Data Set* process in Step 6 (see Section B.4.3 for details)

11 Click on ***Optimizations*** tab and select the second row (***Run Number 2***). After the network updates, click ***OK*** in response to the "FIS structure updated" message. Click on ***FIS Structure*** tab to see the simplified FIS structure.

12 Click on ***Testing*** tab to compare the results with the simplified FIS structure to the previous values found in Step 10. Performance may improve or degrade, but network has been simplified.

13 The network is now ready to be used to classify new image. From the ***Window*** menu, select ***Camera*** and the ***Camera*** should appear.

14 Click on the ***Open Camera*** button to open the link and a camera image should appear.

Note See section B.5.7 for instructions on testing the network with saved images if no camera or test setup is available.

15 Check ***Enable Processing*** and ***Display Graphics***. Feature icons should appear on the image together with a "pass" or "fail" decision, *FIS Output* and *Confidence* numbers in the left-hand panel.

16 Click ***Single Image*** to refresh the image and get a new decision.

17 Click on ***Online***, to get continuous stream of images and decisions (updates about every 1 *sec* depending upon computer), then select ***Offline*** when finished.

B.3 Configuration Window

The configuration window is used to set up the inspection problem by defining features and loading the example images.

B.3.1 File Menu

New Project...
Creates a blank project file and clears the data from the current screen. A prompt will appear requesting a location for the new project file. This action will erase all unsaved data in any of the windows.

Note A single project file is used for all data displayed by the three windows (configuration, training, and camera). Saving, opening, or creating new project files affects all three windows even if they are not currently open.

Open Project...
Opens a saved project file. The project files use the standard .mat Matlab® format and extension. Opening a project file erases all unsaved information in any of the windows.

Save Project
Saves the current project file (if it exists). No prompt is given and overwrite permission is assumed. If the project has not yet been saved, a dialog opens requesting a save location.

Save Project As...
Resaves the project file under a new name. This is useful for saving stages of a project in case you later want to revert to an older instance.

Open Pass Set... and **Open Fail Set...**
Used to select the directory locations for the pass and fail images. The selected directory is scanned to determine the predominate type of image file present and then these images are loaded (.bmp, .png, and .jpg are acceptable file types).

Note While it is possible to have other non-image files in the same directory as the pass or fail images, this is not recommended.

The algorithm will create a thumbnail image during import which will be saved for later quick access. The thumbnail file is always called `thumbnails.jpeg` and is saved back to the same directory.

Close
Deletes all data and closes the window. Any unsaved information from all three windows will be lost.

B.3.2 Feature Menu

The feature menu is used to add features to the inspection problem. Each feature is designed to detect a specific type of pattern in the image.

Line
Line features detect regions of high gradient in the image. High gradient areas are typically associated with edges, changes in surface, and corners in the part. To define the search area for a line, the search box needs to be defined by specifying the region height, width, and angle. Since a line may change direction in the image, a range of allowed angles can also be specified.

The line feature returns three attribute values: the angle of the line, the perpendicular distance from the origin, and the score. The score is based on how strong the gradient is with the strongest being an instantaneous change from black to white.

Circle
Circles are detected by looking for regions of high gradient that are aligned in a circular pattern. The circle detection algorithm is able to detect radius and centre point of the circle. The search region is determined by a box width and height, which define the possible centre-point locations for the circle.

The circle feature outputs four attributes: Centre-point X, Centre-point Y, Radius, and Score. Again, the score is based on the strength of the gradient for the detected circle.

Hole
For faster searching, hole features can be used instead of circles. A hole feature is a circle with a radius that is known. The algorithm will locate the centre-point of the strongest circle with the specified radius.

The hole feature outputs the same four attributes as the Circle feature, but the radius attribute does not change. It is merely a copy of the requested radius.

Colour
For some problems, colour is an important factor in recognizing objects. In the algorithm, colour is treated in the Hue, Saturation, Value colour space. A colour feature is a region with enough Saturation to be considered a colour. The hue

space is then divided into several sections and the hue with the largest number of pixels represented is reported along with the centroid of the pixels. Hue search regions are defined by width and height. The number of divisions for the Hue is specified, and the target Hue can also be specified. If the Hue with the largest region is desired, then the target Hue is set to zero.

Colour features output four attributes: Centroid X, Centroid Y, Strongest Hue, Area of pixels.

Squiggle

The squiggle feature detects a series of lines in each of the X and Y directions. This feature tends to work well in regions with fine detail that won't be represented well by broader features such as lines and circles. It can also be used to map the edge of an irregular part shape.

The Squiggle feature is specified by a width and height and the number of divisions in each direction. A polarity setting determines which direction of gradient is accepted (-1:white to black or 1:black to white) specifying a polarity of zero accepts both directions.

B.3.3 Fixture Menu

A group of features can be used to define a fixture. Fixtures use easy-to-find features in order to locate more precise regions. The nominal position of the included features is recorded. Changes from the nominal position are analyzed for other images, the average transformation is calculated, and any region based on that fixture is transformed.

Edit Fixtures

Opens the fixture editing window so fixtures can be added, removed, or edited (See Section B.3.8).

B.3.4 Window Menu

Transferring the data to other parts of the program can be accomplished using the Window menu. Selecting the appropriate item launches the window and refreshes the data within it.

B.3.5 Toolbox

The toolbox (Figure B.1) gives quick access to create new features and operates identically to the items in the Feature menu.

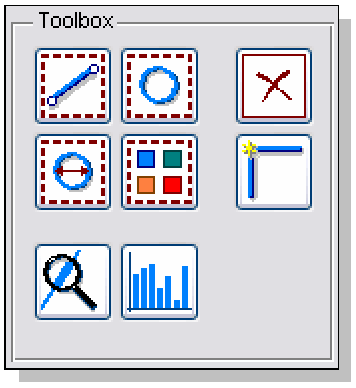


Figure B.1: The Configuration window toolbox

The toolbox contains four additional buttons: a delete feature button which removes the current feature, an auto-search button, a statistics button, and a define fixture button. The last three buttons invoke separate windows.

B.3.6 Statistics Window

The statistics window (Figure B.2) is used to view the range and distribution of the data from the training image set. By processing all of the images in the set, it is possible to quickly identify how well the feature attribute data is segmented and to see if there are any outliers in the data.

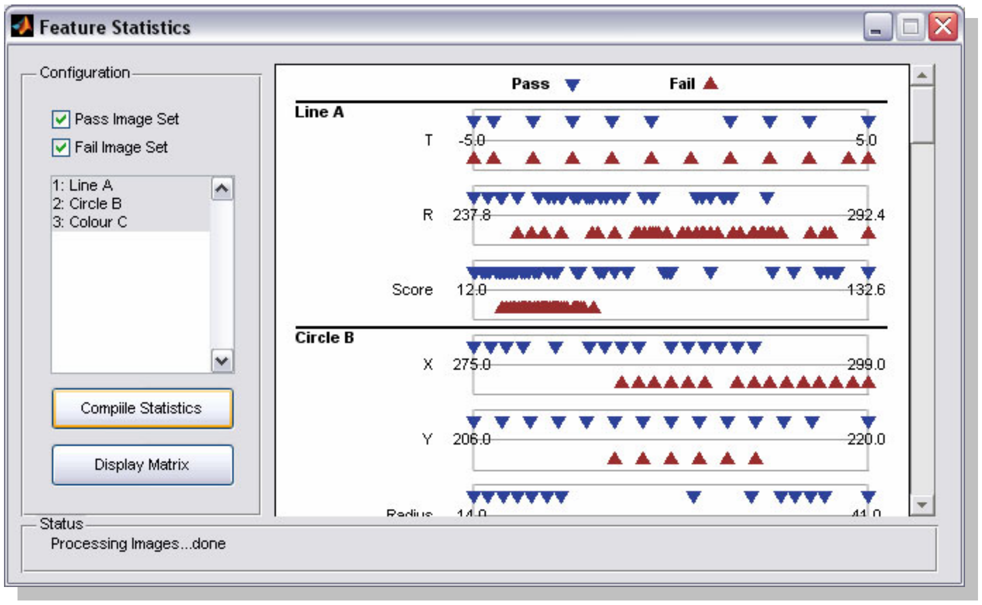


Figure B.2: The feature statistics screen.

Pass/Fail Check Boxes
If the pass and fail image sets are loaded, then both checkboxes at the top of the screen will be active. Choose the image set you want to use for processing (or both if you want to use all of the images).

Feature Selection
All of the features that are defined for the current inspection problem will be displayed in the list box. Selected features will be included in the feature extraction. By selecting fewer features, the processing time to extract feature data will be reduced. Use the SHIFT key to select multiple features.

Compile Statistics
Goes through the selected image set and attribute values associated with the features selected in the list box. A progress bar will indicate how many of the images have been processed. When finished, the extracted values will be shown in the display area.

Display Matrix
Writes the extracted feature attribute data to the main screen in order to make it possible to identify the output from a specific image. The plotted data can be used to identify which particular image is creating an outlier.

Note Sometimes, outliers are caused by unusual conditions in the image that can make feature extractors confused. However, it is also possible, for easier to fix problems to be caught such as a feature search region being too small, or a misclassified image being included in the training set.

Status Bar
Displays status information, warnings, and other information about the process.

Display Area
The data is plotted in the display area with the pass and fail data separated visually. The far left side lists the features by name along with the attributes for each feature. Attribute values from pass images are plotted as inverted blue triangles, and from fail images as upright red triangles. The scale for each attribute is indicated by the values appearing at either end of the bar.

B.3.7 Auto-search

The auto-search button instantly starts the search process based on the current image, image set, and feature. Auto-search works by assuming that the feature in the current image is correct and will begin to expand the region so a similar feature is found in the other images. After each expansion, the original image is tested to ensure that the same feature is being extracted. If not, region expansion is halted.

Auto-search works by forming key-points in the current image and then attempting to find similar key-points in the other images. By computing a mapping between the key-points around the search area in each image, a probable location for the feature in the new image is determined. The search region is similarly transformed by adding the new region to the existing one and a new expanded region is created.

Note The search region in the original image should be as small as possible as each transformation will expand the size of the region even if the feature can be found in the current space.

B.3.8 Fixture Window

The fixture editing window allows features or groups of features to be defined as fixtures. Other features can then be defined relative to the fixture so that they are automatically adjusted at the part moves (see Figure B.3).

