An Automated Apparatus for Non-Contact Inspecting of Mass Produced Custom Products

Shaniel Davrajh-202500776

School of Mechanical Engineering

University of KwaZulu-Natal

Submitted in fulfillment of the academic requirements for the degree of Master of Science in Engineering

February 2009

Supervisor: Professor Glen Bright

PREFACE

I, Shaniel Davrajh, declare that:

(i) The research reported in this thesis, except where otherwise indicated, is my original work.

(ii) This thesis has not been submitted for any degree or examination at any other university.

(iii) This thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.

(iv) This thesis does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

a) their words have been re-written but the general information attributed to them has been referenced;

b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced.

(v) Where I have reproduced a publication of which I am an author, co-author or editor, I have indicated in detail which part of the publication was actually written by myself alone and have fully referenced such publications.

(vi) This thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the thesis and in the References sections.

Acknowledgement

The work presented in this dissertation was carried out under supervision of Professor Glen Bright, at the School of Mechanical Engineering, University of KwaZulu-Natal. I would like to thank Professor Bright for going out of his way to not only develop my academic skills; but also for showing me how to apply my life skills to research and personal development.

I wish to express my gratitude to the following people for various reasons:

- My understanding and generous family for their everlasting support and encouragement. It is on their shoulders that I was able to stand and become who I am
- CSIR for their generous financial and material resources
- The National Research Foundation (NRF) and the University of KwaZulu Natal for all financial support received
- Mr Rathilall Sewsunker for his guidance from an electronic engineering perspective
- Dr. R Loubser for his ever-friendly assistance with mechanical modelling involved with the research
- Mr Riaan Stopforth for assistance with electronic hardware selection and design
- Mr Greg Loubser for assistance in producing certain electronic hardware
- Mr Seffat Chowdhury for immense assistance in software and modelling aspects of the project
- Mr Kumaran Moodley for his assistance in programming of microcontrollers
- The workshop staff from the Mechanical engineering department for fabricating mechanical components
- My loving future-wife for her tolerance, support, and encouragement through testing times
- My friends and colleagues who had an influence on my career

Abstract

The evolution of the manufacturing industry may be viewed as proceeding from Dedicated Manufacturing Systems (DMS) to Reconfigurable Manufacturing Systems (RMS). Customer requirements change unpredictably, and so DMS are no longer able to meet modern manufacturing requirements. RMS are designed with the focus of providing rapid response to a change in product design, within specified part families. The movement from DMS to RMS facilitates mass-production of custom products. Custom parts require inspection routines that can facilitate variations in product parameters such as dimensions, shape, and throughputs. Quality control and part inspection are key processes in the lifecycle of a product. These processes are able to verify product quality; and can provide essential feedback for enhancing other processes. Mass-producing custom parts requires more complex and frequent quality control and inspection routines, than were implemented previously. Complex, and higher frequencies of inspection negatively impact inspection times, and inherently, production rates. For manufacturers to successfully mass-produce custom parts, processes which can perform complex and varying quality control operations need to be employed. Furthermore, such processes should perform inspections without significantly impacting production rates. A method of reducing the impact of high frequency inspection of customized parts on production rates is needed.

This dissertation focuses on the research, design, construction, assembly, and testing of a Non-Contact Automated Inspection System (NCAIS). The NCAIS was focused on performing quality control operations whilst maintaining the maximum production rate of a particular Computer Integrated Manufacturing (CIM) cell. The CIM cell formed part of a research project in the School of Mechanical Engineering, University of KwaZulu-Natal; and was used to simulate mass-production of custom parts. Two methods of maintaining the maximum production rate were explored. The first method was the automated visual inspection of moving custom parts. The second method was to inspect only specified Regions of Interest (ROIs). Mechatronic engineering principles were used to integrate sensor articulation, image acquisition, and image processing systems. A specified maximum production rate was maintained during inspection, without stoppage of parts along the production line occurring. The results obtained may be expanded to specific manufacturing industries.

Table of contents

List of Figures

List of Tables

List of Abbreviations

AGV:	Automated Guided Vehicle
AS/RS:	Automated Storage Retrieval System
AVIS:	Automated Visual Inspection System
CIM:	Computer Integrated Manufacturing
CMM:	Coordinate Measuring Machine
CS:	Conveyor System
CCD:	Charge Coupled Device
CMOS:	Complimentary Metal Oxide Sensor
DMS:	Dedicated Manufacturing System
DC:	Direct Current
FMS:	Flexible Manufacturing System
FOV:	Field of View
GUI:	Graphical User Interface
LED:	Light Emitting Diode
MCM:	Mass Customisation Manufacturing
MHS:	Materials Handling System
NCAIS:	Non-Contact Automated Inspection System
PCS:	Part Centralization System
PCM:	Pulsed Coded Modulation
PIS:	Part Identification System
PKM:	Parallel Kinematic Manipulator
PWM:	Pulsed Width Modulation
RFID:	Radio Frequency Identification
RME:	Reconfigurable Manufacturing Environments
RMS:	Reconfigurable Manufacturing System
RMT:	Reconfigurable Manufacturing Tool
ROI:	Region of Interest
S/N:	Signal to Noise Ratio
SPS:	Sensor Positioning System
UDL:	Uniformly Distributed Load
USB:	Universal Serial Bus

USART:	Universally Synchronous/Asynchronous Receiver Transmitter
VAM:	Value Added Manufacturing
VS:	Vision System

1. Introduction

1.1 Research background

Quality control and part inspection are key manufacturing processes in the lifecycle of a product. These processes have many functions such as detection of flaws, and relaying information as to the nature and location of these flaws, thus providing for the improvement of the overall manufacturing process. No change is made to a product during inspection, in order to increase its value. Time and resources are spent on these processes, without a gain in profit. The advantage of quality control and part inspection is that customer loyalty through product integrity may be sustained. The disadvantage of these processes is that even though the lead time (time taken for products to reach customers) is increased, the product may not be sold at a higher price, since no value has been added [1]. The reduction of the time spent on these processes is therefore an attractive concept to any manufacturer.

Product quality can be defined as the measure of the extent to which a product can satisfy a given function [2]. Quality control is the process of ensuring that a part lies within its design specifications. Quality control can be divided into inspection and testing. Testing a part involves examining the operation of that part, whilst performing its required application. Part inspection is the search for aspects of the product that do not lie within design tolerances, without operation of the part. Inspecting a part has three main applications, namely detection of flawed products; determining significance of flaws; and provision of process feedback [4]. Detection of significant flaws in products allows for prevention of further production of flawed products, thus minimizing materials wastage. The nature and location of detected flaws allows for the feedback of the operation of a process. Using this feedback, process parameters may be optimised in terms of costs and time. An example of where process feedback is essential is to justify the shift to cheaper materials in the design of a product.

The manufacturing industry can be viewed as evolving from Dedicated Manufacturing Systems (DMS) to Intelligent Manufacturing systems. DMS are manufacturing systems that implement processes which perform only one specific operation, on a specific part [5]. An example of this type of manufacturing system is the manufacturing lines employed to manufacture cars in the mid 1900s. DMS are ideally suited for mass-producing identical products in high-volume batches. Producing identical parts in batches allows for the use of statistical inference as a method of quality control. This quality control method involves inspection of a relatively small percentage of sample parts from a batch. Inspecting only a few samples may lead to consumer risk and producer risk [1]. Consumer risk is the acceptance of a batch of defective products, based on the sample set being within tolerances. Consumers may therefore be prone to purchasing defective products. Producer risk involves the possibility of rejecting a batch of acceptable products, based on the sample set consisting of defective products. This allows manufacturers to be vulnerable to significant financial losses, in terms of manufacturing time and other resources, by rejecting parts that lie within tolerances.

Recent manufacturing trends are influenced by customer requirements to such an extent that modern products are now designed to meet each individual's needs. Due to the fact that product requirements may differ for each customer, products currently produced may often be unique and are often produced in small batches, which have varying product parameters. This makes the use of DMS ineffective. New products usually generate the highest revenue on their introduction to a market [6]. As the product matures, the number of competitors that are able to manufacture that particular product increase. This increase in competition leads to a decrease in profit margins for manufacturers. Therefore modern manufacturers need to be able to implement processes that have the capability to provide efficient, cost-effective and rapid response to changes in product design and customer requirements, in order to maintain a competitive edge. Flexible Manufacturing Systems (FMS) are systems that aim to produce a high variety of parts [5]. FMS are employed in situations where a high variety of a small number of products, is required. This type of system is not effective where a variety of parts in large volumes, is required. A strategy employed to handle production of a variety of parts in large volumes is Mass Customisation Manufacturing (MCM). MCM facilitates the concept of mass personalisation. Mass personalisation is the mass production of customised parts and part families [8]. MCM involves manufacturers allowing the customer to have a design input at various stages in the design process. Mass produced custom parts often vary in design parameters such as colour; dimensions; tolerances; assembly; finish; costs; and throughputs.

A system that has been developed to cope with mass-producing custom parts is Reconfigurable Manufacturing Environments (RME). RME involve Reconfigurable Manufacturing Systems (RMS) that are designed for providing rapid response to sudden, unanticipated changes in product requirements [7]. An RMS may be composed of CNC machines, Reconfigurable Machine Tools, Reconfigurable Inspection Machines, and

Materials Handling Systems that facilitate part transportation from one machine to the next within the manufacturing environment. RMS and FMS differ, in that an FMS is implemented in environments that aim at producing small sets of a high variety of products, whereas the main focus of an RMS is to minimise the time taken for a manufacturing system to respond to varying market and customer requirements with respect to limited parts and part families. A part family can be considered as a group of parts, which have some aspect of their design in common. These common aspects include similar shape, colour, dimensions, tolerances, manufacturing processes, cost, etc. The production volumes produced in an RMS may vary from low to high [6]. Table1-1 shows the applicability of the different manufacturing systems to mass-producing custom parts, with the meeting of the requirements highlighted in yellow. It can therefore be deduced that RME are best suited to mass-producing custom parts since they meet all the key requirements, whereas other manufacturing systems don't.

Requirements	**DMS**	**FMS**	**RME**
Production volume	High	Low	Varying from low to high
Part variety	Very low	Very high	High within specified part families
Ability to facilitate varying Inspection requirements	Low	High	High within specified part families
Inspection Time-efficiency	Usually High	Low	High
Frequency of inspection	Infrequent	Very frequent	Ranges from infrequent to very frequent

Table1-1: Comparison of different manufacturing systems for mass producing custom parts

There are six key characteristics that an RMS may hold at all levels of its design. These include: Modularity; Integrability; Customized flexibility; Scalability; Convertibility; and Diagnosability. Modularity may be viewed as the subdivision of a system into subsystems (modules), which are used to perform a given function. These modules allow for the partitioning of the machine service and maintenance requirements, the implementation of which lowers the life-cycle cost of the entire system. This reduction in life-cycle cost is attributed to the fact that modules are easier and more cost effective to maintain than an entire system. The upgrading of the system may often require only an upgrade of certain modules, which again is more cost effective than the upgrade of the entire system. Factors to consider when designing a modular system are selection of the actual modules and integration of these modules (Integrability), which will affect performance of the system as a whole. Customized flexibility is a concept that involves the design of a system to be flexible within a given part or part family. This places various constraints on the design of the components of the system, in order to conform to requirements of specified parts and part families. The advantages of customized flexibility include faster throughput and higher production rates as opposed to the more general flexibility requirements for FMS, which do not have as many confined constraints.

Convertibility is the capability of a system to change its operation to best suit a change in product requirements. An example of this concept is a machine inspecting two rectangular blocks of significantly different sizes, colour, and material. Implementation of this concept has an advantage of a single system being able to adapt to changes in product requirements without having to implement new or other existing systems, thereby reducing the overall process costs. Scalability is the ability of a system to adapt to a change in production capacity. This means that the inspection system would require the ability to inspect moving parts at velocities that sometimes differ. Diagnosability is the ability of a system to detect poor part quality, for manufacturing systems that may differ in process layout [7].

Another manufacturing strategy developed to increase throughputs is Value Added Manufacturing (VAM). This manufacturing concept is the process of retrofitting (modifying or making additions to) a base platform [9]. Often, the inspection requirements of these parts entail inspection of only areas of the part that were significantly affected by the modification process. The reason for this is that the platforms used, have already undergone quality control processes prior to supply of these parts.

1.2 Motivation for the study

Mass-producing custom parts does not allow for use of statistical inference as a method of quality control. This is due to the fact that batch sizes are reduced and often inconsistent; and products having different inspection requirements. These parts therefore require regular inspection in order to ensure high quality standards. Due to the nature of these parts, high frequency of often unique inspection routines, are required. These inspection routines require the capability to be reconfigured in order to handle variations in product parameters. The required high frequency of inspection has significant impacts on inspection times and inherently, production rates. Manufacturers would therefore be required to invest more time and resources whilst inspecting mass-produced custom parts. This increased investment would decrease profit margins, discouraging manufacturers from performing extensive quality control procedures.

Value Added Manufacturing (VAM) of parts awards manufacturers the option of performing quality control by inspection of only key modifications and additions to a part. This option awards manufacturers an advantage in that the inspection processes of these parts can be performed quicker, due to reduction in inspection requirements of the entire part. The maintaining of production rates during quality control of mass- producing custom parts and VAM of parts would make the quality control processes more feasible for manufacturers. A method of maintaining production rates whilst mass-producing custom parts and value added manufactured parts, during quality control processes, is thus needed.

1.3 Contribution of the dissertation

The contribution of this dissertation lies with the fact that current research does not focus on maintaining production rates whilst inspecting mass-produced custom parts. Furthermore, current inspection systems do not take advantage of the fact that value added manufactured parts often only require inspection of the modifications or additions that were made to a particular part. This dissertation details the inspection of ROI on moving custom parts for RME. The results of this research are limited to the inspection requirements of the manufacturing cell that the apparatus was tested in; however the inspection strategies may be applied to many industrial applications.