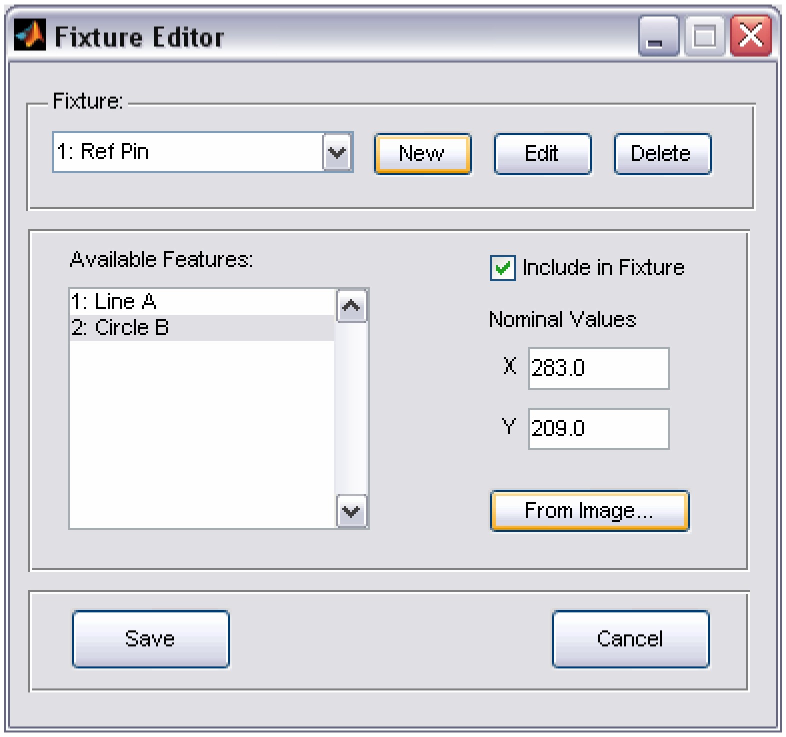


Figure B.3: The fixture editing screen.

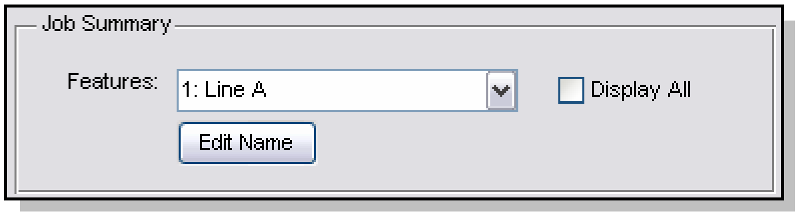
To create a new fixture, click the NEW button and enter the name. The name can be changed at any time by selecting EDIT. Currently defined features are listed in the scroll box on the left side of the screen. Selecting a feature will display the information for that feature on the right side of the screen. To add a feature to the fixture check the INCLUDE IN FIXTURE box and enter the nominal feature attribute values. These are the values that will be used to determine how far the feature has moved from the reference location.

To capture the reference information from an image click the FROM IMAGE... button and select an image file. The feature will be extracted from that image and entered as the nominal values.

To delete a fixture, click DELETE. When finished, click SAVE, or to revert to the state before the window was opened, click CANCEL.

B.3.9 Job Summary Area

Displays information about the current inspection job (see Figure B.4). The currently defined features are displayed in the drop-down box and selecting a specific feature will make that feature active.



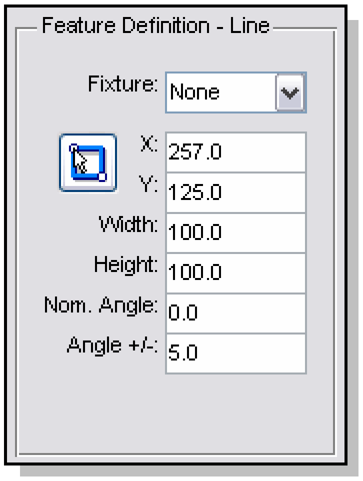
Features: 1: Line A

Figure B.4: The Job Summary portion of the Configuration window.

To change the name of a feature, use the EDIT NAME button and type the new description. Changing the feature parameters is accomplished using the FEATURE DEFINITION area on the sidebar. The DISPLAY ALL check box shows the graphics for all features currently defined.

B.3.10 Feature Definition Area

The feature definition panel shows the extraction settings for the active feature (see Figure B.5)



Fixture: None
X: 257.0
Y: 125.0
Width: 100.0
Height: 100.0
Nom. Angle: 0.0
Angle +/-: 5.0

Figure B.5: The feature definition panel.

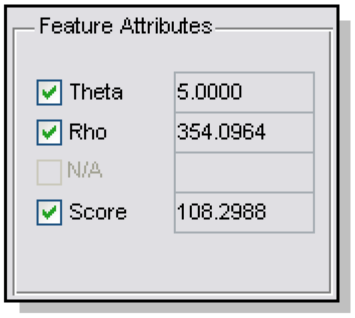
The contents of the panel will change depending on the type of feature as each feature has different extraction parameters. In general, the first four to six lines will refer to the area of interest specification, and the remainder are specific to each feature. Clicking the "graphical select tool" (cursor inside a blue square) will allow interactive selection of the area of interest.

If any fixtures are defined, they can be selected by choosing a fixture from the drop-down list. The feature search region will then be adjusted according to the fixture position.

Note A feature cannot be associated with a fixture of which it is a part.

B.3.11 Feature Attributes Area

The feature attributes panel shows the active outputs for the selected feature. The available attributes will change depending on the feature selected (see Figure B.6).



✓ Theta	5.0000
✓ Rho	354.0964
N/A	
✓ Score	108.2988

Figure B.6: The feature attributes panel.

When a feature is selected, and there is a valid image in the main window, the values of the feature attributes will also be shown alongside the attribute names. Deselecting the checkboxes means that attribute data will not be included in subsequent processing. The default is for all attributes to be selected. Some entries may be disabled if the current feature has fewer than four attributes.

B.3.12 Image Set Area

The image set area displays thumbnails of the currently loaded image sets (see Figure B.7) along with the image number and image file name.

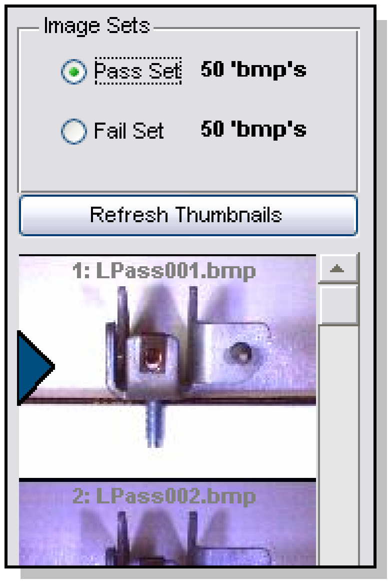


Figure B.7: The image set area of the Configuration screen.

When the set is loaded for the first time, a thumbnail image is created and displayed in the thumbnail window. If the images in the directory change, the thumbnails can be updated using the REFRESH THUMBNAILS button.

If both image sets are loaded, the currently displayed set can be selected using the radio buttons above the thumbnails. Using the scroll bar will access images beyond the bottom of the thumbnail area. Clicking on an image loads it from disk and displays it in the main window. The blue arrow indicates the current image.

B.4 Training Window

The training window is used to develop the classification system that will identify pass and fail images.

B.4.1 File Menu

All items in the file menu operate identically to the File menu on the Configuration screen (see Section B.3.1).

B.4.2 Method Menu

The Method menu is used to select which classifier should be used. Both classifiers can be defined, but only one can be used and viewed at a time.

Neuro-Fuzzy

Selects the neuro-fuzzy classifier and updates the screen to show the data from the neuro-fuzzy system. All tabs will change to display the data associated with the current neuro-fuzzy system.

Threshold

Selects the threshold classifier and updates all screens and tabs to display the threshold-based system data.

B.4.3 Process Menu

The process menu gives access to the main functions of this window. The development of the classifier has been broken into four steps for clarity, but the steps must be performed in order.

Extract Data

The extract data option takes the defined features and extracts the feature attribute data from the images defined in the pass and fail sets. A warning will be issued if these sets have not been loaded. When extraction is complete, the data will be plotted in the main window in a manner similar to the statistics window (see Figure B.2).

Divide Data Set

Takes the feature attribute information extracted in the previous step and randomly divides it into three parts: training, checking, and testing. The user is given the option of deciding the percentage of the dataset allocated to each part.

The default is 40/20/40. Division is based on the smaller of the two data sets. If one set is larger, then the extra data will go into the testing set. For the training and checking sets, the number of pass and fail images will always be balanced. After dividing the set, the training portion is randomly sorted in order to work better for training the ANFIS network.

Note The division breakdown is entered by selecting three percentage values. If the total of these three do not equal 100, then the default values will be used instead.

Train Method

If the neuro-fuzzy method is active, then the feature attribute values are used along with the associated classification to develop a neuro-fuzzy structure and train it to produce the best classification accuracy possible (in terms of RMS error on the training set). When complete, the membership functions will be displayed.

If the threshold method is active, then each attribute will be divided by thresholds using the automatic algorithm. This algorithm searches for continuous groups of pass or fail data and separates them by thresholds. The user inputs the minimum size of group to qualify both in number of points included, and in span compared to the total data range. The resulting groups are displayed when the process is complete using shading underneath the data plots.

Optimize Solution

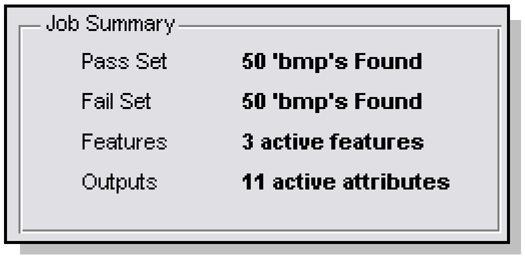
Attempts to determine the best feature attribute combination for the current inspection task. The user is asked for the checking error at which to stop optimization. The default value is 0 which will continue optimization until no further improvement is possible. The results are displayed in tabular form when complete.

B.4.4 Window Menu

Identical to the Configuration window (see Section B.3.4).

B.4.5 Job Summary Area

The job summary area (see Figure B.8) shows relevant information about the currently loaded data.



Job Summary

Pass Set	50 'bmp's Found
Fail Set	50 'bmp's Found
Features	3 active features
Outputs	11 active attributes

Figure B.8: The Job Summary area for the Training window.

If the Training window was opened from another window, then any data that was defined in the previous window will be loaded and displayed. The number of features and attributes are based on the total number defined and do not change when different optimizations are selected.

B.4.6 Feature Data Tab

The feature data tab shows a plot of the feature attribute data in the main window. The feature names are shown alongside the pass and fail data (see Figure B.9). Using the scroll bar will show any attributes which appear off the bottom of the page.

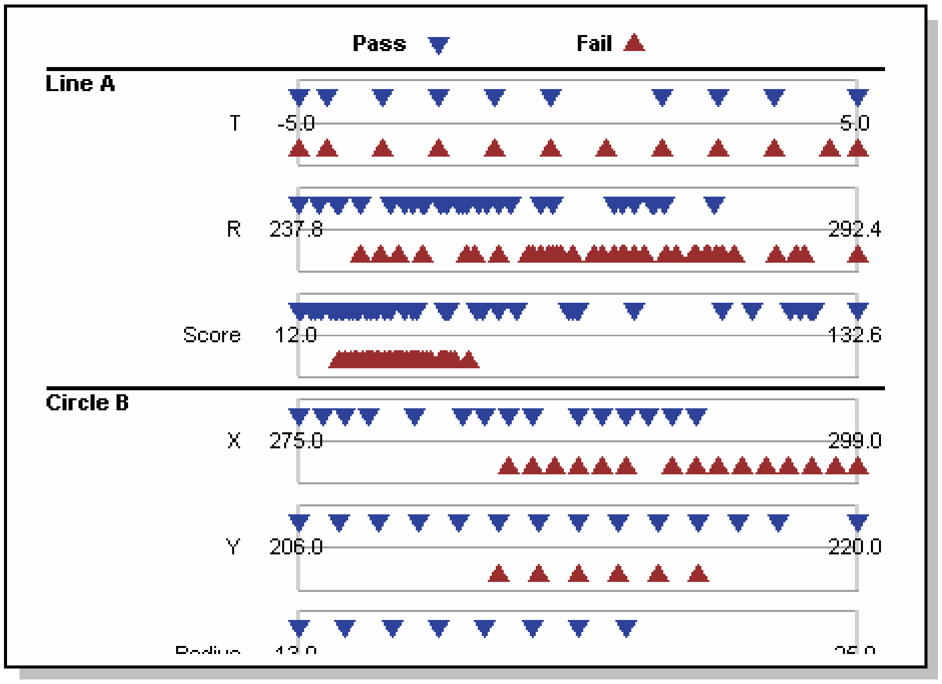


Figure B.9: The feature data plot in the Training window.

Note For all tabs, only the feature attributes for the currently selected optimization are shown. If an optimization is selected with only one feature attribute, then only one plot will appear when the Feature Data tab is selected. To see all of the plots again, the default optimization must be selected (all attributes active).

B.4.7 Membership FNs/Groupings Tab

If the neuro-fuzzy method is selected, then this tab shows the membership functions plotted against the data for that attribute (see Figure B.10). Different attributes can be accessed using the scroll bar on the side of the window. The dimension number and attribute name are shown in the top left corner for reference.

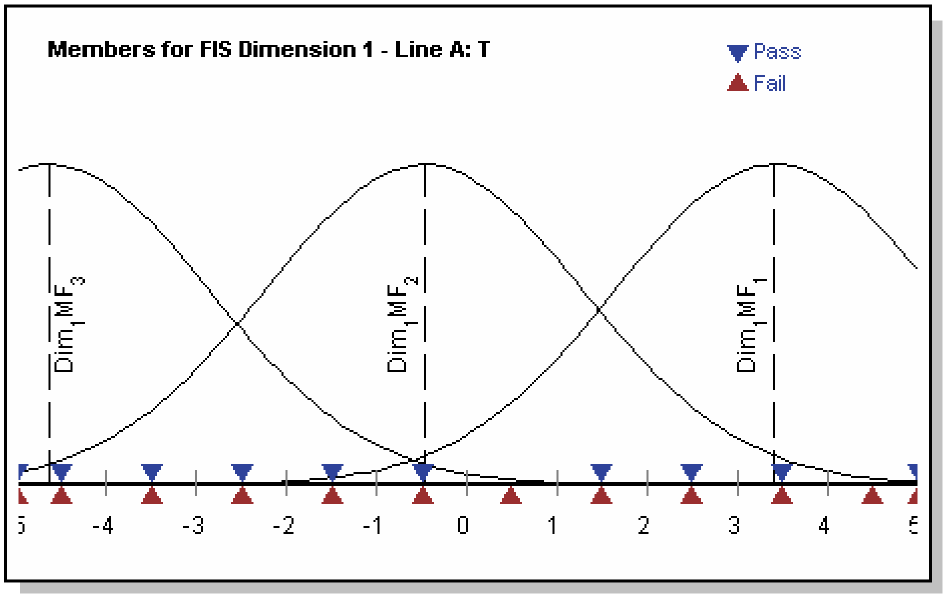


Figure B.10: The membership function plot in the Training window

If the threshold method is selected, then this tab shows a plot similar to the Feature Data tab but with shading added to show the results of the threshold training process (see Figure B.11). Regions shaded blue have been set up as pass regions and red regions as fail. Feature attributes which are shaded grey are too evenly distributed to form any thresholds. Using the scroll bar will show any attributes which appear off the bottom of the page.

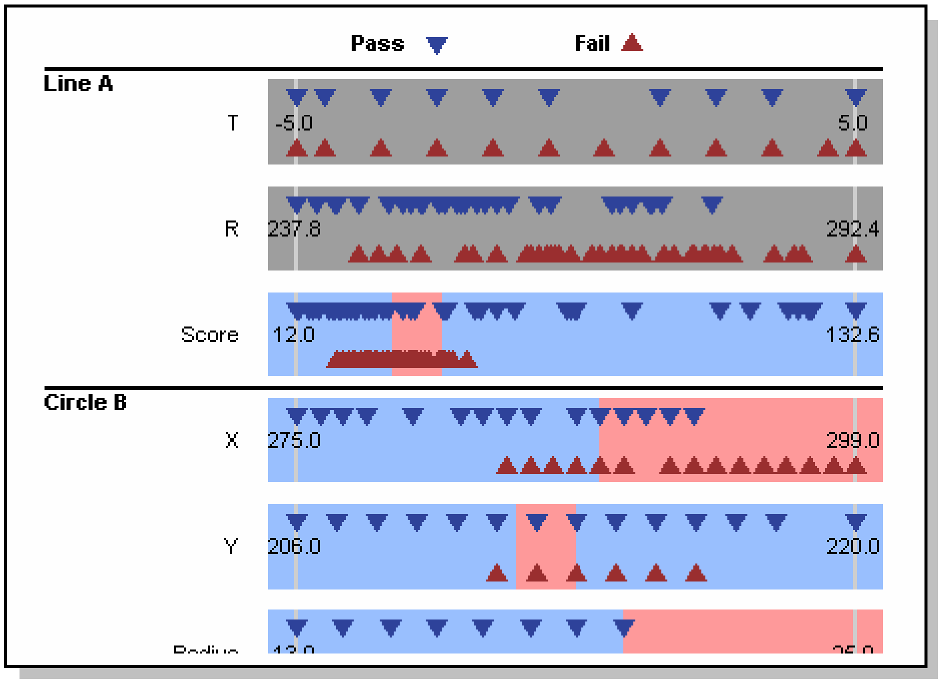


Figure B.11: The threshold grouping plot in the Training window

B.4.8 FIS Structure Tab

This tab is only active when the neuro-fuzzy method is selected. It displays the nodes and links for the current neuro-fuzzy network (see Figure B.12).

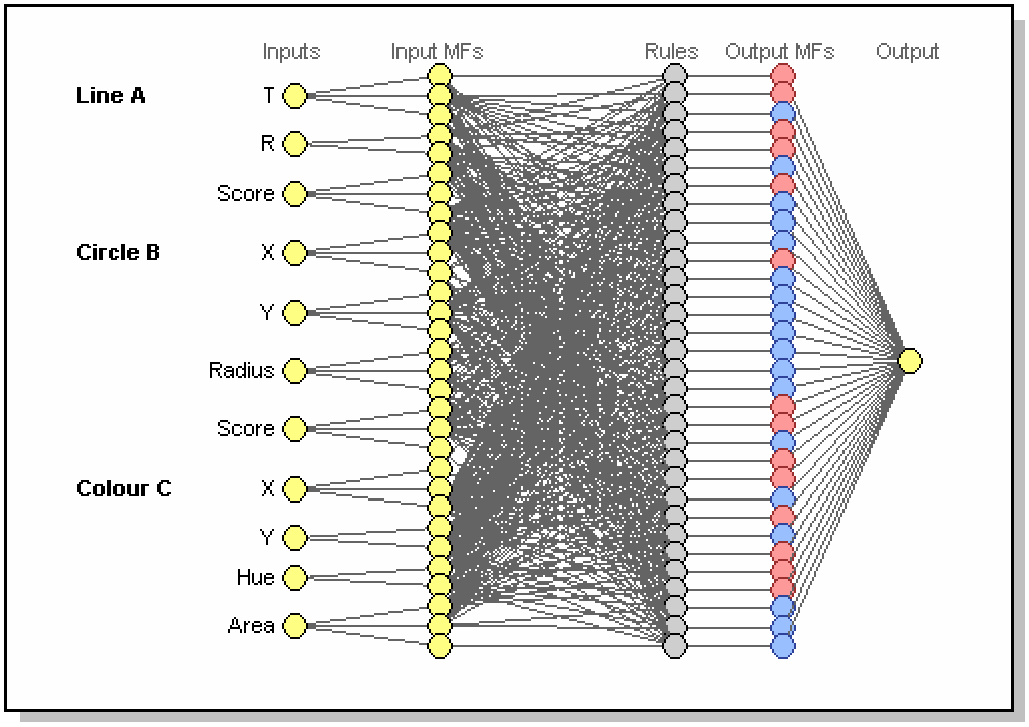
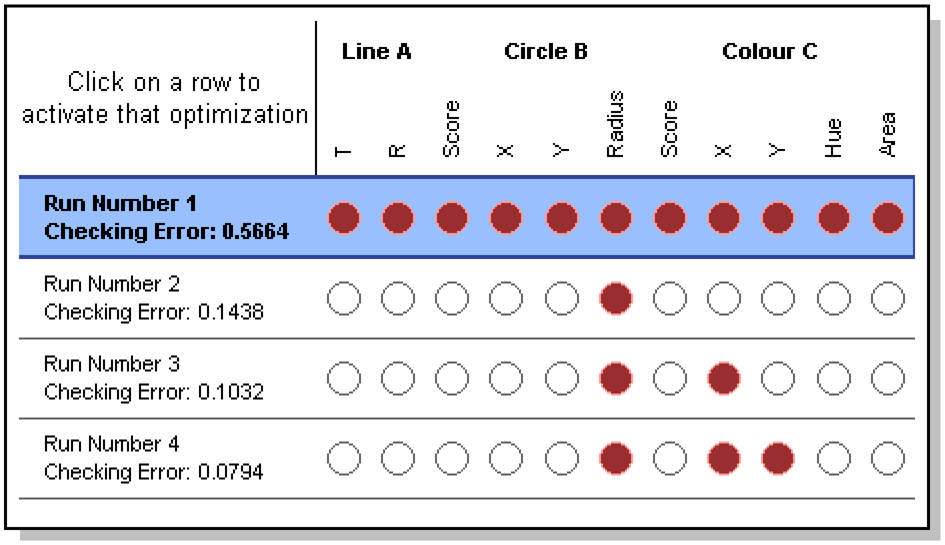


Figure B.12: The neuro-fuzzy structure display in the Training window.