1.4 Project objectives

The objectives of this project were to use the Mechatronic engineering approach to:

- Research, design, construct, and assemble the apparatus by which custom parts are visually inspected. The apparatus must display the ability to be integrated into an RME
- Perform visual inspection of at least one part family by determining the presence/absence of features in a part; validating correct assembly of components;

validating correct finish with respect to part colour (and to some extent, dimensions); determining the presence and significance of flaws

- Research and implement machine intelligence that provides optimised inspection of user defined ROI
- Provide dynamic access to various ROI in order to facilitate inspection of moving custom parts
- Test and obtain results for maintaining production rates in a given CIM cell

1.5 Design specifications for apparatus

One of the requirements of the design was that the apparatus should integrate with the existing infrastructure (CIM cell) at the School of Mechanical Engineering, University of KwaZulu-Natal. The time allocated for inspection was therefore based on the maximum operating conveyor speed of the CIM cell. This speed was recorded as 0.02m/s. It was decided to use the existing Automated Visual Inspection System (AVIS), which was already designed to integrate into the infrastructure of the CIM cell, as a platform on which to base the design of the NCAIS. The following criteria were considered for the research, design, and development of the apparatus:

- Inspection of a part moving at 0.02m/s must occur without the part having to stop during the inspection routine.
- All faces except the bottom face of inspected parts needed to be accessible by the sensor at least once during an inspection routine.
- The maximum height, width, and length of a part that would be inspected were 200mm x 200mm x 200mm respectively.
- Positioning and orientation of the sensor needed to be accurate and repeatable in order to inspect the appropriate ROI. Due to image processing software being able to correct positioning errors, an accuracy of within 8mm of the desired point, was acceptable.
- Motion of the part during the inspection routine must occur at a constant velocity for synchronisation of the sensor and part pose
- The inspection routine must be complete as the part exits the apparatus. This includes the decision about the acceptability of that part. Using the maximum conveyor speed and maximum part travel distance of 800mm, the allowable inspection time when running at maximum throughput was set to be 40s.

1.6 Outline of dissertation

The layout of this dissertation is as follows:

Chapter 1: Introduces the reader to the background of and motivation for the project. All objectives and specifications are explicitly stated in this chapter.
Chapter 2: Informs the reader about different inspection methods used for quality control processes, and highlights the most appropriate inspection sensors for inspecting mass-produced custom parts
Chapter 3: Discusses proposed concepts for the design and operation of the apparatus
Chapter 4: Details the design of the core elements of the apparatus and its subsystems using Mechatronic engineering
Chapter 5: Deals with the assembly phase of the project
Chapter 6: Quantifies system results and validates against system specifications. Discusses the obtained results for the system
Chapter 7: Concludes the project and discusses flaws in the design as well as future improvements.
Chapter 8: Provides references for information presented
Chapter 9: Shows system drawings; system results; component specifications; and software coding

1.7 Summary of chapter 1

This chapter served to introduce the reader to the subject background for inspecting moving custom parts in RME. A motivation for the study; dissertation contributions; definition of project objectives; and definition of system specifications were all explicitly stated in this chapter. The outline of the dissertation was also described.

2. EXISTING INSPECTION METHODS

2.1 Contact and non-contact inspection

Methods of inspection can be divided into contact and non-contact inspection techniques. Non-contact methods for quality control and part inspection are generally quicker than contact methods, and deformable parts can also be inspected. The possibility of contamination due to contact between sensor and parts is eliminated when implementing non-contact methods of inspection. Coordinate Measuring Machines (CMMs) offer highly flexible methods of inspection, and have been employed by manufacturers who wish to minimise the cost of adapting to manufacturing changes [6]. These machines are used to physically measure the geometrical properties of a part. A probe is attached to a manipulator, and is used to acquire information about the size and location of features of interest on a part. This information is then used for dimensional measurement; profiling; and image construction [10]. The advantage of these machines is that they are highly flexible and are hence able to perform inspection of a high variety of parts. This reduces capital investment by manufacturers who require inspection of a variety of parts.

The disadvantages of CMMs are that the inspection rates of these machines are often significantly less than production rates, and information obtained is limited to geometrical properties of the part [6]. Slow inspection rates cause bottlenecks and so inspection using these methods is often performed off-line, on an infrequent basis. Defective parts are therefore only detected after a significant number of these parts have been produced. This results in batches of parts being rejected as opposed to rejection of individual parts. CMMs are therefore not suited for mass producing custom parts.

2.2 AVIS structure and design overview

The AVIS was developed to perform automated multi-faced part inspection using PC based technology and a single digital camera. The sensor was mounted on a C shaped track (shown in Figure 2-1), perpendicular to the direction of flow of products within the machine. This was done to achieve various sensor heights. A rotational part manipulation platform allowed for the part to rotate relative to the sensor, allowing for multi-faced access. The AVIS was able to produce 2D images of a product, which were used for inspection purposes. The structural frame of the AVIS was designed to have high rigidity and vibration damping characteristics, thus minimising vibrations imparted to the sensor, during operation of the machine. This allowed the inaccuracies associated with the acquisition of images to be reduced [2].

The AVIS was designed having a rectangular-volume base with a trapezoidal-volume upper-half, which was used as the inspection workspace. Figure 2-1 shows the mechanical structure designed for the AVIS, along with the sensor housed on the track. The length, width, and height of the base were 800mm x 800mm x 600mm respectively. The trapezoidal volume available as a workspace had a base area of 800mm x 550mm; a top surface area of 450 mm x 550mm; and a height of 460mm.

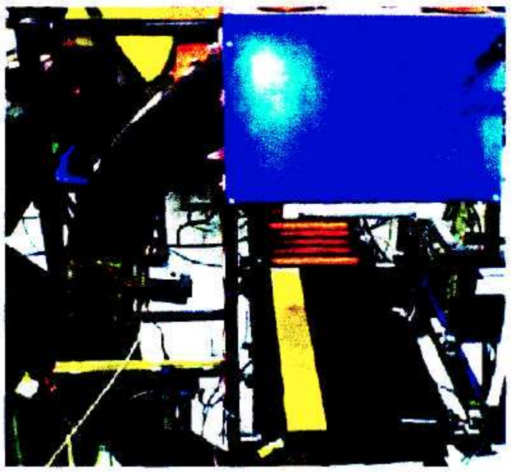


Figure 2-1: The AVIS

The operation of the AVIS involved a part entering the machine and stopping at the centre at the part manipulation system. The camera would then be positioned at the required height to acquire an image of the part. Due to the single plane motion of the camera, the part was rotated for accessing different faces of the inspected part. Inspection routines required that the part be stationery during inspection, which impacted production rates. The conveyor system used was two parallel belts, driven by a single motor. These belts were found to move at different rates due to slipping, over a period of time, and this caused rotation of the part as it moved along the machine. This system was not acceptable for inspection of moving parts, when the inspection routine required a predictable part pose. The lighting system involved the use of two fluorescent lights fixed above the conveyor. This was not optimal for varying inspection routines.

2.3 Sensors for inspection

Sensors are devices that are used to convert a physical property of an object into a signal that can be used by processing units [1]. Sensors that may be used for inspection purposes can be divided into contact and non-contact sensors. Contact sensors include CMM probes, which make contact with the part being inspected. Non-contact inspection methods involve a specified distance between the sensor and the part; and can be divided into optical and non-optical technologies [11]. Laser systems; and cameras are amongst the most common optical methods found in industry. Non-optical sensors include eddy-current sensors; radiation sensors; and ultrasonic sensors. Non-contact sensors offer advantages of fast inspection times; and ability to inspect sensitive parts without any deformation occurring.

2.3.1 Optical techniques

Optical techniques for inspection allow for measurement and analyses of parts. Information about a part is obtained via reflectance; light scatter or diffusion; and laser technology. Due to high speeds of operation, these techniques allow for in-line inspection.

2.3.1.1 Cameras

A camera is an image sensor that is used for converting light into electric charge, and then processing it into electric signals [3]. Automated inspection image sensors can be divided into Charge Coupled Devices (CCDs) and Complementary Metal-Oxide Semiconductor (CMOS). Although not as fast as some other methods of inspection, these sensors allow for the highest amount of information to be extracted from a reading. These sensors can be used for obtaining information about dimensions; colours; shapes; and even surface finish to some extent. Since these sensors are able to provide such information about inspected parts, they are well suited toward inspecting custom parts. More detailed information about cameras is given in section 4.2.2.2

2.3.1.2 Lasers

A laser is a device that emits monochromatic, highly structured light; using a process called stimulated emission [3]. Laser sensors are used to determine the physical dimensions and profiles of an object. These systems are extremely fast and accurate. Inspection involving lasers include time-of-flight; laser scanning; and structured lighting [11]. Time-of-flight sensors use the time delay of a reflected laser beam to determine the distance between the sensor and the object reflecting the beam. This method yields resolution of up to twenty-five microns. Laser scanning is based on triangulation methods to determine the distance of an object from two known locations, using trigonometric relationships between the sensor, light source, and object. Using this method, it is possible to obtain three-dimensional spatial data of an object, with a resolution in the region of thirty-five microns. The disadvantage of laser-scanning systems is that they are prone to shading and occlusion problems. Structured lighting systems involve projecting coded light onto a surface and checking for fringes in the projected beam. Fringes will occur at variant surface profiles. This method of inspection is also prone to shading and occlusion problems. Laser sensors are very accurate and facilitate in-line inspection of parts; however only geometrical object data can be obtained [14]. This made sole use of laser sensors for inspection of custom parts unsuitable.

2.3.2 Non-optical techniques

2.3.2.1 Eddy-current sensors

Eddy-current sensors are non-contact sensors that use magnetic fields for high resolution inspection of conductive objects. Eddy-currents are small currents which are induced in an object, due to electromagnetic induction, by an alternating current in a probe coil. These small currents create an opposing magnetic field, which resist the field initiated by the probe coil. This field interaction is proportional to the distance between the probe and the inspected object. A signal is generated when a change in current field interaction is sensed. These sensors are used for position measurement; drive shaft monitoring; and thread detection. Figure 2-2 shows the principle of operation of these sensors.

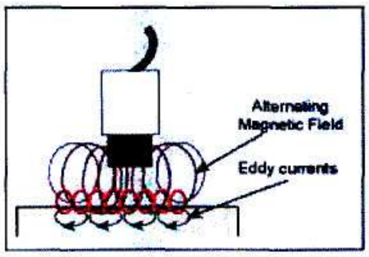


Figure 2-2: Principle of operation of eddy-current sensors [12]

Eddy-current sensors can also be used to detect material thickness; coating thickness; conductivity measurements; and crack detection. Advantages of inspection using eddy-current methods include detection of small cracks; portable equipment; minimum part preparation required; and high tolerances of dirty environments. Disadvantages of eddy-current sensors are that only conductive materials can be inspected; they have high sensitivity to non-uniformities in material structure; and flaws that lie parallel to the probe, are often undetected [12]. Due to the fact that only conductive parts may be inspected using these sensors, the sole use of eddy-current sensors for inspecting mass-produced-custom parts is unsuitable.

2.3.2.2 Capacitive sensors

Capacitive sensors offer non-contact, high-resolution measurements of conductive targets. These methods of inspection involve measurement of the capacitance between two conductive surfaces. A change in capacitance would indicate a change in distance between the sensor and the part, and this can be used for profiling feedback. These sensors have the advantage of being extremely accurate (within nanometre accuracy); high tolerance to changes in materials; and are stable with temperature. The disadvantages of these sensors are that they are expensive and are not effective in dirty or wet environments. Applications of inspections involving capacitive sensors include precision vibration measurement; precision thickness measurement; and assembly validation. These sensors may also be used in inspection of non-conductive materials. This is achieved by placing the non-conductive material between the sensor and a reference conductive plate. Reference readings are acquired when running the sensor along the plate at a fixed distance. Any differences in sensor readings when a part is present can only be attributed to properties of the part [13]. Due to the high-cost factor, these sensors were not considered for this project.

2.3.2.3 Ultrasonic sensors

Inspections using ultrasonic sensors involve emitting and receiving of high frequency sound waves in order to detect and evaluate flaws; make dimensional measurements; and characterize materials [14]. Technological advancements have made these sensors more robust, flexible, and affordable. The operation of ultrasonic inspection relies on a sensor pair (emitter and receiver) to measure distances between sensors and inspected objects. Internal material properties may be examined by use of an appropriate coupling. Advantages of ultrasonic sensors are that minimal part preparation is required; they are relatively inexpensive, and they are reasonably accurate. The disadvantage of ultrasonic inspection is that materials that are rough, irregular in shape, very small, non-homogenous, and materials with low densities, are all difficult to inspect. These disadvantages all contribute to the rejection of ultrasonic sensors for use as sole sensors of the NCAIS.

2.4 Vision systems- overview

Machine vision systems for manufacturing industries are primarily concerned with the automatic interpretation of images in order to obtain information about, and control, manufacturing processes [15]. An image can be defined as a spatially discrete scalar function of two independent variables, these two variables being spatially discrete as well. Images obtained may be due to visible light, x-ray, infra-red energy, and ultrasound information. The NCAIS used images based on visible light (light intensity), and so these systems will be discussed. The purpose of a light-image is to numerically represent a physical object in detail, by obtaining spatial (geometric) and spectral data about real world scenarios. The image can then be used to mathematically represent the image as a matrix of light intensity levels. This matrix is made up of picture elements which are referred to as pixels. Images can be classified into binary images; grey-scale images; and colour images [16].

Binary images allow a pixel value of either 1 or 0, corresponding to white or black respectively. The classification of a pixel value is set on a predefined light-intensity threshold value. These images require the least amount of memory for representation of all three images. Applications of binary imaging include those involving low memory-applications for determining the presence or absence of an object; measurement of an object; and applications involving high speed processing. Grey-scale images are monochrome (usually 8-bit) images which allow for pixels to have a value ranging from 0 to 255. These pixel values are based on an adaptive thresholding technique that represents various shades of grey. The convention used is that a pixel value of 0 represents black; 255 represents white; and values in between represent the linear shade of grey. These images require more memory than binary images; however they have the advantage of distinguishing a wider variety of objects with different reflective and lighting properties. Colour images are images that represent an image on the Red, Green; and Blue (RGB) spectrums. These images may be thought of as the combination of three monochromatic images, one for each channel of the RGB spectrums. Consequently, these images require the most amount of information to be represented, but they are able to provide the most accurate data from all three types of images.

The output of a vision system is generally to make an inspection decision or to permit a comparison with other data. This output can be categorised into one of four machine vision applications namely: assembly verification; flaw detection; dimensional measurement; and positioning applications [17]. Vision systems consist of vision hardware and software. The key components of a vision system include a light source, an image capture device (camera), a digitising device, and a digital processing unit (vision computer). The light source, image capturing device, and digitising device, are responsible for the image acquisition and formation process. The processing unit is responsible for analysis of the acquired images. The signal to the processing unit may be either digital data or analogue data. In the case of analogue data transmission, a frame grabber is required for digitisation of the image. A schematic layout of a typical vision system, with the flow of the acquired signal, is shown in Figure 2-3.