The inputs (yellow) are on the left and are connected to the input membership function nodes. These are then connected to the rules (grey). Each rule is associated with an output node which is coloured either blue or red depending if its value is above or below 0.5 respectively. This gives some indication of which rules are contributing to a pass classification, and which rules are contributing to a fail classification.

B.4.9 Optimizations Tab

After the optimization has been performed, this tab shows the results in tabular form (see Figure B.13).



Click on a row to activate that optimization	Line A			Circle B				Colour C			
	T	R	Score	X	Y	Radius	Score	X	Y	Hue	Area
Run Number 1 Checking Error: 0.5664	●	●	●	●	●	●	●	●	●	●	●
Run Number 2 Checking Error: 0.1438	○	○	○	○	○	●	○	○	○	○	○
Run Number 3 Checking Error: 0.1032	○	○	○	○	○	●	○	●	○	○	○
Run Number 4 Checking Error: 0.0794	○	○	○	○	○	●	○	●	●	○	○

Figure B.13: The optimization display in the Training window.

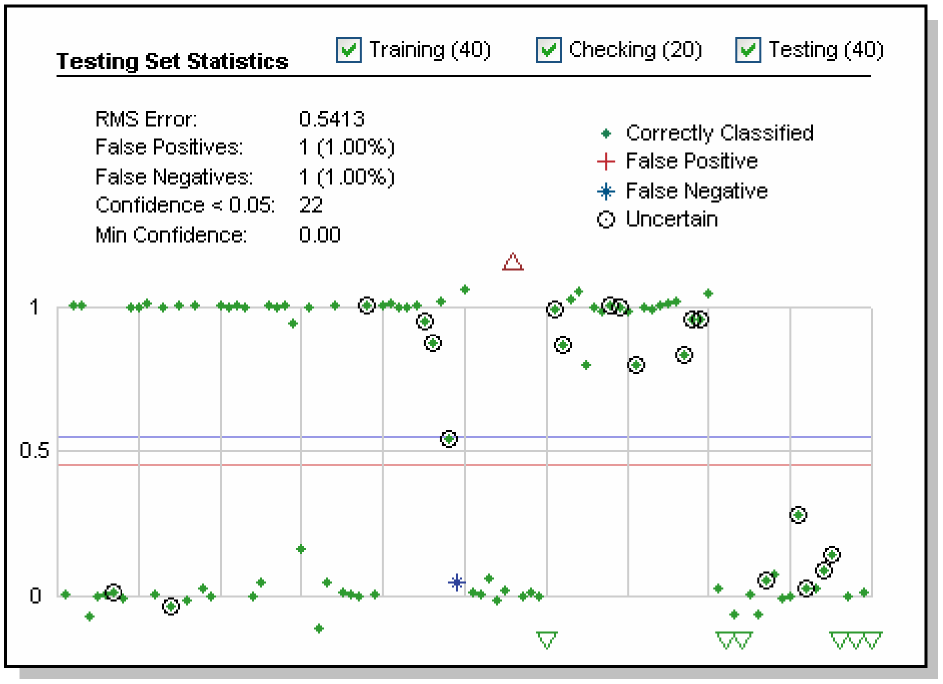
Each row of the table represents a particular optimization run. The RMS error on the checking data is displayed in the left hand column under the run number. Each of the columns on the right side corresponds to one of the feature attributes as indicated by the labels above the column. If an attribute has a red dot for a particular run, then it was included in that optimization.

Clicking on one of the rows will configure all of the other tabs to show data related to that optimization. The selected run will be highlighted in blue.

Note When the neuro-fuzzy method is selected, clicking on an optimization row rebuilds the network to match the attributes selected. This may take a moment to complete.

B.4.10 Testing Tab

The testing screen shows the results from using the current method to classify the data in the data set (see Figure B.14).



RMS Error: 0.5413
False Positives: 1 (1.00%)
False Negatives: 1 (1.00%)
Confidence < 0.05: 22
Min Confidence: 0.00

Figure B.14: The testing screen plot in the Training window.

The check boxes along the top of the screen can be used to turn the display on or off for each portion of the data set (Training, Checking, and Testing).

Note The results are plotted in the order they appear in the data set. The training data set gets sorted randomly before training so pass and fail images will appear in random order on the plot. The checking and testing datasets are always sorted with the pass data first.

Each point is identified as either being correctly classified (green dot), a false positive (red cross), or a false negative (blue star). Points classified as uncertain either because of a low confidence output from the neuro-fuzzy network, or because the fall too close to 0.5 will be circled. The boundaries for the uncertain region around 0.5 are shown by the blue and red horizontal lines.

Points which fall outside the range of the plot are indicated by upright or inverted triangles on the upper or lower boundaries. The colour of these arrows indicates their state according to the colours in the legend.

The top left corner of the screen gives a summary of the results for the data shown on the plot. For the neuro-fuzzy method, the entry "Confidence < #.##" records the number of uncertain values.

Note To change minimum confidence that must be achieved for a point not to be marked uncertain and the boundaries around 0.5, the code must be altered. See the section on the plotTestResults function in Appendix A for details.

In order to identify which image is causing a particular false negative or false positive, open the main Matlab window. There will be a text listing of the misclassified images which gives the pass or fail set number. This matches the number displayed in the Configuration window.

B.5 Camera Window

The camera window is used to view the output of the classifier for new live camera or stored images. It can also be used to update the network with new image information.

B.5.1 File Menu

The project related commands and the CLOSE command on the File menu are identical to those in the Training and Configuration windows (see Section B.3.1).

Load Test Set...
Allows a directory to be selected which contains example images. These images will be displayed as thumbnails on the right side of the screen and can be selected to simulate camera images. The process by which images are located in the selected directory is identical to the method used when opening pass or fail image sets (see Section B.3.1).

B.5.2 Processing Menu

Process Test Set
Reserved for future use. This command would classify all of the images loaded in the test set and report the results in a method similar to the testing tab of the Training screen.

Show Test Results
Reserved for future use. Would display the results from the most recent run of "Process Test Set".

Select Processor
Allows the user to select between the neuro-fuzzy and threshold-based classifiers (and algorithms from other schools if implemented later). This is a global setting so it will initially match the selection made in the Training window. When a new processor is selected, the Inspection area will update to reflect the output of the current method.

B.5.3 Window Menu

Identical to the window menus in the Training and Configuration windows (see Section B.3.4)

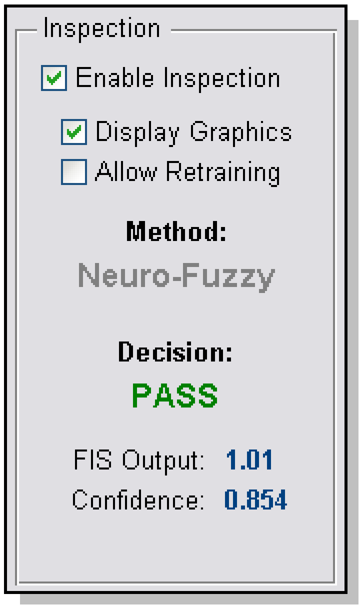
B.5.4 File Summary Area

The file summary area shows basic information about the current project. The first line gives the number of feature attributes that are active, and the second line gives the details of the neuro-fuzzy structure.

Note If the number of feature attributes does not match the number of inputs to the neuro-fuzzy structure, processing of the images will not be possible.

B.5.5 Inspection Area

When ENABLE INSPECTION is checked, the inspection area shows the results of running the current classifier on the image in the main window (see Figure B.15).



Method:
Neuro-Fuzzy

Decision:
PASS

FIS Output: 1.01
Confidence: 0.854

Figure B.15: The inspection area of the Camera window

Checking DISPLAY GRAPHICS will show the feature extraction regions and the extracted features. A legend in the top right corner identifies the feature type and number.

Checking ALLOW RETRAINING will ask the operator to enter the correct classification whenever an image is encountered that has been classified as uncertain. This information will then be used to update the neuro-fuzzy network.

Note Network updates occur to the active network structure only. If the Training window is re-opened the active network may be updated from the full network structure and changes could be lost. Any changes due to retraining should be saved to a new project file.

The Method section of the display indicates the active classification system (neuro-fuzzy or threshold for now). And the decision output of that classifier is shown under "Decision". The three possible decisions are Pass, Fail, and Uncertain. Details of the classification are given by the decimal output of the classifier "FIS Output", and the confidence of the decision (based on the rule firing strengths within the network).

B.5.6 Camera Area

The camera panel is used to connect to an external camera DCAM compliant camera (see Figure B.16).

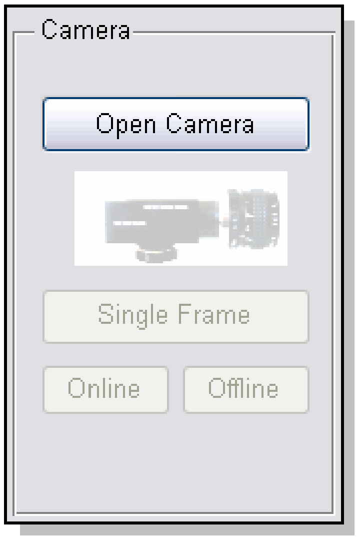


Figure B.16: The camera control panel of the Camera window

The camera must be connected and powered before selecting OPEN CAMERA. An error message will appear if the camera is not found. For more details about camera set up see the entry for OpenCamera.m in Appendix A.

Once the camera is opened, the Open Camera button will change to Close Camera, the image of the camera will become solid, and the other three buttons will be activated.

The SINGLE FRAME button captures one frame from the camera and displays it in the main window. If inspection is enabled, then the image will be classified, and the results displayed in the Inspection panel.

ONLINE and OFFLINE toggle the camera in and out of continuous capture mode. In continuous mode images are captured and displayed to the main window as fast as the software can handle. If a smoother image is desired (for example for doing the initial set up), then the inspection can be disabled.

Note Always go OFFLINE before closing the camera. Failure to do this may result in the camera locking up.

Show Reference **Button**

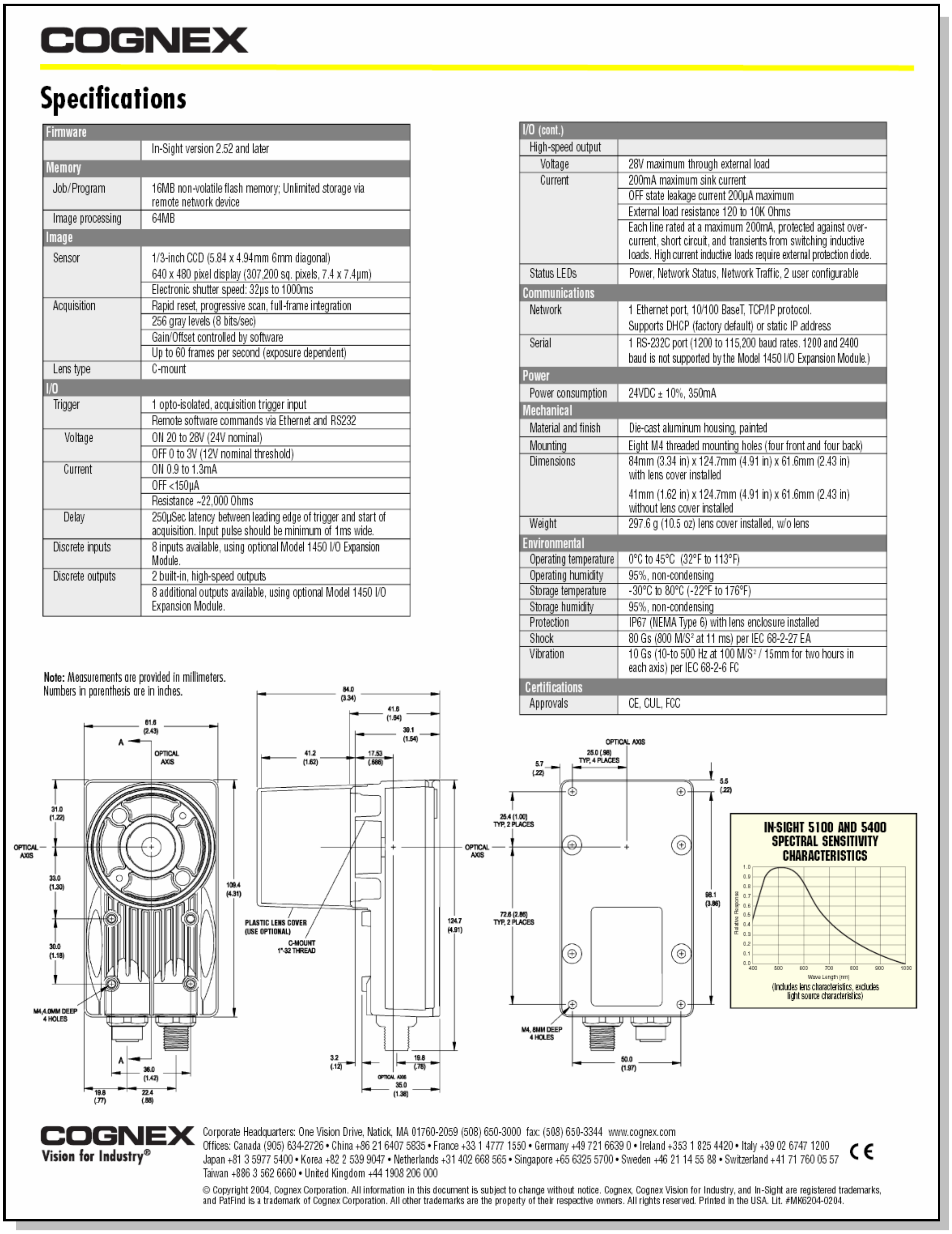
When setting up the system with a new camera, or camera location, it is useful to be able to have some reference for how the images should look compared to the training images. Clicking the Show Reference button (located just above the main window) allows the user to select a reference image. The outlines of this image are then overlaid onto the camera image it can be aligned. Clicking the button again removes the outline.

B.5.7 Image Set Area

The image set area shows a list of thumbnails from the current Test Set. The list operates in a similar manner to the thumbnail set in the Configuration window. Clicking an image will load it into the main window and perform inspection (if enabled).

Appendix C

Camera Specifications



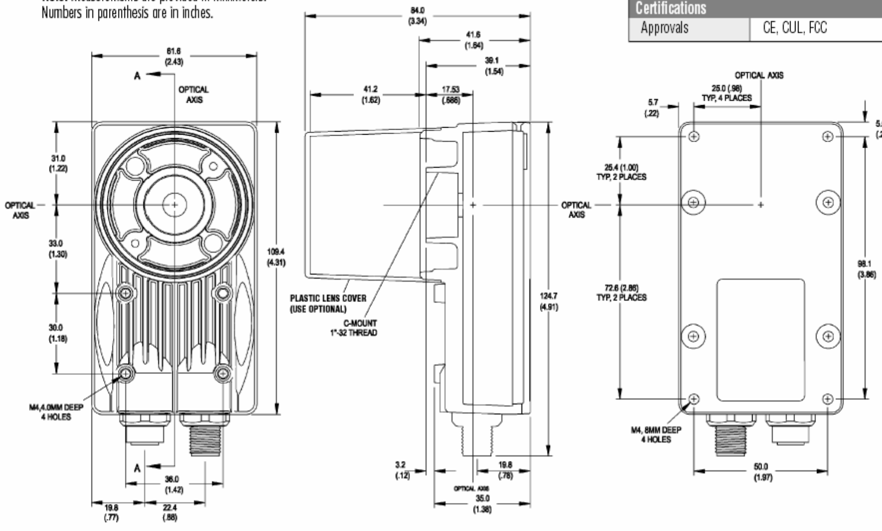
COGNEX

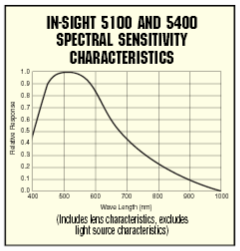
Specifications

Firmware	
	In-Sight version 2.52 and later
Memory	
Job/Program	16MB non-volatile flash memory; Unlimited storage via remote network device
Image processing	64MB
Image	
Sensor	1/3-inch CCD (5.84 x 4.94mm 6mm diagonal)
	640 x 480 pixel display (307,200 sq. pixels, 7.4 x 7.4µm)
	Electronic shutter speed: 32µs to 1000ms
Acquisition	Rapid reset, progressive scan, full-frame integration
	256 gray levels (8 bits/sec)
	Gain/Offset controlled by software
	Up to 60 frames per second (exposure dependent)
Lens type	C-mount
I/O	
Trigger	1 opto-isolated, acquisition trigger input
	Remote software commands via Ethernet and RS232
Voltage	ON 20 to 28V (24V nominal)
	OFF 0 to 3V (12V nominal threshold)
Current	ON 0.9 to 1.3mA
	OFF <150µA
	Resistance ~22,000 Ohms
Delay	250µSec latency between leading edge of trigger and start of acquisition. Input pulse should be minimum of 1ms wide.
Discrete inputs	8 inputs available, using optional Model 1450 I/O Expansion Module.
Discrete outputs	2 built-in, high-speed outputs
	8 additional outputs available, using optional Model 1450 I/O Expansion Module.

I/O (cont.)	
High-speed output	
Voltage	28V maximum through external load
Current	200mA maximum sink current
	OFF state leakage current 200µA maximum
	External load resistance 120 to 10K Ohms
	Each line rated at a maximum 200mA, protected against over-current, short circuit, and transients from switching inductive loads. High current inductive loads require external protection diode.
Status LEDs	Power, Network Status, Network Traffic, 2 user configurable
Communications	
Network	1 Ethernet port, 10/100 BaseT, TCP/IP protocol. Supports DHCP (factory default) or static IP address
Serial	1 RS-232C port (1200 to 115,200 baud rates. 1200 and 2400 baud is not supported by the Model 1450 I/O Expansion Module.)
Power	
Power consumption	24VDC ± 10%, 350mA
Mechanical	
Material and finish	Die-cast aluminum housing, painted
Mounting	Eight M4 threaded mounting holes (four front and four back)
Dimensions	84mm (3.34 in) x 124.7mm (4.91 in) x 61.6mm (2.43 in) with lens cover installed 41mm (1.62 in) x 124.7mm (4.91 in) x 61.6mm (2.43 in) without lens cover installed
Weight	297.6 g (10.5 oz) lens cover installed, w/o lens
Environmental	
Operating temperature	0°C to 45°C (32°F to 113°F)
Operating humidity	95%, non-condensing
Storage temperature	-30°C to 80°C (-22°F to 176°F)
Storage humidity	95%, non-condensing
Protection	IP67 (NEMA Type 6) with lens enclosure installed
Shock	80 Gs (800 M/S^2 at 11 ms) per IEC 68-2-27 EA
Vibration	10 Gs (10-to 500 Hz at 100 M/S^2 / 15mm for two hours in each axis) per IEC 68-2-6 FC
Certifications	
Approvals	CE, CUL, FCC

Note: Measurements are provided in millimeters. Numbers in parenthesis are in inches.