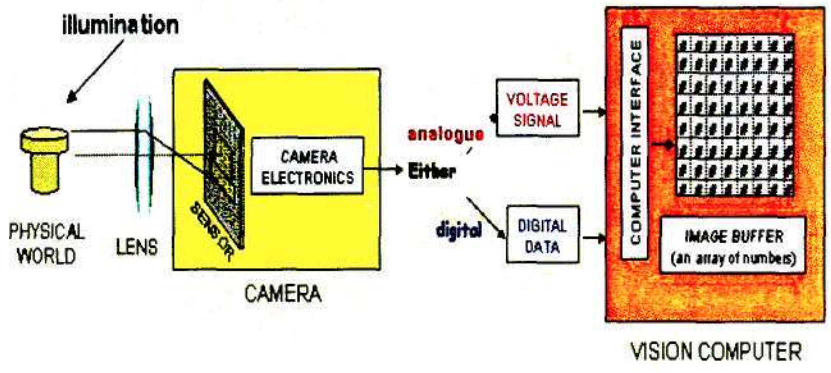


Figure 2-3: Typical layout of a vision system [23]

The reliability and accuracy of a vision system are largely dependent on the images acquired. The quality of the images acquired depends on lighting and sensors used for the acquisition process. The acquisition process aims to produce images that have the following characteristics:

- Well defined features/ ROI (high contrast with sufficient detail)
- High signal to noise (S/N) ratios
- High resistance to external disturbances

Well defined ROI allow for a more reliable and accurate operation as more data can be obtained for processing. High S/N ratios allow for more accurate representation of objects, allowing for more reliable image processing. A high resistance to external disturbances such as light and vibrations leads to clearer and more detailed images, again leading to more accurate and reliable image processing.

Following acquisition of an image, processing is required for turning the information into usable data and then using that data for making a decision. Processing can be divided into image enhancement and image analysis. Image enhancement uses algorithmic transforms to provide a more reliable and accurate image for analysing. This is achieved by performing pixel operations to reduce or eliminate noise; enhance significant features; and suppress irrelevant background information. The objective of image analysis is to generate quantitative data, from the data supplied by the enhancement processes, for feature identification and extraction; assembly verification and dimensional conclusions. All this is to allow the system to make an accept/reject decision. The sequence of a few typical operations that an image undergoes, after acquisition has occurred, is shown below in Figure 2-4.

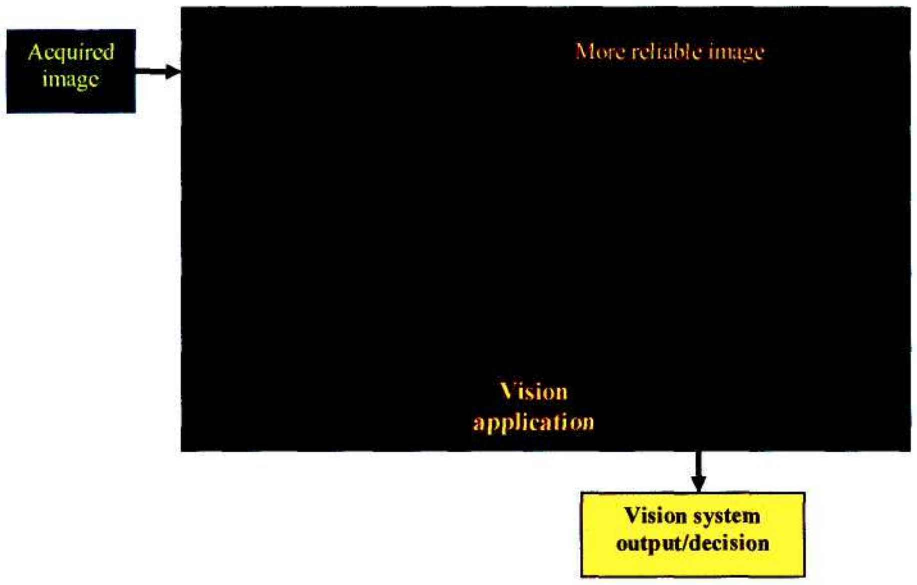


Figure 2-4: Typical sequence of operations on acquired images during processing

2.5 Summary of chapter 2

This chapter highlights the various sensors used in inspection of products. Differences between non-contact and contact methods are stated. The AVIS used for previous inspection in the CIM cell was described. An overview of vision systems was discussed.

3. CONCEPTUALIZED SYSTEM

3.1 Mechatronic engineering approach

Mechatronic engineering can be defined as the holistic design of engineering systems which involve the integration of mechanical engineering, electrical and electronic engineering with software engineering, at all levels of the design [18]. This design approach was introduced in the late 1960's, and stemmed from the use of computer-based technologies to improve the level of performance of mechanical systems [20]. Concurrent engineering is a concept that encompasses a similar approach of considering different aspects of the design concurrently. The difference between Mechatronic engineering and concurrent engineering however, is that in the Mechatronic engineering approach, the interchange of functionality between each of the core elements is considered at conceptual stages of the design process. The core elements of Mechatronic engineering, along with their functional dependencies, are shown below in Figure 3-1 [20]

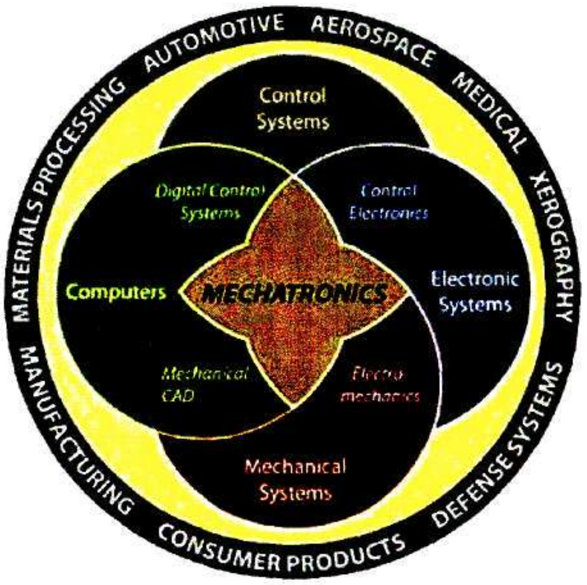


Figure 3-1: Graphical representation of Mechatronic design approach [20]

The mechanical aspect of a Mechatronic engineering approach entails consideration of spatial relationships and interactions between the various mechanical elements of the system. Such relationships and interactions include static and dynamic loading of members; spatial constraints for members induced by physical boundaries; and suitable materials selection for design optimisation. The electrical and electronic components of the approach involve signal-processing and communication between these signal-processing components. The software aspects of the Mechatronic engineering approach involve processing of data and information. A Mechatronic system may not necessarily constitute of equal amounts of mechanical, electrical and electronic, and software components. The advantages of integrating the core elements of Mechatronic engineering include the design of systems that offer enhanced performance, better reliability, and safer operation than if these core elements were designed independently. This approach offers a commercial advantage to manufacturers.

Design and implementation of a Mechatronic engineering system involves eight phases in their respective sequence. These phases are: defining system requirements; conceptualising the system holistically and then subdividing into conceptualised subsystems; design of subsystems; construction of subsystems; testing of subsystems; assembly of subsystems to constitute the entire system; testing of the entire system; and operation of the system [21]. Defining system requirements is dependent upon the problem that is to be solved. This phase is responsible for defining system constraints with respect to part parameters such as throughputs, dimensions, and requirements. The next phase involves conceptualising the system as a whole, in order to operate within the constraints obtained in the previous phase. The subdivision of the system into subsystems allows for incorporation of a modular design. Each subsystem would then have to perform a certain function within an inherent set of constraints. All aspects of the core elements are initially interchanged in the conceptualisation phase.

An example of this is the consideration of the automated speed control of a mechanical member. Given that an actuator has to move a mechanical member at a certain speed, an appropriate sensor and control hardware needs be selected for governing this given speed. Software that is able to provide sufficient control, via sensor feedback, needs to be implemented in order to maintain efficient response to system disturbances. On completion of system conceptualisation, the design phase is executed. The design of subsystems accounts for easy Integrability of subsystems in the assembly phase. Construction of the various subsystems involves manufacturing of customized parts as well as use of Commercial off the Shelf (COTS) products [23]. Once the construction of subsystems is complete, independent testing occurs to ensure that these subsystems perform their specific functions satisfactorily. The assembly phase involves the integration and optimisation of the subsystems. The system is considered as a single system from this phase. On completion of full system integration, the system is tested to ensure that it performs the given function within the constraints that were defined in the initial phase. Operation of the system follows the testing phase. Refining of the system may still occur during the operation phase. Figure 3-2 below shows the sequence of the different phases involved in a Mechatronic engineering design.

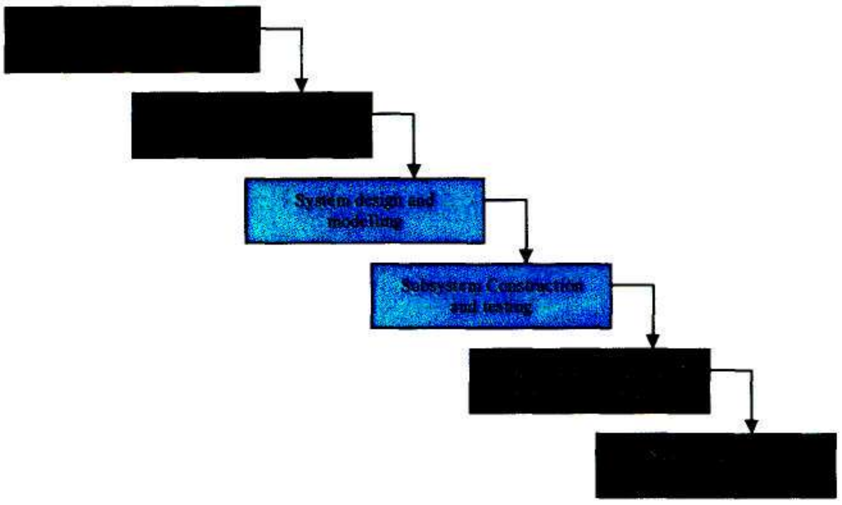

System design and modelling
Subsystem Construction and testing

Figure 3-2: Sequence of design phases involved with Mechatronic engineering design [24]

Use of this approach places emphasis on a detailed and often lengthy conceptual design phases. The reason for this is that the overall number of design iterations and development costs may be significantly reduced. The conceptualisation and design of systems involves implementation of some sort of simulation. Simulating a Mechatronic engineering design may be categorised into a combination of one or more of three representations, namely the physical model; mathematical model; and the computer graphics model [24]. Simulations based on use of a physical model are generally expensive, time consuming, difficult to modify, and impractical. These models are however, easy to understand, and are effective in identifying and isolating key aspects of the design. Mathematical models implement algorithms that are used to represent system behaviour and performance analytically. These models have proven to be accurate; however there is a large amount of difficulty incurred, in fully defining inherent complexities associated with interrelationships between various elements of a design. A method of enhancing accuracy of mathematical models is to merge mathematical models with system hardware. Computer graphics simulations allow for real-time analyses and visualisations of system performance. The use of this type of simulation awards the designer to make critical changes in a design prior to major investment in constructing a prototype.

3.2 CIM cell overview

The CIM cell at the University of KwaZulu-Natal, in the School of Mechanical Engineering, comprised of an Automated Storage and Retrieval System (AS/RS); a conveyor system; a materials handling apparatus; an Autonomous Guided Vehicle (AGV); a Reconfigurable Machine Tool (RMT); and an AVIS (on which the NCAIS was developed). The AS/RS was used to store finished products and raw materials which were to be machined. The conveyor system was used to transport parts to the various components of the CIM cell. A Puma robot and indexing devices were used to handle materials at certain points along the conveyor, for transfer of parts. The AGV was used to autonomously transport parts between the conveyor and remote locations in the cell. The RMT performed off-line reconfigurable machining operations on parts, and these parts were then transported to the AVIS for inspection which occurred whilst the part was stationary [25]. Figure 3-3 below shows the layout of the CIM cell, with the AGV and AS/RS not in view.

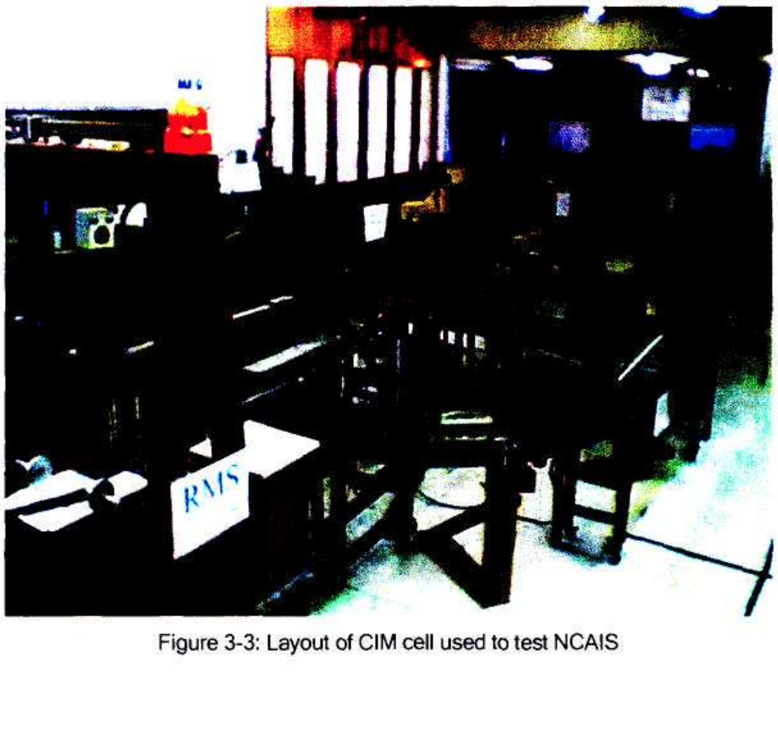


Figure 3-3: Layout of CIM cell used to test NCAIS

3.3 NCAIS operation

The NCAIS was designed to integrate into the existing CIM cell. This constrained the operation of the apparatus to the operating parameters of the cell. Among the objectives of the CIM cell, were minimisations of the user input required; and optimisation of the process involved. Consequently, the NCAIS was designed to operate with minimal user input, and with the objective of optimising inspection routines. The operation of the system was based on inputs from the user with respect to the inspection routine parameters. Parts being inspected could have either been new or reproduced. New parts required that the user enter all the necessary information about the routine. Existing parts had the possibility of allowing for changes in inspection routines. In this instance, the user would edit only the parameters that require editing. The NCAIS inspection routines were designed to have a maximum of eight user inputs for a part. These inputs were:

- Defining whether the part is new or existing
- Part family that the part belongs to
- Conveyor speed of that batch
- Part dimensions
- Faces for inspection
- ROI on the selected faces
- Orientation of the part on the conveyor
- Type of inspection required

The general sequence of operation of the inspection routine involved determining the presence of a new part by use of a line sensor; identifying the part present so that the inspection parameters could be loaded; loading and execution of the routine; obtaining part information; determining the acceptability of the inspected part based on predefined information; accepting or rejecting the inspected part. A flowchart of this operation is shown in Figure 3-4.

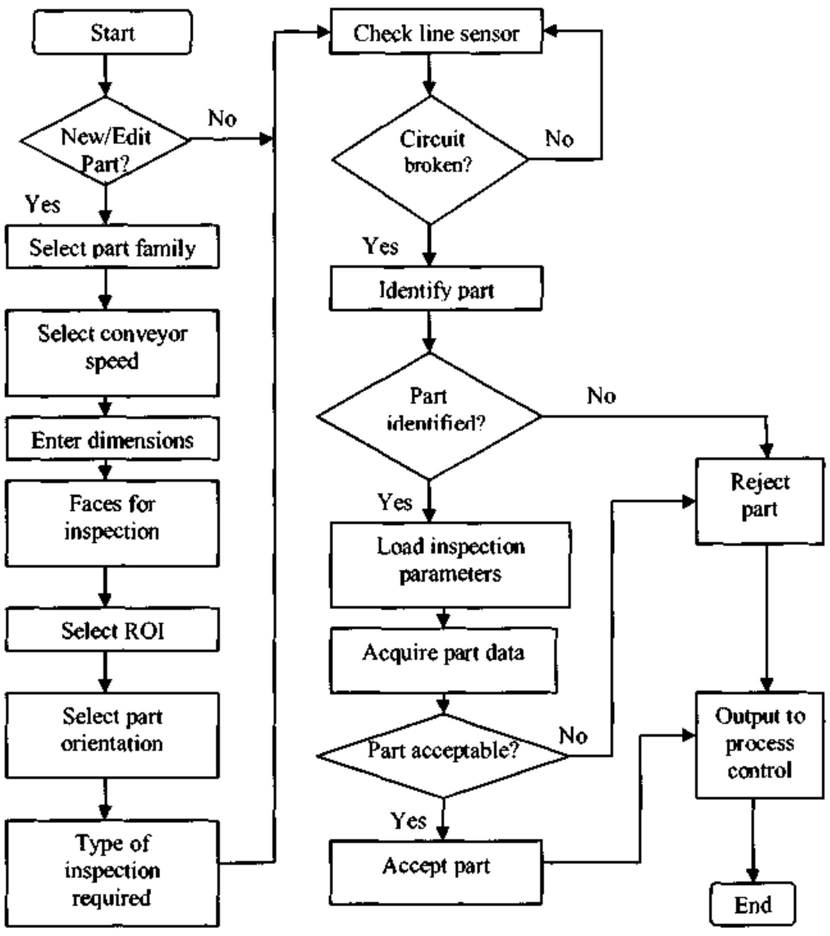


Figure 3-4: Flowchart showing proposed operation of the NCAIS

The operation of the NCAIS was dependent upon two assumptions. The first assumption was that the inspected part arrived at a known orientation. Secondly, constant conveyor speed was assumed during inspection routines. Diagnosability, within a given time limit, was the main consideration in the design of the NCAIS. The successful diagnosis of an imperfect part can inform the user as to the location of flaws in the process layout. This information can be used to tune a newly configured process, or to change the configuration of an inefficient process layout.

3.4 Classifications of parts

The concept of RME is that the manufacturing system is flexible with respect to a finite range of classified parts and part families. The inspection routines for RME should therefore account for these classes. Part families could be classified in terms of throughputs, dimensions, shape, colour, and type of inspection required. The most generic method of classification was classifying parts with respect to the shape of their volumes. Classifying parts in terms of their dimensions would facilitate parts that vary with respect to dimensions only. Changes in parameters such as part shape would not be accommodated for when classifying parts in terms of dimensions. This means that classifying parts in terms of dimensions will not be suitable for RME, as classes will need to be updated every time a change in product shape, colour, type of inspection, and throughput occurs. Throughputs of parts may change frequently. The throughput alone does not qualify as an intrinsic part property, but instead can be considered as a characteristic of the manufacturing process for that part. This opposes the classification of parts in terms of their throughputs. The colours of parts may be similar, yet the inspection routines may differ drastically. This makes classifying parts in terms of colour, ineffective for inspecting parts that vary in shape, dimensions, throughputs, and types of inspection. Classifying parts in terms of types of inspection would not be practical as changes in part shape, dimension, throughputs and colour, may not be accommodated. The most suitable method of classifying a part family within RME is to classify the parts according to the shape of their volumes. This type of classification facilitates variations in parameters such as dimensions, colour, throughputs, and types of inspection.

The NCAIS was designed to accommodate the inspection of three part families. These part families were classified in terms of the shape of their volume. The three classes of volumes were rectangular volumes, cylindrical volumes, and cubic volumes. Assigning a part to a part family was achieved by the user selecting the part family that most closely matched the actual part volume to be inspected. An example of this assignment would be assigning a cylinder head to the rectangular volume part family. Another example would be to assign a spur gear or oil filter to the cylindrical part family. The execution of the inspection routines was based on these volumes moving at a constant velocity, relative to the machine. Diagrams of the three part families are shown in Figure 3-5 (a) to (c).

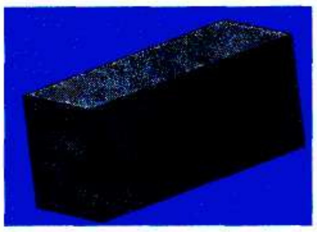


Figure 3-5(a): Rectangular Volume

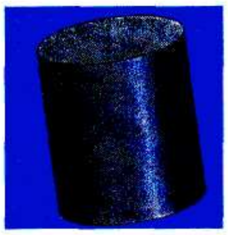


Figure 3-5(b): Cylindrical Volume

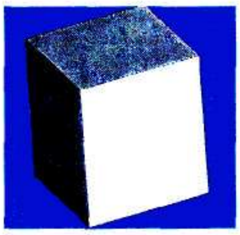


Figure 3-5(c): Cubic Part

The rectangular part family required the user to input the length, breadth, and height of the rectangular volume that the part fitted into. The cylindrical part family required the radius and height of the cylindrical volume that the part fitted into. The cubic family required the value of the side for the cubic volume that the part fitted into. Even though three part families were conceptualised, the testing of the machine was based on the rectangular part families.

3.5 Inspection strategies

Two strategies for system optimisation were considered. The first method was to inspect only significant Regions of Interest (ROI). The second method was to perform inspection of these ROI on moving parts. Not all aspects of a manufactured part require detailed inspection. It is more efficient to therefore concentrate inspection efforts toward localized regions of a part, rather than inspecting the part as a whole. The areas that require attention are referred to as ROI. Inspection of ROI allows for optimization of the inspection process by investing resources in only essential aspects that require inspection; and acquiring detailed information about localized regions as opposed to general information about the part as a whole. Mass-produced-custom products involve frequent changes in design, which results in frequent changes in inspection requirements. Reconfigurable inspection of ROI allows for facilitation of these changes in regions that require inspection. Inspection of ROI is also effective when considering the concept of Value Added Manufacturing (VAM), in which acceptable manufactured parts are used as platforms and built upon, resulting in new products with different functionalities. These value-added parts require inspection of only the additions or changes that occurred to the base product. The rectangular and cubic-volume part families were allocated a 3x3 matrix of ROI on each face, whilst the cylindrical part family was divided into eight sections about its longitudinal axis, and divided into three segments along its length, as shown in Figure 3-6. The user was allowed to inspect a face as a whole if it was required. Dynamic access to these various ROI was proposed, in order to acquire information whilst the part was moving. Since the objectives of this research were to perform inspections of only one part family, the testing and implementation of the system was to facilitate inspection of only rectangular parts.

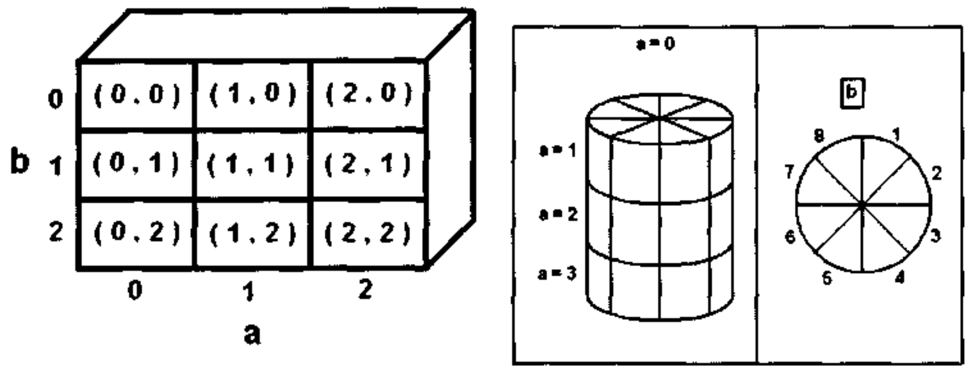


Figure 3-6: ROI for the respective part families

3.6 Conceptualized subsystems

The NCAIS incorporated use of the Mechatronic Engineering approach when subdividing the system into subsystems. These subsystems were the Sensor Positioning System (SPS), Vision System (VS), Materials Handling System (MHS), Part Identification System (PIS), and the software for general system management. The SPS was conceptualized to perform sensor articulation for dynamic access to various ROI. The VS was conceptualized to perform image acquisition and processing for obtaining information about the part being inspected. The PIS was responsible for identifying the part present, in order to load the necessary inspection routine parameters. The MHS was responsible for materials handling applications, and consisted of a conveyor belt for moving the part through the machine; as well as a Part Centralization System (PCS) for aligning the centre of the part with the centre of the machine. This was proposed for ensuring predictable part pose. Figure 3-7 shows the layout of the subsystems of the NCAIS.

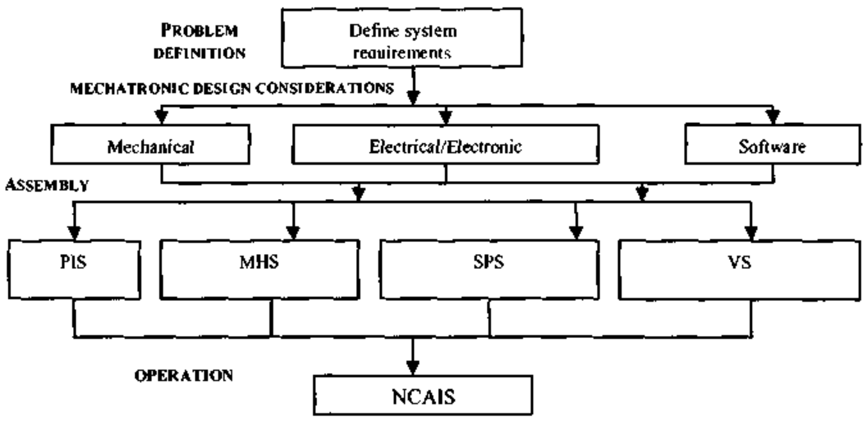


Figure 3-7: Flowchart showing constituent subsystems of NCAIS

3.7 Summary of chapter 3

This chapter describes the Mechatronic engineering approach taken in order to solve the problem. An overview of the CIM cell used as a testing environment is given. The intended method of operation for the NCAIS is also given, with the subsystems highlighted. Part families and ROI division on the different part families were discussed.

4. MECHATRONIC DESIGN

4.1 Mechanical design

4.1.1 Sensor manipulation

The inspection process was required to perform part inspection whilst reducing the impact of high frequencies of inspections on the inspection times and hence the production rates involved with mass-producing custom parts. One method of doing this was to perform the inspection routine whilst the part was moving. Inspecting a moving part required the sensor being able to dynamically access the various ROI on that part. The mechanism for positioning the sensor required:

- Accuracy
- Repeatability
- Dexterity
- Efficiency in speed
- Clean; and smooth operation

The accuracy requirements for the sensor positioning system stemmed from the fact that the vision system relied upon the centre of the ROI being located at the centre pixel of the acquired image (within some tolerance). The aligning of the image centre and the ROI centre had to take into account the precise positioning of the camera centre as a function of time. Due to the fact that the inspection process involved comparison of the inspection image and the reference image, the positioning of the sensor had to be repeatable and consistent. This involved obtaining images of the objects from the same perspective and distance, so that the image differencing operation was more reliable. The fact that the apparatus was designed for the purpose of inspecting mass-produced custom parts meant that the sensor would have been required to access various features at various locations on the part. Access to various regions of the part required dexterous motion of the sensor. This indicated that the sensor positioning system would have to be dexterous in operation, and would therefore be required to have a large work envelope. The speed of the sensor relative to the part was an essential factor when designing the system. This relative speed needed to be large enough in order to facilitate dynamic access to various ROI, and to accommodate avoidance of collisions. In order to prevent contamination of the inspected part during the inspection routine, the mechanical components of the inspection system were required to be clean in operation. The image acquisition process required that the vibrations experienced by the camera be minimised, in order to prevent blurring of pictures, resulting in a more reliable image acquisition process. This requirement meant that the motion of the camera, as it was moved from point to point, be smooth and fluent. The dexterous access to the various ROI and the speed of sensor motion were considered to be the most important factors in the design of the sensor positioning system.