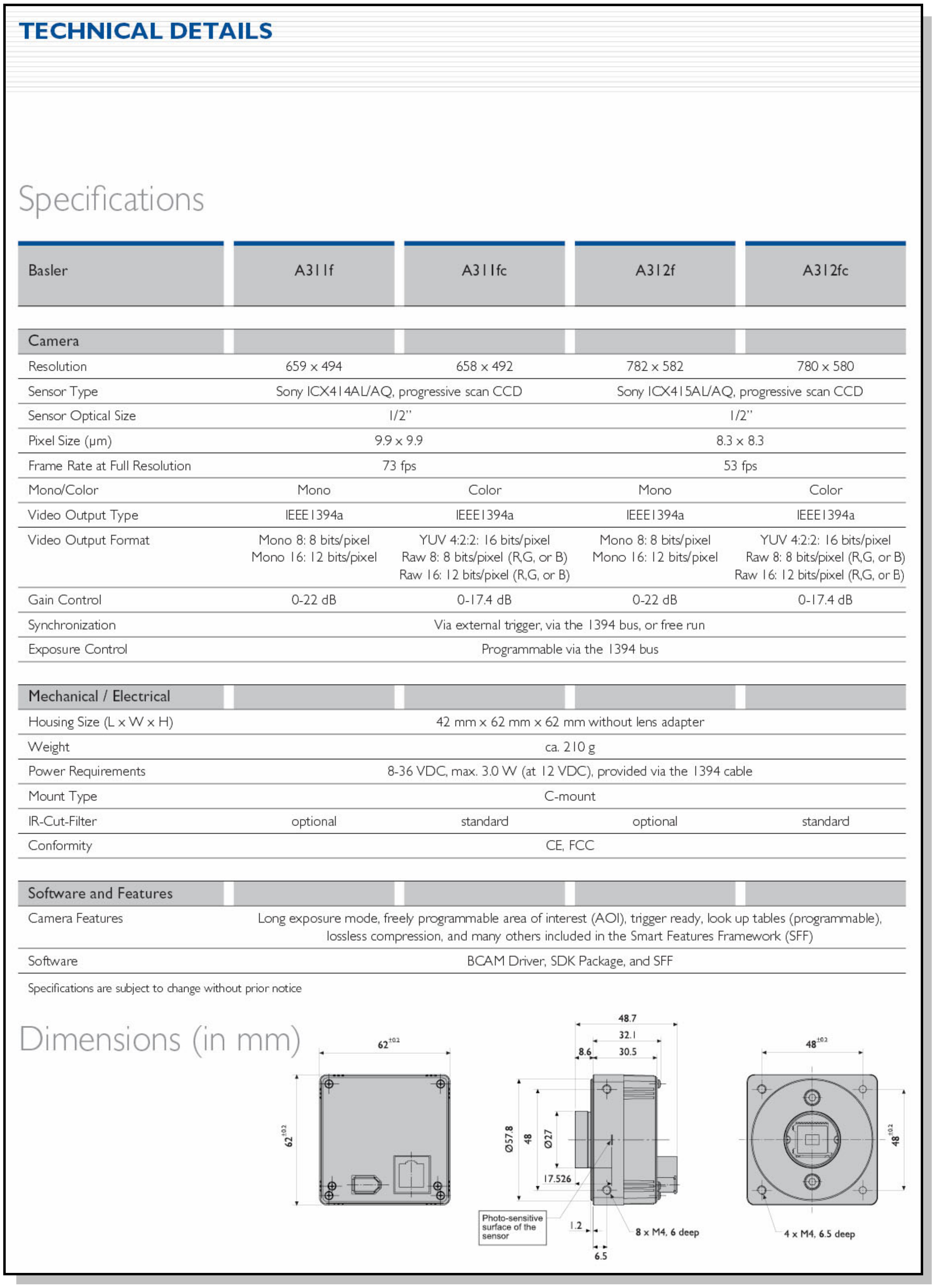


COGNEX Vision for Industry®

Corporate Headquarters: One Vision Drive, Natick, MA 01760-2059 (508) 650-3000 fax: (508) 650-3344 www.cognex.com
Offices: Canada (905) 634-2726 • China +86 21 6407 5835 • France +33 1 4777 1550 • Germany +49 721 6639 0 • Ireland +353 1 825 4420 • Italy +39 02 6747 1200 Japan +81 3 5977 5400 • Korea +82 2 539 9047 • Netherlands +31 402 668 565 • Singapore +65 6325 5700 • Sweden +46 21 14 55 88 • Switzerland +41 71 760 05 57 Taiwan +886 3 562 6660 • United Kingdom +44 1908 206 000

CE

Figure C.1: Specification sheet for the Cognex® 5400C camera in use at Van Rob (Sure, 2007)



TECHNICAL DETAILS

Specifications

Basler	A311f	A311fc	A312f	A312fc
Camera				
Resolution	659 x 494	658 x 492	782 x 582	780 x 580
Sensor Type	Sony ICX414AL/AQ, progressive scan CCD		Sony ICX415AL/AQ, progressive scan CCD	
Sensor Optical Size	1/2"		1/2"	
Pixel Size (μm)	9.9 x 9.9		8.3 x 8.3	
Frame Rate at Full Resolution	73 fps		53 fps	
Mono/Color	Mono	Color	Mono	Color
Video Output Type	IEEE1394a	IEEE1394a	IEEE1394a	IEEE1394a
Video Output Format	Mono 8: 8 bits/pixel Mono 16: 12 bits/pixel	YUV 4:2:2: 16 bits/pixel Raw 8: 8 bits/pixel (R,G, or B) Raw 16: 12 bits/pixel (R,G, or B)	Mono 8: 8 bits/pixel Mono 16: 12 bits/pixel	YUV 4:2:2: 16 bits/pixel Raw 8: 8 bits/pixel (R,G, or B) Raw 16: 12 bits/pixel (R,G, or B)
Gain Control	0-22 dB	0-17.4 dB	0-22 dB	0-17.4 dB
Synchronization	Via external trigger, via the 1394 bus, or free run			
Exposure Control	Programmable via the 1394 bus			
Mechanical / Electrical				
Housing Size (L x W x H)	42 mm x 62 mm x 62 mm without lens adapter			
Weight	ca. 210 g			
Power Requirements	8-36 VDC, max. 3.0 W (at 12 VDC), provided via the 1394 cable			
Mount Type	C-mount			
IR-Cut-Filter	optional	standard	optional	standard
Conformity	CE, FCC			
Software and Features				
Camera Features	Long exposure mode, freely programmable area of interest (AOI), trigger ready, look up tables (programmable), lossless compression, and many others included in the Smart Features Framework (SFF)			
Software	BCAM Driver, SDK Package, and SFF			

Specifications are subject to change without prior notice

Dimensions (in mm)

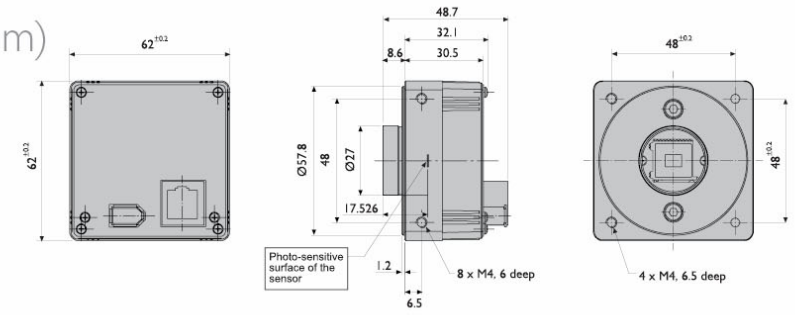
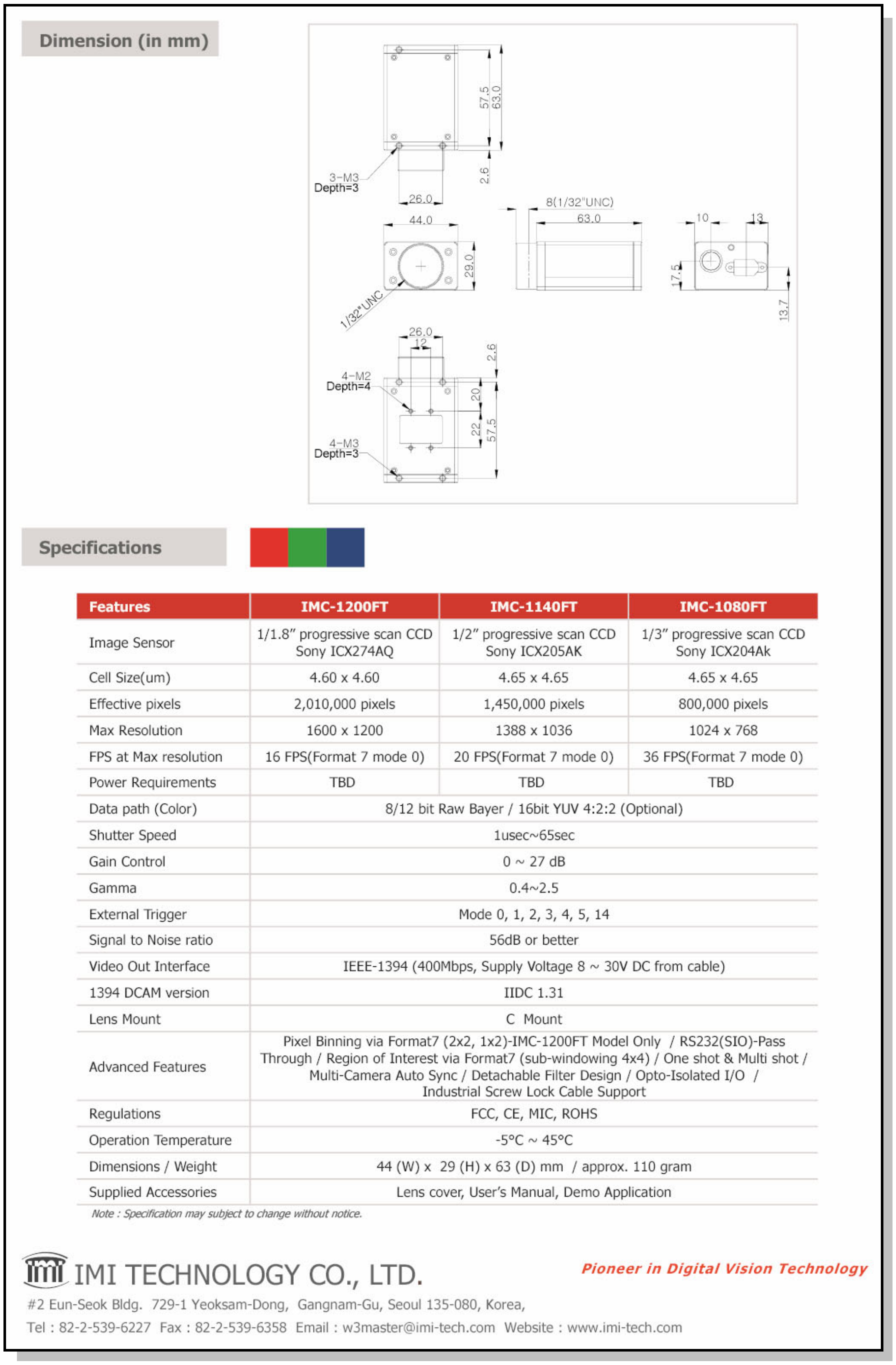
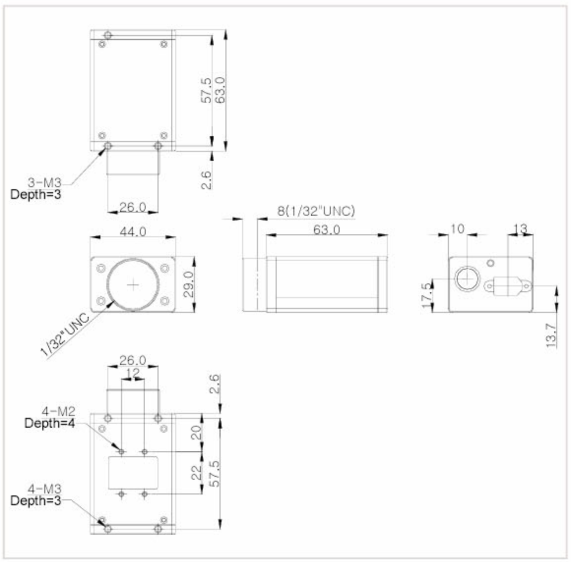


Figure C.2: Specification sheet for the Basler 312fc Camera (Basler, 2007)