There are many concepts that need to be understood prior to modelling the behaviour of mechanical members of a robot. These include workspace; types of joints; accuracy; resolution. The workspace of a manipulator is defined as the volume of space that the end-effector is able to access. The workspace of a manipulator can be divided into a reachable workspace and dexterous workspace. The reachable workspace is the volume of space that the end-effector can access in at least one configuration. The dexterous workspace is the volume of space that is accessible to the end-effector in all possible orientations. Three common types of joints used in the design and construction of industrial robots are prismatic joints; revolute joints; and spherical joints. Prismatic joints (shown in Figure 4-1(b)) are joints that allow a single degree of freedom translation along an axis. Revolute joints (shown in Figure 4-1(a)) are one-degree-of-freedom joints that allow rotation about an axis. Spherical joints are joints that allow rotation about an infinite number of axes.

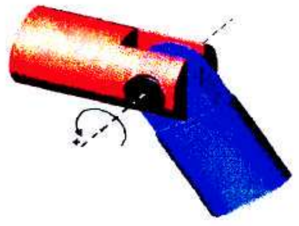


Figure 4-1(a): Revolute joint [26]

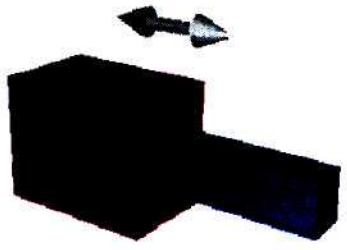


Figure 4-1(b): Prismatic joint [26]

4.1.1.1 Types of Sensor Manipulation

Robotic manipulators can be divided into Parallel Kinematic Manipulators (PKM) and Serial Manipulators [28]. The main purpose of a manipulator is to position and orientate an end-effector. The selection of a method of manipulation is dependent on the shape of the allocated workspace. Parallel manipulators generally consist of a fixed platform (referred to as the base), connected to parallel links. These links have actuated and non-actuated joints. Examples of actuated joints include prismatic and rotary joints. Universal and spherical joints are amongst the more common examples of non-actuated joints. The mechanical components in a parallel manipulator do not experience significant loading in the form of bending, as a result of their parallel architecture. This allows for lighter materials to be used, which then decreases the weight of the members. This decrease in weight allows for mechanical systems that are quicker and more accurate than mechanical systems with serial architectures. Due to the fact that the load of the end-effector is distributed amongst the parallel members, the motion capability bandwidth for the end-effector is high. The actuators for parallel manipulators can be fixed onto the motionless base. This reduces the inertia of moving members, which also contributes to the reduction in weight of moving members. The disadvantages of PKM involve limited workspace, due to collisions between mechanical members during certain configuration changes. PKM become unstable; uncontrollable; and experience loss of stiffness when accessing singular positions [27].

Serial manipulators are manipulators, which consist of a base connected to links that are connected in series. The advantages of serial manipulators are that they are able to access a large workspace with respect to their own volume [28]. The disadvantages of serial manipulators involve the low stiffness characteristics; accumulation of errors whilst operating; inaccuracies incurred by moving members with high inertia; and the relatively low effective load that they can manipulate. The low stiffness characteristic is a disadvantage due to the fact that the motion and positioning of the part may be prone to errors as a result of an open kinematic architecture. This open kinematic architecture also results in the errors induced by joint actuators, being accumulated along the chain through to the end-effector. The accuracy of the system is good but less than when compared to closed kinematic architectures. Serial manipulators require the actuators to be placed at joints. The inertia of the members is thus increased. As a result, the accuracy of the system may be influenced, and decreased. The fact that the weight of the end-effector is not distributed amongst multiple members, serial manipulators are able to only manipulate end-effectors with a low effective load. Typical serial and parallel manipulators are shown in Figure 4-2 (a) and (b) respectively. Table 4-1 shows the comparison between the characteristics of serial manipulators and PKM.

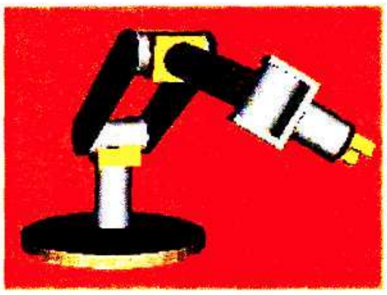


Figure 4-2(a): Serial manipulator [28]

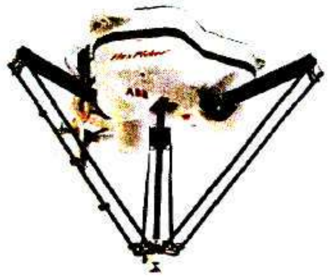


Figure 4-2(b): Parallel manipulator [27]

Characteristic	PKM	Serial Manipulator
Speed of operation	Excellent/high	Good
Load capacity	High	Low
Accuracy	Excellent	Good
Workspace	Limited	Excellent
Ease of control	Difficult	Easy
Stiffness	High	Low
Bandwidth of motion capability	Excellent	Good

Table 4-1: Comparison of PKM and Serial manipulators

Cartesian robots are robots that consist of at least three mutually perpendicular prismatic joints [29]. These robots are an industrial class of serial manipulators, although they may be considered as a combination of parallel and serial manipulators. The regional workspace of a Cartesian manipulator is a rectangular volume. Gantry robots are similar to Cartesian robots, however the difference between gantry robots and Cartesian robots are the way in which they are mounted. Gantry robots are mounted on rails, above their workspace. Cartesian robots are mounted below their workspace. The parallel aspects of the design allow high stiffness, since the axes are supported at both ends. Figure 4-3 (a) and (b) show typical Cartesian and gantry manipulators.

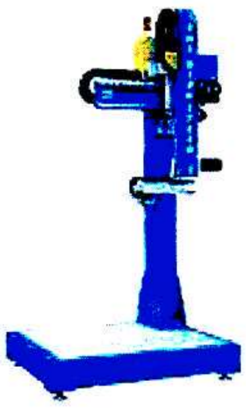


Figure 4-3 (a) Cartesian manipulator [29]

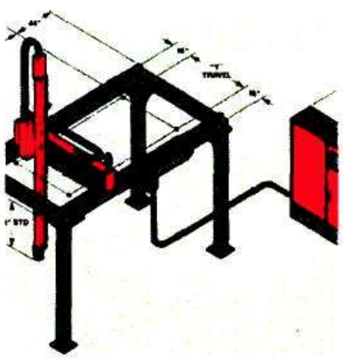


Figure 4-3(b) Gantry manipulator [30]

Gantry architectures allow for high-speed and high-accuracy operation. These structures are also generally associated with high repeatability. These are crucial characteristics when considering dynamic positioning of the camera. The control of gantry system is relatively easy, as the inverse positioning of the sensor is merely a vector subtraction operation; with no need for rotational transformation matrices.

4.1.1.2 Methods of Actuation

Actuators have the function of converting an input signal into a controlling action on a device [18]. The sensor positioning system was responsible for the accurate and repeatable positioning of the sensor. This depended largely upon the method of actuation selected for the design. Common industrial actuators can be categorised into three fields, namely pneumatic actuators; electric actuators; and hydraulic actuators.

4.1.1.2 (a) Pneumatic Actuation

Pneumatic actuation involves the use of compressed air to execute mechanical motion. This method of actuation is relatively inexpensive, and is able to perform at high speeds of operation. The process of pneumatic actuation is clean, which is important when inspecting parts that are easily contaminated. The use of compressed air as a driving force makes this form of actuation readily available. Current manufacturing buildings are commonly equipped for use of compressed air, which makes implementation of pneumatic actuators relatively easy [18]. The life expectancy of pneumatic equipment is generally high with respect to mechanical fracture, and actuators can stall without incurring any damage. The reason for this is that the compressible air will absorb any shock, due to sudden motion or stopping of motion. The disadvantages of pneumatic systems involve low accuracy and difficult speed control of actuators.

4.1.1.2 (b) Hydraulic Actuation

Hydraulic actuation involves the use of an incompressible fluid for high power transmission [33]. The incompressibility of the fluid allows for the amplification of the applied force. This results in the ability of a system to move a heavy load using a relatively smaller applied force. The advantage of incompressible fluids is that the fluid itself does not absorb any of the applied energy. This means that relatively small tubes can be used for transmitting power, which would have been an advantage when considering the spatial constraints of the NCAIS. Another advantage of using hydraulic actuation is that a very high level of servo control can be achieved, without damage whilst stalling. The disadvantages of using hydraulic actuation methods involve the possibility of fluid leaks; high initial investment; and relatively slow operation.

4.1.1.2 (c) Electrical actuation

Electrical actuation involves the conversion of electrical power into some form of mechanical motion. Electrical actuators may be categorised into switching devices; solenoid type devices; and drive systems [33]. Switching devices use an input control signal to toggle a circuit on or off. Mechanical switches; diodes; thyristors; and transistors are all types of switching devices. Solenoid type actuators use an electrical input through a solenoid in order to actuate a soft iron core. This mechanism may be used for actuating hydraulic or pneumatic flow. An example of an electrical drive actuator is the input of power to a motor in order to obtain rotary motion. The advantages of electric actuation are that they are generally fast and accurate; relatively inexpensive; generally easy to control; have regular updates in designs; low weight; high torque; and rapid response time. The disadvantages include overheating in stalled conditions; have some sort of shaft play, which limits precision; and may be prone to electrical arcing. Table 4-2 lists the different types of actuation along with their advantages and disadvantages.

Actuation methods	**Advantages**	**Disadvantages**
Pneumatic	Relatively inexpensive	Low accuracy
	High speed of operation	Difficult speed control
	Clean actuation	
	Readily available	
	Easily implemented	
	High life expectancy	
Hydraulic	Large lift capacity	Possible fluid leaks
	Very high level of servo control	High initial investment
		Relatively slow speed of operation
Electrical	Fast and accurate	Overheat in stalled conditions
	Relatively inexpensive	Experience general play on motor shafts
	Easy to control	Prone to electrical arcing
	Fast development times for new models	
	Low weight	
	High torque	
	Rapid response time	

Table 4-2: Comparison of different methods of actuation

4.1.1.3 Manipulator selection

Table 4-3 shows the extent to which the different types of actuation and manipulation meet the system requirements.

Requirement	Pneumatic	Hydraulic	Electrical	Serial	PKM
Accurate positioning	Yes	Yes	Yes	Reasonable	Yes
Fast operation	Yes	Yes	Yes	Reasonable	Yes
Able to move a light load	Yes	Yes	Yes	Yes	Yes
Easy to control	No	Yes	Yes	Yes	No
Clean	Yes	No	Yes	Yes	Yes
Cost effective	High initial investment	No	Yes	.	.
Smooth operation	Yes	Yes	Yes	Yes	Yes
Large Workspace	.	.	.	Yes	Limited

Table 4-3: Comparison of actuation and manipulation methods for meeting the system requirements

The method of actuation that best suited the requirements was electrical actuation. Since the positioning of the sensor had to be precise and repeatable, pneumatic actuators were not selected for the design of the sensor positioning system. The use of hydraulic actuation could have caused contamination to the parts being inspected. The costs involved with implementing pneumatic and hydraulic actuation were also deterrents in the selection of these methods of actuation. Due to the fact that one of the main requirements for the sensor positioning system was a large workspace, serial manipulation was chosen for the design of the sensor positioning system. The collision avoidance between sensor and part could be more easily implemented with serial manipulation than with PKM.

The final design of the sensor positioning system was the merging of a gantry system with a pan-tilt mechanism. This structure was designed to have five degrees of freedom, with three-degrees-of-freedom stemming from the gantry structure, and two-degrees-of-freedom from the pan-tilt mechanism. The combination of orthogonal-prismatic, and revolute joints in the design allow for excellent workspace and dexterity characteristics. Figure 4-4 shows a conceptual drawing of the system's motion.

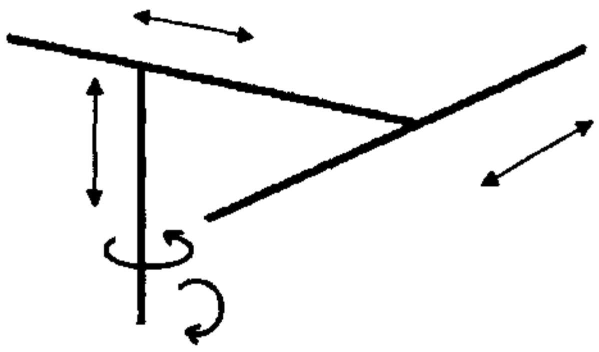


Figure 4-4: Conceptual drawing of the Sensor Positioning System motion

4.1.1.4 Kinematic model

The sensor positioning system was required to position and orientate the sensor at calculated target points. Forward kinematics is used to determine the position and orientation of an end-effector, given a sequence of joint movements. Inverse kinematics is used to determine the required joint movements to position and orientate the end-effector to a given pose. Since the motion of the gantry system was trivial subtractions, the use of intensive inverse kinematics was not considered.

The kinematic model for the manipulator was designed for precisely calculating the required position and orientation of the sensor as a function of time and ROI, for rectangular parts. This model had to account for the fact that the part was moving. The first step in the design of the model was to define three sets of co-ordinate systems. The first co-ordinate system, labelled frame O, was the global co-ordinate system, and its origin was fixed on the stationary frame for use as a global reference point. The second co-ordinate system, labelled frame B, had its origin on the part and was referred to as the part co-ordinate system. The third co-ordinate system was frame A, representing the position of the camera. Frame B was relative to frame O, and frame A was relative to frame B. The model assumed that the part co-ordinate system was moving relative to the global co-ordinate system. The purpose of defining a part co-ordinate system was to track the moving part as it went through the machine. In doing so, the ROI on the part could also be tracked as a function of time. Frame

O had its origin at the bottom-left corner of the workspace as shown in Figure 4-5 (a) and (b), with frame A being omitted.