Dimension (in mm)



Specifications 🟥🟩🟦

Features	IMC-1200FT	IMC-1140FT	IMC-1080FT
Image Sensor	1/1.8" progressive scan CCD Sony ICX274AQ	1/2" progressive scan CCD Sony ICX205AK	1/3" progressive scan CCD Sony ICX204Ak
Cell Size(um)	4.60 x 4.60	4.65 x 4.65	4.65 x 4.65
Effective pixels	2,010,000 pixels	1,450,000 pixels	800,000 pixels
Max Resolution	1600 x 1200	1388 x 1036	1024 x 768
FPS at Max resolution	16 FPS(Format 7 mode 0)	20 FPS(Format 7 mode 0)	36 FPS(Format 7 mode 0)
Power Requirements	TBD	TBD	TBD
Data path (Color)	8/12 bit Raw Bayer / 16bit YUV 4:2:2 (Optional)		
Shutter Speed	1usec~65sec		
Gain Control	0 ~ 27 dB		
Gamma	0.4~2.5		
External Trigger	Mode 0, 1, 2, 3, 4, 5, 14		
Signal to Noise ratio	56dB or better		
Video Out Interface	IEEE-1394 (400Mbps, Supply Voltage 8 ~ 30V DC from cable)		
1394 DCAM version	IIDC 1.31		
Lens Mount	C Mount		
Advanced Features	Pixel Binning via Format7 (2x2, 1x2)-IMC-1200FT Model Only / RS232(SIO)-Pass Through / Region of Interest via Format7 (sub-windowing 4x4) / One shot & Multi shot / Multi-Camera Auto Sync / Detachable Filter Design / Opto-Isolated I/O / Industrial Screw Lock Cable Support		
Regulations	FCC, CE, MIC, ROHS		
Operation Temperature	-5°C ~ 45°C		
Dimensions / Weight	44 (W) x 29 (H) x 63 (D) mm / approx. 110 gram		
Supplied Accessories	Lens cover, User's Manual, Demo Application		

Note : Specification may subject to change without notice.

IMI TECHNOLOGY CO., LTD. *Pioneer in Digital Vision Technology*

#2 Eun-Seok Bldg. 729-1 Yeoksam-Dong, Gangnam-Gu, Seoul 135-080, Korea,

Tel : 82-2-539-6227 Fax : 82-2-539-6358 Email : w3master@imi-tech.com Website : www.imi-tech.com

Figure C.3: Specification sheet for the imi tech IMC1080FT Camera (imi tech, 2007)

Appendix D

Dataset Description

D.1 Dataset 1

The first image set obtained from Van Rob contains a total of 706 images. Of those, 651 are pass images and 55 show something wrong that should cause the part to fail. This image set has good colour rendition, but there are often changes in clip appearance due to the reflections off the clip surface.

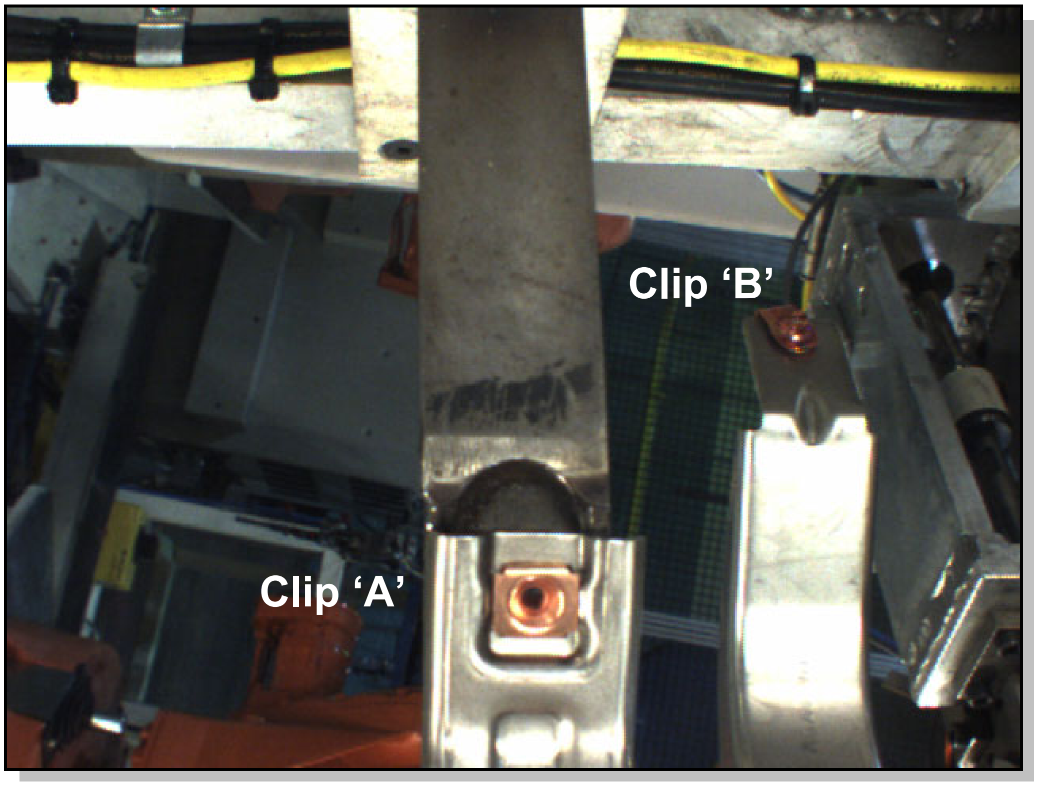


Figure D.1: An example of "View 2" under the original lighting conditions at Van Rob. Two clips need to be inspected as indicated.

D.2 Dataset 2

The second image set obtained from Van Rob contains a total of 3,506 images. Of those, 3223 are pass images and 283 show something wrong that should cause the part to fail. Although the reflections in these images are more stable than the first image set, the lighting is so strong that almost all colour information is lost.

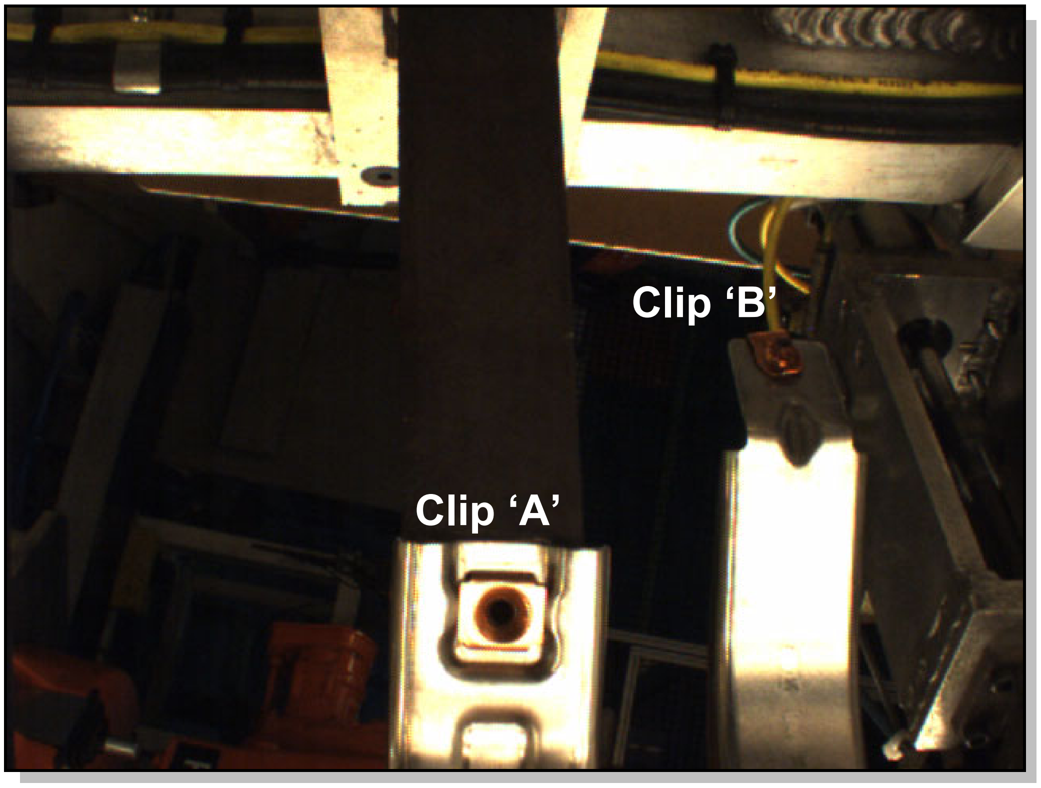


Figure D.2: An example of "View 2" from the second Van Rob image set. Two clips need to be inspected as indicated.

Table D.1: Summary of the images included for View 2

Directory Name	Description	# of Images
Class_BadA	Only clip A is missing	47
Class_BadAB	Both clips are missing	50
Class_BadB	Only clip B is missing/incorrectly inserted/bent	13
Class_None	No clips are missing	1,595

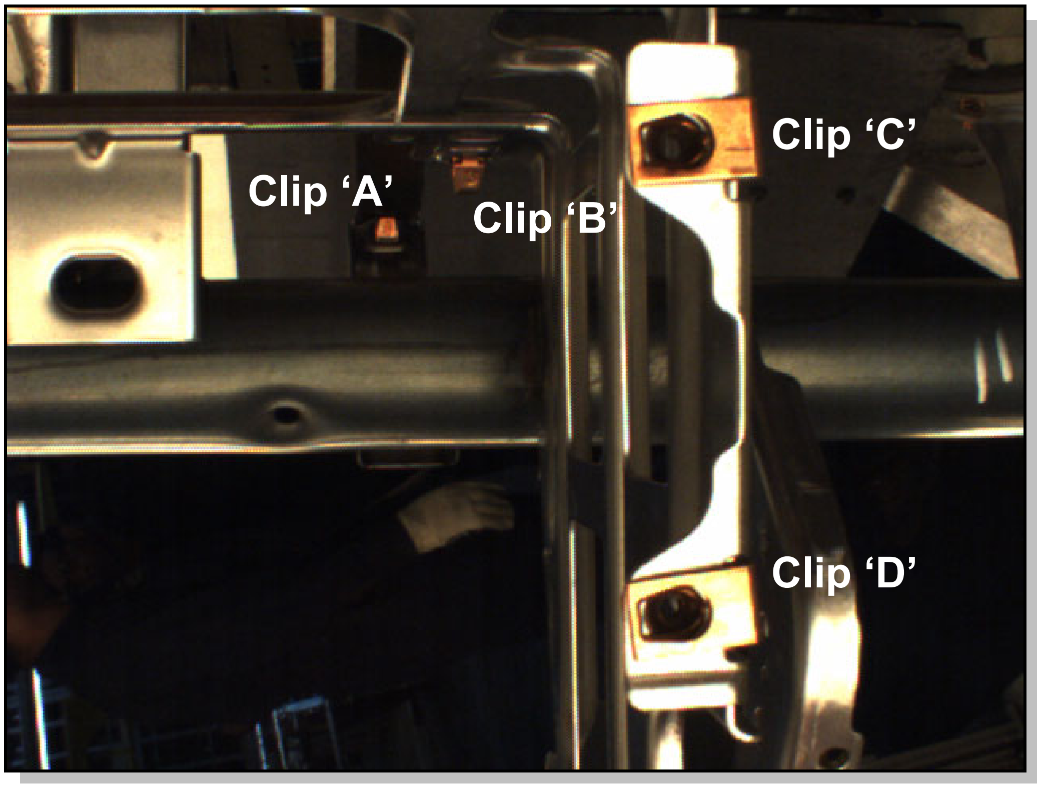


Figure D.3: An example of "View 8" from the second Van Rob image set. Four clips need to be inspected as indicated.