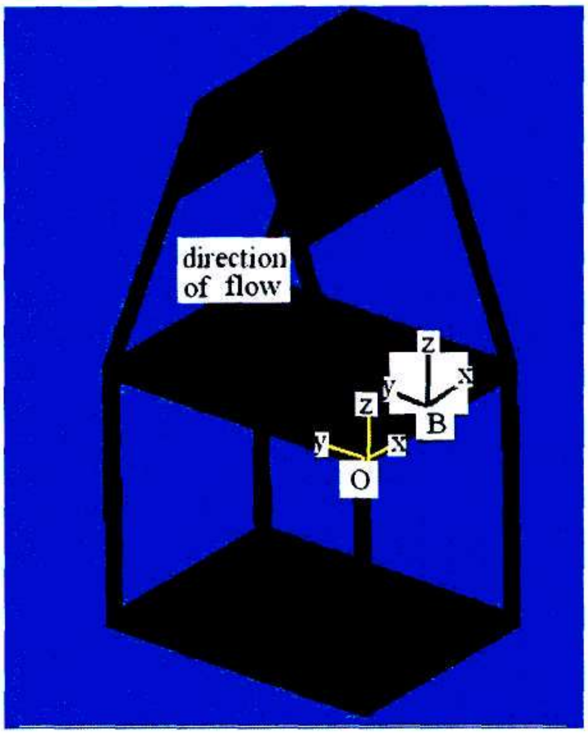


Figure 4-5 (a): Frame A and B locations

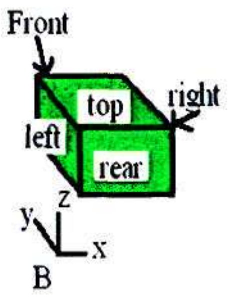


Figure 4-5(b): Part configuration

The rectangular and cubic-volume part families could only be inspected on 5 faces. The reason for this was that the bottom face was not accessible to the sensor. Two modes of operation were considered for the manipulation of the sensor in space. These were sequential positioning and point-to-point positioning. The sequential positioning of the part was based on the sensor moving parallel to each face at different divisions of time. If a ROI were encountered along the trajectory, an image was acquired. This method of manipulation was flawed in that the sensor would still move along a face even if no ROI were present. This would have lead to time wastage, which was unacceptable for these inspection routines. Figure 4-6 depicts the operation of this mode of manipulation on the rectangular volume part families. The method of manipulating the sensor from point-to-point allowed for an optimal path to be selected (discussed further on) and so this method was selected for sensor manipulation.

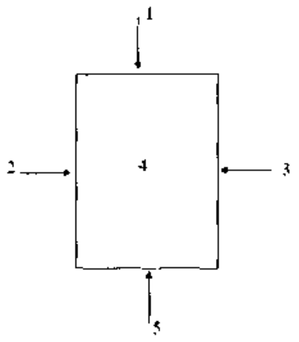


Figure 4-6: Sequential inspection for rectangular volume part inspection

Once the co-ordinate systems were defined, the user-defined ROI were converted into spatial co-ordinates, with respect to frame B. These local co-ordinates were then transformed into global co-ordinates by mapping the motion of frame B with respect to frame O. The NCAIS was designed to perform inspection on a part that had motion only in the direction of the conveyor. This meant that no lateral motion was incurred by the part. The motion of a point on the conveyor was therefore given by:

$$y(t) = y_0 + y' * t \tag{4.1}$$

where y(t) was the final position of that point on the y-axis; y_0 was the initial position along the conveyor; and $y' * t$ was the distance moved along the conveyor after a time t. Consequently, the motion of frame B relative to frame O was calculated as a function of time, using equation 4.1.

$$\underline{B}(t) = 0.5(w - a)\underline{i} + y(t)\underline{j} \tag{4.2}$$

where 'w' was the width of the manipulator workspace; and 'a' was the dimension of the volume in the x-direction. The equation accounts for the fact that the centre of the part runs along the centre of the workspace volume, in the y-direction.

The position of the centre of a ROI, with respect to frame B, was described by the vector

$${}^{B}\underline{P}_{i} = \begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} \tag{4.3}$$

where the superscript denotes the frame that the particular vector was referenced to.

The ROI were rectangles which had dimensions proportional to the dimensions of the part volume. In order to obtain the exact co-ordinates of the centres of these ROI, relative to frame B (in order to determine the components of ${}^{B}\underline{P}_i$, five cases were considered. The reason for considering five cases was to account for a ROI lying on each of the accessible faces. Assuming that the user selects a ROI that has co-ordinates (m,n) on each face of a part with volume (a,b,c), and f is the number of ROI in a row or column, the following equations apply.

Case 1: ROI lies on front face:

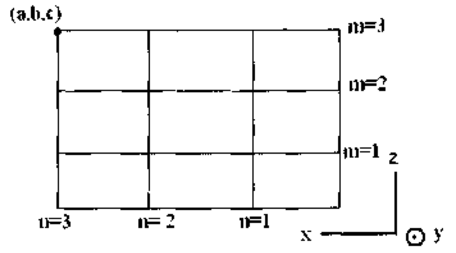


Figure 4-7: ROI on front face

$${}^{B}\underline{P}_i = (n_i - 0.5) * \frac{a}{f}\underline{i} + b\underline{j} + (m_i - 0.5)\frac{c}{f}\underline{k}$$

Case 2: ROI lies on left face

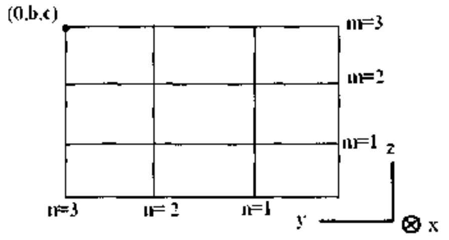


Figure 4-8: ROI on left face

$${}^{B}\underline{P}_i = (n_i - 0.5)\frac{b}{f}\underline{j} + (m_i - 0.5)\frac{c}{f}\underline{k}$$

Case 3: ROI lies on the right face

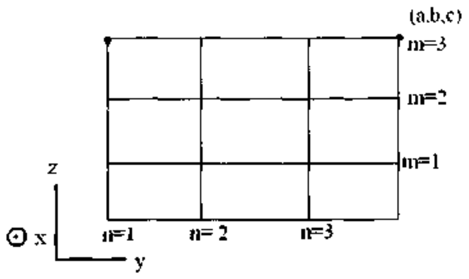


Figure 4-9: ROI on right face

$$^{B}\underline{P}_i = a\underline{i} + (n_i - 0.5)\frac{b}{f}\underline{j} + (m_i - 0.5)\frac{c}{f}\underline{k}$$

Case 4: ROI lies on top face

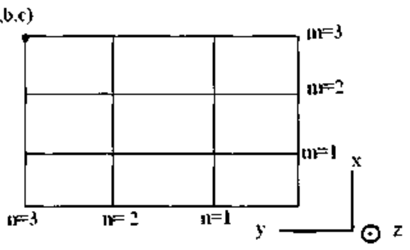


Figure 4-10: ROI on top face

$$^{B}\underline{P}_i = (n_i - 0.5)\frac{a}{f}\underline{i} + (m_i - 0.5)\frac{b}{f}\underline{j} + c\underline{k}$$

Case 5: ROI lies on rear (back) face

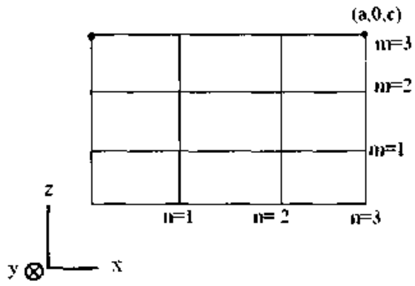


Figure 4-11: ROI on rear face

$$^{B}\underline{P}_i = (n_i - 0.5)\frac{a}{f}\underline{i} + (m_i - 0.5)\frac{c}{f}\underline{k}$$

The position of an ROI relative to frame O was then given by adding equation 4.2 and equation 2.3:

$$ {}^{o}\underline{P}_i(t) = {}^{B}\underline{P}_i + {}^{o}\underline{B}(t) \tag{4.4}$$

The set points ${}^{o}\underline{P}_i$ (i = 0…n) in equation 2.4 were considered as inspection points, and were used for predicting the position of the centre pixel during image acquisition. The focus of the camera lens was kept constant, so in order to obtain high quality clear images, a fixed distance d, between the camera centre and the inspection point, was implemented. The centre of the camera lens was placed orthogonal to the inspection point to reduce perspective errors incurred during image acquisition. Once the co-ordinates of the inspection points and directions of the normal vectors were determined, along with d, a set of target points were established. These target points were the co-ordinates at which the centre of the camera was required to be, in order to acquire an image. The target points were inherently, functions of time and ROI positions. The ith target point was determined:

$$ {}^{o}\underline{T}_i = {}^{o}\underline{P}_i(t) + {}^{B}\underline{N}_i d $$

where ${}^{B}\underline{N}_i$ was the outward normal vector relative to frame B. The vector ${}^{o}\underline{T}_i$ accounted only for the position of the camera as a function of time. The orientation of the camera was determined by considering the normal vectors for the target points. Due to the fact that inspection was assumed to have occurred on flat faces, five orientations were considered for the camera. The orientation of the camera lens facing the front face of the part as it entered the machine was taken to be the reference orientation.

With the inspection points on the following faces, the orientation of the camera was as follows:

Front face:	$orientation\_angle = 0^0$	(face the negative y-direction)
Left face:	$orientation\_angle = 90^0$	(rotate about z-axis in positive right hand screw direction, parallel to y-z plane)
Right face:	$orientation\_angle = -90^0$	(rotate about z-axis in negative right hand screw direction, lens parallel to y-z plane)
Top face:	$orientation\_angle = 90^0$	(rotate about x-axis in positive right hand

screw direction, lens parallel to x-y plane)

Rear face: $orientation\_angle = 180^0$ (rotate about z-axis in positive right hand screw direction, parallel to y-z plane)

Once a set of target points for the camera was determined, an optimised path needed to be selected. The model operated on the assumption that the trajectory from one target point to the next would always be a linear path. Consequently, the method of finding the path from point to point was to subtract the linear distance of each axis of the gantry system. The shortest linear path between two points that may be followed is given by vector S, which was given by

$${}^{o}\underline{S} = {}^{o}\underline{T}_{final} - {}^{o}\underline{T}_{initial}$$

or put into component form,

$${}^{o}\underline{S}_{i} = (x_i - x_{i-1})\underline{i} + (y_i - y_{i-1})\underline{j} + (z_i - z_{i-1})\underline{k}$$

For example, consider the centre of an end-effector with initial co-ordinates:

$${}^{o}\underline{T}_{initial} = \begin{bmatrix} 5 \\ 8 \\ 25 \end{bmatrix}$$

Suppose that we are required to move the end effector from this initial point, to a point:

$${}^{o}\underline{T}_{final} = \begin{bmatrix} 19 \\ 68 \\ 13 \end{bmatrix}$$

This yields the following translations:

$${}^{o}\underline{S} = \begin{bmatrix} 19-5 \\ 68-8 \\ 13-25 \end{bmatrix} = 14\underline{i} + 60\underline{j} + (-12)\underline{k}$$

For n target points, the number of possible linear paths was given by n!. In order to optimise the manipulator motion, the shortest overall distance from point O to point n, was considered. This allowed for selection of a path that involved the shortest inspection distance. Prior to selection of the shortest possible path, a collision detection algorithm needed to be implemented. A collision was defined as a point in the sensor spatial domain having identical co-ordinates to a point in the part volume, at the same instance of time during the inspection routine. A conservative minimum radius r was defined to represent the spherical volume of the camera, to account for all possible orientations of the camera. Thereafter, mathematical collision testing for each segment of a path was performed. If any segment of the path collided with the part, the path was rejected. A collision was defined as the camera being closer to any region of the part than the clearance r. The collision detection procedure depended on the part family and the dimension of the part. The camera was limited to a piecewise-linear path and so any single segment of a piecewise-linear path could be modeled parametrically as a ray with the parameter t, shown in equation 4.5.

$$r(t) = r_0 + r_d t \quad (0 < t < || r1 - r0 ||) \tag{4.5}$$

where r0 = x0 i + y0 j + z0 k was the starting point of the move and r1= x1 i + y1 j + z1k was the end point. The unit-vector ray-direction rd was defined, shown in equation 4.6.

$$r_d = \frac{(r_1 - r_0)}{\|r_1 - r_0\|} = x_d i + y_d j + z_d k \tag{4.6}$$

The intersection of such a ray with the part was determined by solution of simultaneous equations. For example, given the ray r (t) and the plane x = c (for a rectangular part), intersection point $r_a = x_a i + y_a j + z_a k$ could then be calculated by solving for t_c (collision time) in the following equation (always taking positive values of t_c only).

$$x_a = c = x_0 + x_d t_c \Rightarrow t_c = \frac{(c - x_0)}{x_d}$$

Using similar methods, collisions in the y and z axes were checked for. If a time existed where a collision occurred, the path was rejected.

Even though the research was aimed at rectangular part families, an algorithm was devised for facilitating inspection of cylindrical part families. The upright-cylindrical case was treated similarly to the rectangular-prismatic case. The cylinder equation was a function of the height (H) and radius (r) of the part, shown in equation 4.7.

$$x^2 + y^2 = (R + r)^2 ;\ 0 < z < (\mathrm{H} + \mathrm{r}) \tag{4.7}$$

where the addition of r to R was implemented to account for the cylinder radius plus the camera spherical radius. Collision detection was then performed by obtaining equation 4.8, equation using equation 4.7.

$$(x_0 + x_d t_c)^2 + (y_0 + y_d t_c)^2 = (R + r)^2 \tag{4.8}$$

Equation 4.8 may be re-written in the form

$$A t_c^{\,2} + B t_c + C = 0$$

where

$$A = (x_d^{\,2} + y_d^{\,2});\ B = 2(x_d x_0 + y_d y_0);\ C = x_0^{\,2} + y_0^{\,2} - (R + r)^2$$

The solution of the system was given by equation 4.9.

$$t_c = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \tag{4.9}$$

This approach required that an acceptable solution existed only if A was nonzero; the discriminant *(B^2 – 4AC)* was positive; and if a positive solution was found. Meeting these requirements, t_c was back-substituted to give x_a, y_a and z_a. It was also checked to see if z_a lay within the interval *[0, H + r];* and the plane $z = H + r$ was also intersection-tested with the ray as described earlier. The point of intersection, if any, was checked to determine if it lay within a radius of *(R + r)* of the local origin in the x-y plane. If any one of these conditions was satisfied then it was assumed that the path was prone to collision.

4.1.1.5 X-axis design

The mechanism facilitating motion in the x-direction was required to transport the load induced by the weight of the y-axis mechanism; the z-axis mechanism; the pan and tilt mechanism; and the sensor. The gantry setup required that motion in the x-direction be supported at two ends, with the y-axis mechanism sliding between these two supports. Use of a gantry system also required that the structure be placed above the part. Consequently, the structure had to be mounted onto the frame, which was at a constant slope of 70 degrees to the conveyor. In order to simplify the assembly of the structure, adapter brackets were designed to provide a vertical surface on which the structure could be mounted. These brackets were designed as bracket pairs which mounted on either side of the bar of the frame.

Eight mountings were designed for two points of support on either end of the x-axis structure. These mounting brackets were designed to house the leadscrew and support bars, whist being secured onto the adapter brackets. This design was intricate as many holes had to be included in a small space, in order to facilitate fastening. Two of these mountings were designed for support on either end of the x-axis on the slider side of the gantry mechanism. An x-axis motor bracket was designed for mounting the drive motor onto the frame. Two x-axis support bars were designed from stainless steel. The x-axis slider was responsible for motion of the sensor in the x-direction. This slider consisted of a brass nut, which provided the motion, fixed onto a frame used for support of the y-axis structure.

Motion in the x-axis was driven by a motor attached to a leadscrew mechanism. Due to the high cost of ball screws, threaded bar was used to convert rotational motion into translational motion. The mechanism was then also referred to as a power screw. The geometry of a typical screw thread is shown in Figure 4-12.

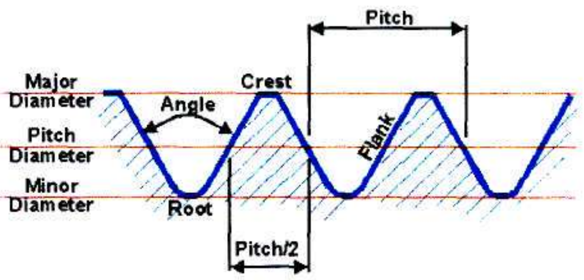


Figure 4-12: Geometry of screw threads [34]

The major diameter of a thread is the cross-sectional distance between the outer ends of the thread, on opposing sides of the longitudinal axis of the bar. The cross-sectional distance between the roots of threads, is referred to as the minor diameter. The pitch diameter is half the sum of the inner and outer diameters. The pitch is defined as the distance, measured parallel to the longitudinal axis, between corresponding points in the same direction of adjacent thread surfaces, in the same axial plane [34].

The outer diameter of the threaded bar selected was 18mm and was made from high tensile steel. In order to prevent warping of the bar during operation, the drive mechanism included two support bars. These support bars lay on the same plane with their centres parallel to each other. In addition to vertical load distribution, these support bars also provided a rotational reaction force, preventing rotation of the x-axis slider. Figure 4-13 (a) shows the designed x-axis mechanism with the enlarged view of the slider in Figure 4-13 (b). The threaded bar is shown in green in both figures.

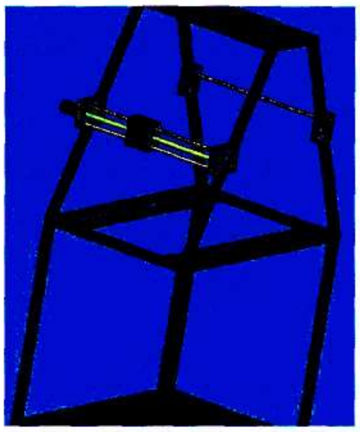


Figure 4-13 (a) Designed x-axis mechanism

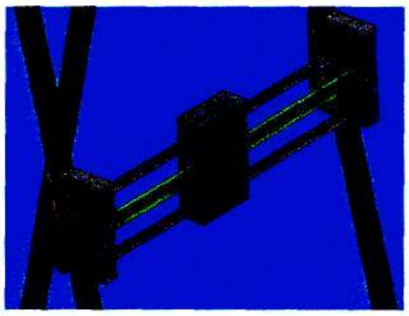


Figure4-13 (b) enlarged view of the x-axis slider

Figure 4-14 (a) shows the constructed x-axis components, and Figure 4-14(b) shows the mounting of the x-axis onto the NCAIS frame.

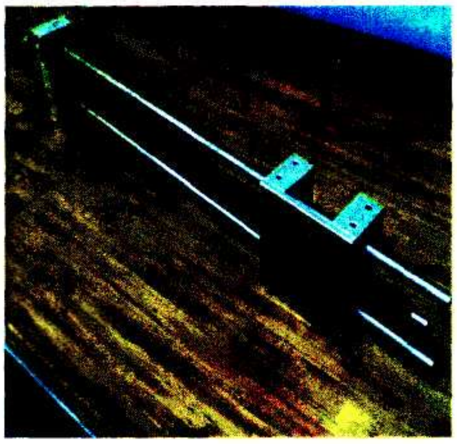


Figure 4-14(a) Constructed x-axis subassembly

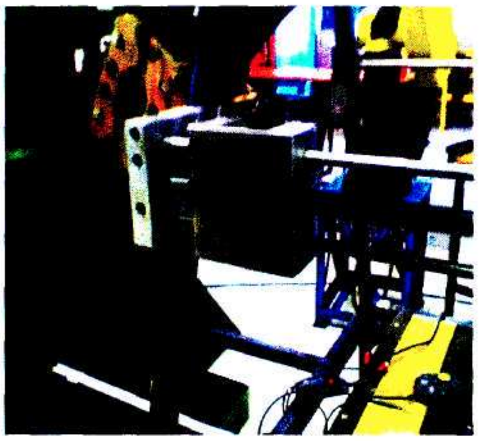


Figure 4-14(b) x-axis mounted onto frame

The height at which the y-axis slider was placed influenced, and was influenced by, the width and height of the part that could be inspected. The positioning of this y-axis slider also influenced its length. A height of 460mm was selected for positioning the centre of the y-axis mechanism. Since the y-axis mechanism was fixed on the x-axis slider, the top surface of the x-axis mechanism was located at 440mm above the conveyor surface. The difference in heights between the adapter brackets was to accommodate the positioning of the rollers.

4.1.1.6 Y-axis design

The motion in the direction of the y-axis was one the main focuses in the design of the SPS. The reason for this was that the part only had a velocity in the y-direction. The motion in this direction had to be faster than the maximum conveyor speed in order to ensure that all ROI were dynamically accessible. The y-axis drive system was subject to loading conditions induced by mass of the z-axis mechanism and the sensor. This drive system was supported at either end of the frame. One side was supported by the x-axis slider. This was done to allow for vertical support; and motion of the drive system in the x-direction. The opposite side required support in the vertical direction, and minimal resistance to motion in the x-direction. Roller supports were selected for this application. The y-axis support bar, threaded bar, and y-axis slider were all similar to those designed for the x-axis. The design of the y-axis structure including all components of the x and y-axes is shown in Figure 4-15(a) and the constructed y-axis subassembly is shown in Figure 4-15(b).

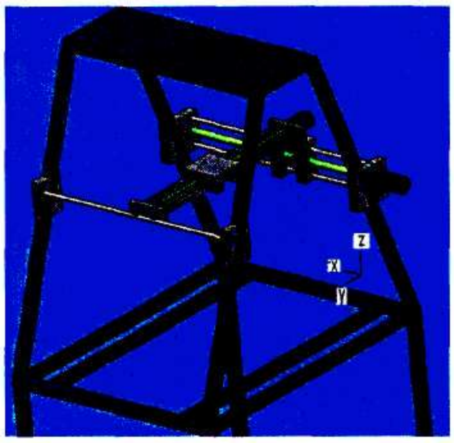


Figure 4-15(a): Y-axis design

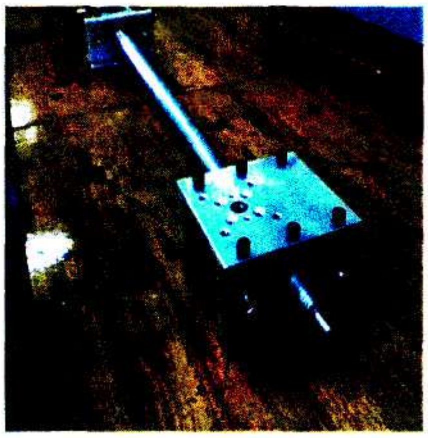


Figure 4-15 (b): Constructed y-axis structure

4.1.1.7 Z-axis design

The z-axis drive system was responsible for increasing and decreasing the height of the sensor. Using the height of the y-axis structure within the trapezoidal volume, the components of the z-axis drive system were designed. The constraints placed on the z-axis structure were due to the c-shaped tack. The design of the z-axis structure was different to the x and y-axes structures since the z-axis structure could only be supported at one end. The reason for this was essentially that the sensor would be positioned on the pan-tilt mechanism which lay on the end of the z-axis structure. The problem experienced with this requirement was the location of the fixed reference point which was used for motion in the z-axis. Three possibilities were considered for the design of this structure.

The first was to have the camera fixed onto the leadscrew, and move the drive system vertically. The problem with this concept was that any motion above the gantry structure may have resulted in a collision between the drive system and the frame of the apparatus. The second concept was to fix the motor onto the y-axis slider, and attach the camera to move along the leadscrew. The problem with this concept was that the pan-tilt mechanisms would not be able to function properly, which would drastically reduce the sensor access to various ROI. The third concept explored was to fix the camera onto the y-axis slider, and design a slider extension mechanism for the z-axis. This extension mechanism would allow for a pan-tilt mechanism to be integrated into the design.

The components, except steel leadscrew and stainless steel support bars, designed for the z-axis motion used Aluminium as the selected material. A motor mounting was designed to fix the motor onto the y-axis slider. The positioning of the motor had to account for the support bars and the leadscrew of the y-axis structure. A cylinder mounting bracket was designed to fix the cylinder onto the y-axis slider mechanism. The design of this component had to consider limited space between the components of the slider. Aluminium was the selected material for this design. The shape of this component had to account for the diameter of the cylinder and the maximum allowable diameter within the slider mechanism.

A cylinder was designed for providing a fixed reference structure on which the slider extension could move along. Slots in this cylinder were designed to prevent rotational motion of the camera due to motion of the z-axis leadscrew. The slider-extension mechanism was responsible for moving the sensor vertically as a result of the rotation of the leadscrew. This mechanism was designed to consist of a brass nut; four support rods; and a base plate. The brass nut was used for moving up and down the fixed leadscrew. The support rods were rigidly attached to the brass nut through the slots of the cylinder at one end, and to the base plate at the opposite end. This allowed the base plate to follow the same motion as the brass nut, at a fixed distance from the nut. The constructed z-axis components are shown in Figure 4-16(a) and the 3D design of the fully assembled structure is shown in Figure 4-16(b).

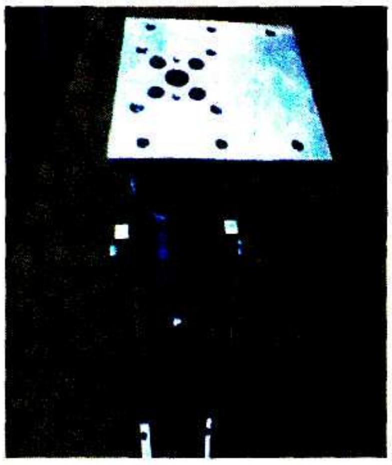


Figure 4-16(a): Constructed Z-axis components

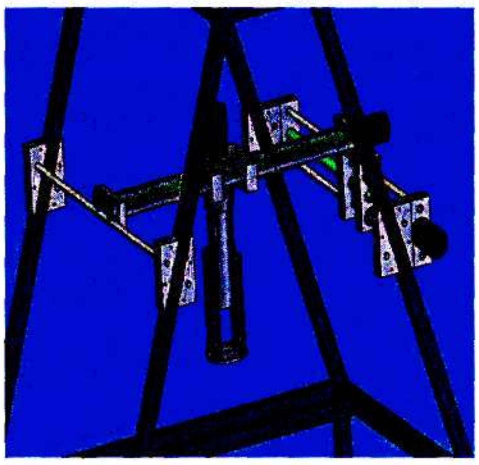


Figure 4-16(b): Full 3D gantry structure design

4.1.1.8 Pan-tilt mechanism design

The pan and tilt mechanism was designed to enable change in sensor orientation. This mechanism was fixed at the base of the z-axis slider extension. Due to the fact that the camera did not impose a heavy weight load, aluminium plating (2mm thick) was chosen for this structure. The constructed structure is shown below. This structure facilitated rotation about the z-axis and rotation about axes lying in a horizontal plane. The assembled components of this design are shown in Figure 4-17.

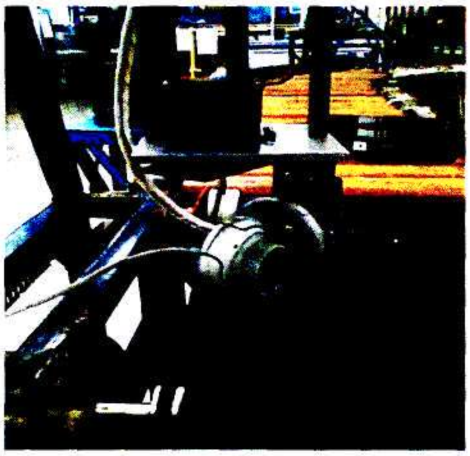


Figure 4-17: Assembled components of the pan-tilt mechanism

4.1.1.9 Static analysis

A static analysis of the sensor positioning system was performed to determine the structural integrity of the members under static loading conditions. Structural failure of the system due to static loading could have occurred as a result of the following modes of failure:

- Shear failure of members
- Plastic deformation of members due to bending
- Tensile failure of members

Each mode of failure was considered, depending on the loading conditions of different members. Deflections and stresses were calculated to justify the dimensioning and material selection of the diameter of the support bars and threaded bar for the x and y-axes members.

Simple calculations were also performed to verify the design of the z-axis mountings and the pan-tilt mechanism.

4.1.1.9 (a) X-axis

The static loading conditions of the x-axis involved the weight of the members of the x-axis; y-axis; z-axis; pan-tilt mechanism; and sensor. This loading was distributed among the slider of the x-axis drive system and the roller on the opposite end, shown in Figure 4-18. This weight distribution was most uneven when the load was at either end of the y-axis. To incorporate some form of safety factor, the analysis used the entire weight as the loading condition, for both the slider and the roller ends.