Table D.2: Summary of the images included for View 8

Directory Name	Description	# of Images
Class_BadAB	Both clips A and B are missing	14
Class_BadB	Only clip B is missing	26
Class_BadC	Only clip C is missing/not inserted correctly	9
Class_BadD	Only clip D is missing/not inserted correctly/bent	33
Class_None	No clips are missing	627

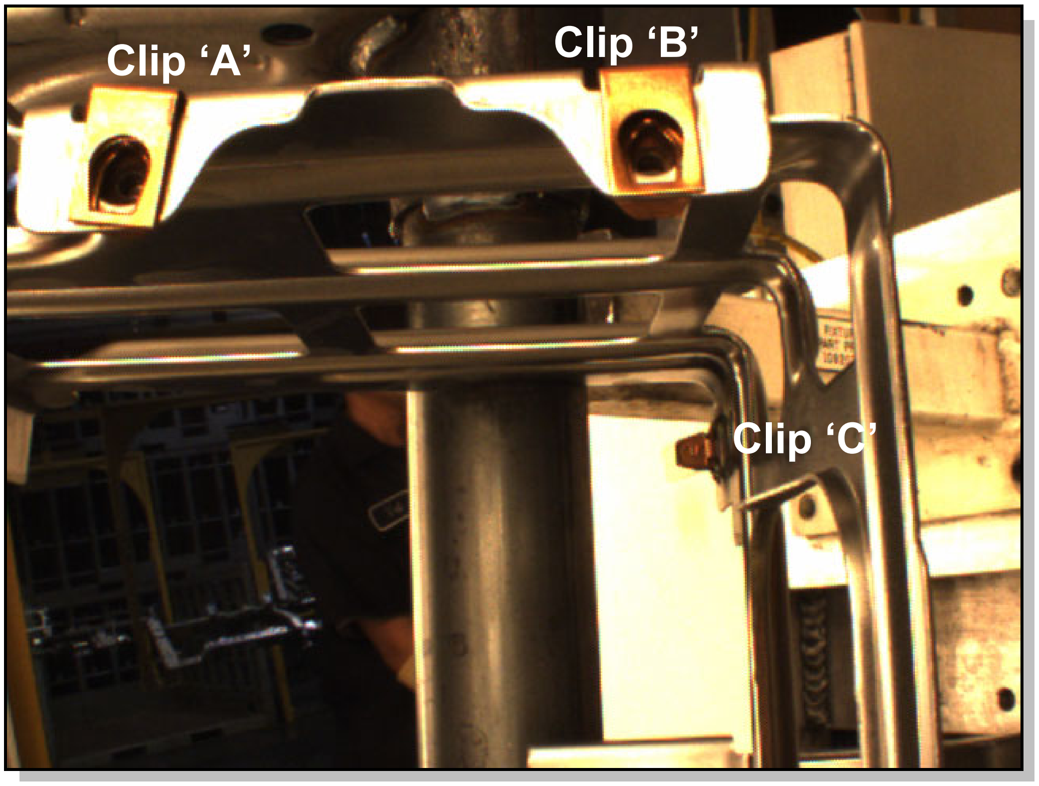


Figure D.4: An example of "View 9" from the second Van Rob image set. Three clips need to be inspected as indicated.

Table D.3: Summary of the images included for View 9

Directory Name	**Description**	**# of Images**
Class_BadAB	Both clips A and B are missing	36
Class_BadB	Only clip B is missing	1
Class_BadC	Only clip C is missing/not inserted correctly	3
Class_BadD	Only clip D is missing/not inserted correctly/bent	51
Class_None	No clips are missing	1,009

D.3 Laboratory Generated Dataset

The laboratory generated images consist of 100 images of a portion of the beam containing one J-type clip. The pass images have the clip present under various lighting conditions (see Figure D.5).

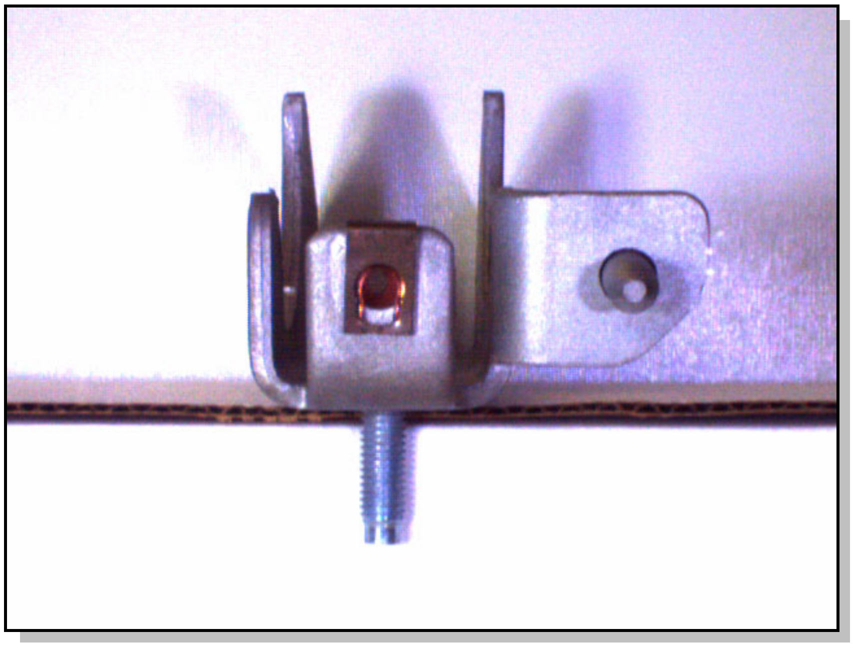


Figure D.5: An example of a pass image from the laboratory generated image set.

Since the clip can rotate when properly installed, several different clip orientations are imaged as part of the set (see Figure D.6).

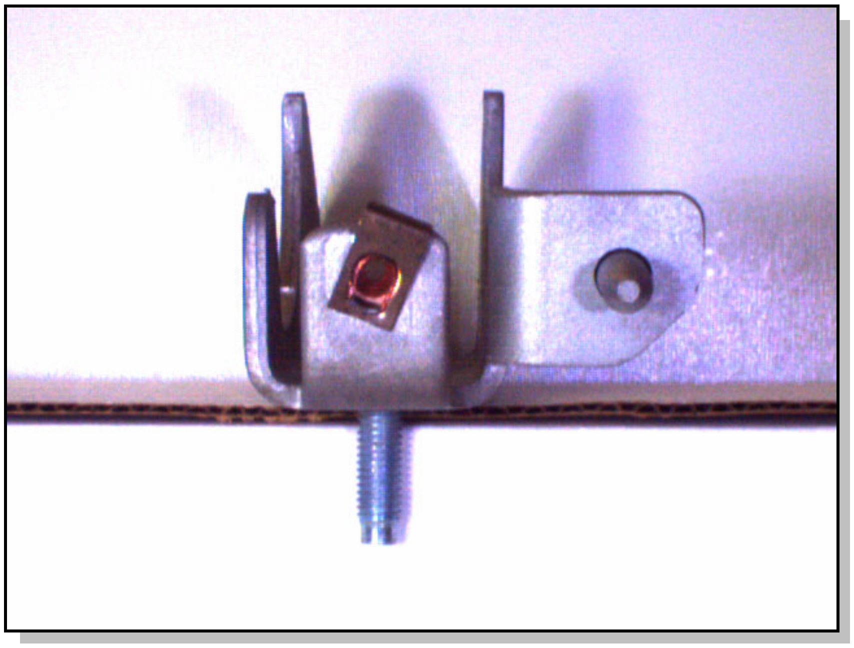


Figure D.6: An example of a pass image with the clip rotated but still correctly installed.

For the fail images, the clip is completely removed and the part is again imaged under various lighting conditions (see Figure D.7).

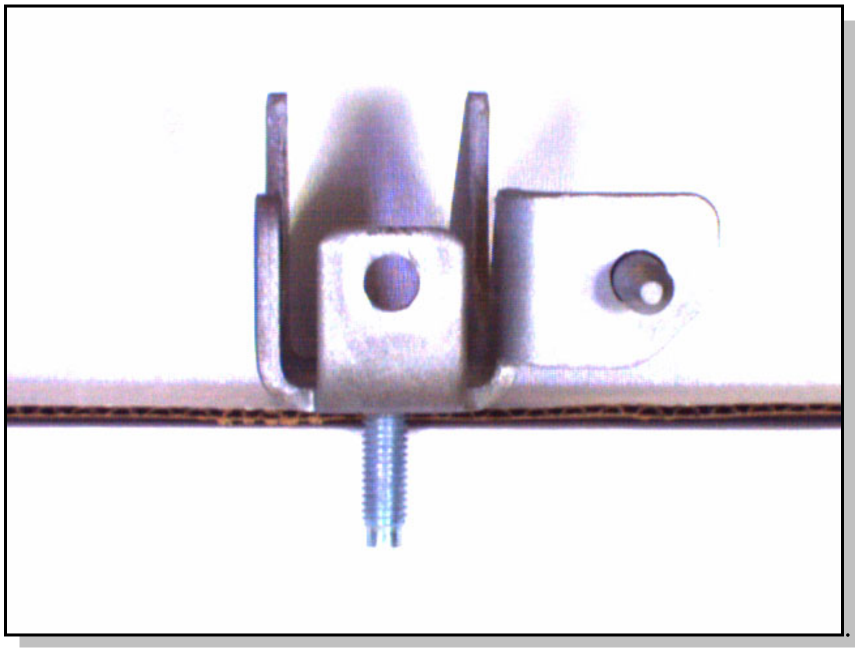


Figure D.7: An example of a fail image from the laboratory generated image set.