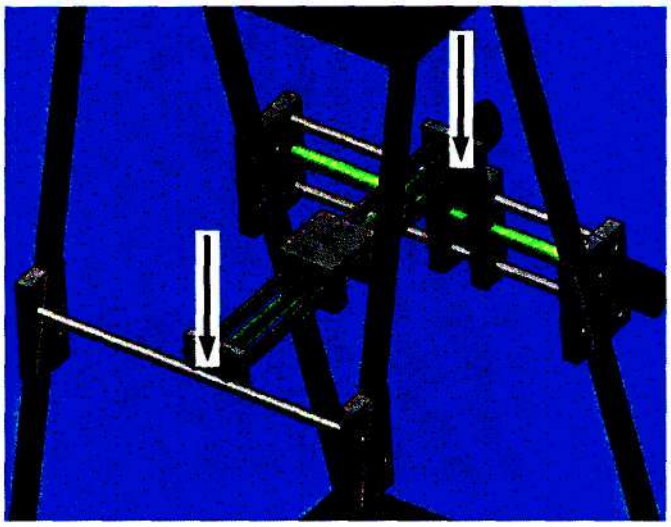


Figure 4-18: Vertical Loading on x-axis

The slider end of the x-axis consisted of three parallel members rigidly fixed at each end. Two stainless steel rods were used as support bars. These bars each shared the vertical load induced. The support bars each had two contact points with the slider mechanism. The threaded bar was subjected to only the axial load induced by the friction involved with moving the load. This loading condition is shown in Figure 4-19.

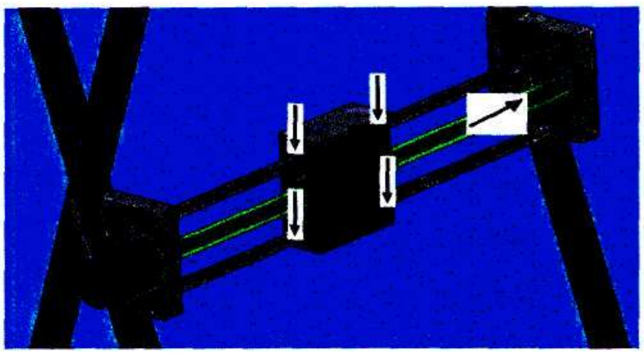


Figure 4-19: Forces on x-axis bars

The axial loading of the threaded bar was determined using equation 4.10 [35].

$$\sigma = \frac{\mu W}{A} \tag{4.10}$$

The numerator in equation 4.10 is the frictional force to be overcome in moving the load. Using a typical value for the frictional coefficient μ as 0.2 [40] whilst considering the diameter of the bar to be 18mm and estimating the load to be 30N, the axial force initiated in the bar was found to be 0.11MPa. This value was negligible and so no further axial calculations were performed for the y-axis, as this axis had a smaller load to carry. The threaded bar was also analyzed for failure due to buckling. The critical load for buckling of the threaded bar was found using equation 4.11 [59].

$$P_{cr} = \frac{\pi^2 EI}{L^2} \tag{4.11}$$

Substituting E=200GPa, I=$1.013 * 10^{-8} m^4$, and L=0.55m into equation 4.11, the critical load for buckling of the threaded bar was found to be approximately 67kN which was well above the applied axial load ($\mu W = 6N$). No buckling analysis was performed on the y-axis threaded bar since this bar had a smaller load. The design was thus found to be safe against failure due to buckling.

The support bars were analyzed as having two point loads, a fixed distance apart, and a UDL due to the weight of the bars. These support bars were analyzed as statically indeterminate beams, as shown in Figure 4-20. The loading conditions for each support bar, including the bar on the opposite end of the y-axis were identical under maximum loading conditions and so analysis of one support bar was sufficient. The point load was due to the weight of the members of the rest of the system. The UDL was due to the weight of the bars themselves. The worst loading condition involving bending occurred when the slider was placed at the centre of the beam, causing the greatest overall deflection and moment.

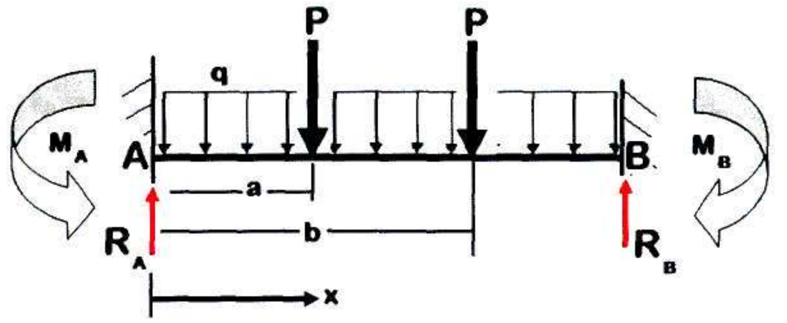


Figure 4-20: Loading conditions for the support bars of the x-axis

The loads P were considered to be equal, and the conditions for each of the support bars were considered as identical due to the even load distribution between these two structures. Using McCauley's method [37], the moment of a point on the beam was modelled to be:

$$EI\frac{d^2y}{dx^2} = -M_A + R_Ax - \frac{qx^2}{2} - P(x-a) - P(x-b) \tag{4.12}$$

Integrating equation 4.19 yields the slope of the deflection being given by equation 4.13.

$$EI\frac{dy}{dx} = -M_Ax + \frac{R_Ax^2}{2} - \frac{qx^3}{6} - \frac{P(x-a)^2}{2} - \frac{P(x-b)^2}{2} + A \tag{4.13}$$

The deflection of the support bars was then modelled by further integration of equation 4.13, resulting in equation 4.14.

$$EIy = -\frac{M_Ax^2}{2} + \frac{R_Ax^3}{6} - \frac{qx^4}{24} - \frac{P(x-a)^3}{6} - \frac{P(x-b)^3}{6} + Ax + B \tag{4.14}$$

The boundary conditions were used to find the constants of integration and these constants were calculated to be zero (the method for calculating the values of these constants can be found in [36]). Using these equations, the reactions for the system were calculated as shown in equation 4.15 and equation 4.16 respectively.

$$R_A = \frac{ql}{2} + \frac{P(l-a)^2(l+2a)}{l^3} + \frac{P(l-b)^2(l+2b)}{l^3} \tag{4.15}$$

$$M_A = \frac{ql^2}{12} + \frac{Pa(l-a)^2}{l^2} + \frac{Pb(l-b)^2}{l^2}$$

(4.16)

With the centre of the slider being in line with the middle of the beam, the value of a in metres was then given by equation 4.17.

$$a = \frac{l}{2} - 0.05 \tag{4.17}$$

The distance b was represented by equation 4.18.

$$b = \frac{l}{2} + 0.05 \tag{4.18}$$

Substituting equations 4.15, 4.16, 4.17, 4.18 into equation 4.14 resulted in the maximum deflection and moment represented by equation 4.19 and 4.20 respectively.

$$y_{x=\frac{l}{2}} = -\left(\frac{ql^4}{384EI} + \frac{P\left(\frac{l}{2}+0.05\right)^2 (l-0.2)}{48EI} + \frac{P\left(\frac{l}{2}-0.05\right)^2 (l+0.2)}{48EI} + \frac{2.08*10^{-5}}{EI}P \right) \tag{4.19}$$

$$M_{x=\frac{l}{2}} = \frac{1}{EI}\left(\frac{ql^2}{24} + \frac{P\left(\frac{l}{2}+0.05\right)^2}{2l} + \frac{P\left(\frac{l}{2}-0.05\right)^2}{2l} - 0.05P \right) \tag{4.20}$$

Using equations 4.19 and 4.20, the following values shown in Table 4-4 were calculated for the static loading of the x-axis.

	Deflection (mm)	Moment (Nm)	Bending Stress (kPa)
x-axis support bars	0.28	$15.9*10^{-3}$	162

Table 4-4: Calculated values of static analysis at x=L/2

It is important to emphasise that the calculated values in the above table assumed that the entire load was acting on each member. The two point loads on the support bars were assumed to be 15N each, as a result of symmetry. The loads on the support and threaded bars were unevenly distributed, and the deflections of these bars would have to be equal due to the slider being rigid. The calculation therefore assumed a worst case loading condition in which the maximum loads were supported by each bar alone.

The methods of failure analysed for the x-axis mechanism were:

- Failure due to excessive bending stresses when the slider was at the centre of the structure
- Shear of the bars due to the slider being at one end of the beam

Analysing failure of the system due to bending stresses involved calculating the maximum moment for each bar and then used in the equation 4.21 [35] to determine the induced bending stress.

$$\sigma_{bending} = \frac{M\bar{y}}{I} \tag{4.21}$$

This induced bending stress was found to be 2.6kPa. This value was compared to the yield stress of the material, which was 290MPa [37]. The induced stress was much less than the yield stress and tensile stress of the material, and so the design was deemed safe against failure due to bending stresses. The method of failure due to shear was performed using the reaction forces calculated when the slider was at the end of either side of the beams. These reaction forces were then substituted in equation 4.22 [35].

$$\tau = \frac{F}{A} \tag{4.22}$$

The induced shear stress was then calculated and compared to the maximum allowable shear stress for the material. The calculated shear stress was 0.12Mpa for the threaded bar and 0.19MPa for the support bars. This was well below the allowable shear stress value of 330MPa and 186MPa respectively [37]. The design was then considered safe against shear loading. Since the loading conditions caused such minimal stresses, the method of failure due to crushing of members was disregarded.

The adapter brackets used three steel bolts, 6mm in diameter, to secure the x-axis structure onto the frame of the apparatus. The worst case of loading for these brackets was when the slider mechanism was on either end of the x-axis. A maximum reaction force of 30N was assumed. The method of failure predicted for this loading was due to shear of the three bolts. The loading condition is shown in Figure 4-21.

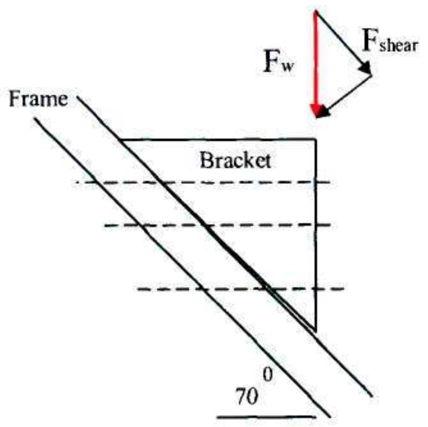


Figure 4-21: Shear loading of bolts for x-axis mounting bracket

The shearing force was calculated using equation 4.23.

$$F_{shear} = \frac{F_{Weight}}{\left(1+\frac{\tan 20}{\tan 70}\right) * \sin 70} \qquad (4.23)$$

The shearing force was calculated to be 28.2N. This value was then used to calculate the shear stress involved using 4.29. The induced stress was calculated to be 1.4MPa, which was under the allowable shear stress of 200MPa [39]. All calculated values confirmed that the induced stresses were within the allowable stresses for the design, and this structure was safe against failure due to static loading.

4.1.1.9 (b) Y- axis

The y-axis loading conditions were identical in layout, to the slider end of the x-axis structure. The only difference was due to the actual weight supported by the structures. The y-axis mechanism had to support the weight of its own members; the z-axis structure; the pan-tilt structure; and the sensor. Using the same methods to verify the design of the slider end of the x-axis structure, the following values were calculated, which verified the design of the y-axis structure.

	Deflections (mm)	Moment(Nm)	Bending Stress(kPa)	Shear Stress(kPa)
y-axis support bar	0.19	$12.2*10^{-3}$	124	480

Table 4-5: Calculated values of static analysis at y=L/2

As explained for the x-axis analysis, these values assumed that the entire loading was supported by each member alone. The results all confirmed that the design was safe under static loading.

4.1.1.9 (c) Z- axis

The z-axis structure was mounted onto the y-axis slider, and so the only load on the z-axis structure was the weight of the pan-tilt mechanism. The force induced by the camera and components was measured to be 10N. This load was used to calculate the tensile stress in the shaft, which was calculated to be 0.35MPa. Since this value was well below the elastic limit, no further analysis was performed.

4.1.1.9 (d) Pan-Tilt mechanism

The pan-tilt mechanism was subjected to bending stresses; with the pan motor shaft experiencing tensile loading, and the tilt motor shaft experiencing bending stresses due to the sensor. The bending stress on the structure was induced by the weight of the tilt motor and sensor. The loading conditions for this system are shown in Figure 4-22. The tensile force was estimated as 8N; F1 was estimated to be 5N; and F2 was estimated as 2.5N. Due to the small loads and high rigidity of the plate used to house the tilt motor, the deflection of the plate was not analyzed.

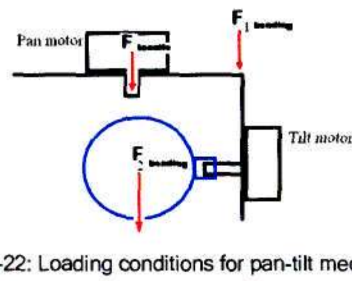


Figure 4-22: Loading conditions for pan-tilt mechanism

The following values, shown in Table 4-6, were calculated for this system. The results all confirmed that the design was safe under static loading

Force	Associated Stress
Tensile	1.1MPa
F1	17.1MPa
F2	8.8MPa

Table 4-6 Calculated loading conditions for pan-tilt mechanism

4.1.1.10 Dynamic analysis

A dynamic analysis was performed on the SPS to determine the maximum torque and speed required to drive the x; y; and z-axes. This model considered the motion of the sensor independently, as a point to point motion along each axis. The loading conditions reflected to the drive motors of the system were due to inertial and frictional loading. The most crucial aspect of the motion of the SPS was the sensor velocity in the y-direction. The reason for this was that the part had only a y-velocity component, and the greatest overall distance would have to be travelled by this axis. It was decided that the sensors' lowest maximum y-velocity be equal to a parts maximum y-velocity. The y-axis drive system was therefore constrained to be able to operate at a velocity of at least 20mm/s. The required motor speed was dependent on the lead of the screw and the mass that was being moved. Figure 4-23 illustrates the schematic layout of the drive system for the x; y; and z-axes of the SPS.

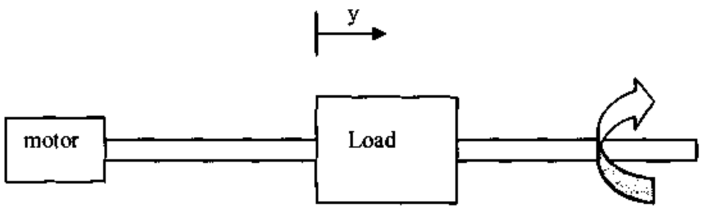


Figure 4-23: Schematic of the drive system for the axes of the SPS

Using the parameters of the leadscrew, with n being the number of threads per revolution and p being the pitch of the threaded bar, the lead of the system was then calculated using equation 4.24 [40].

$$l = np \tag{4.24}$$

The relationship between the rotational and translational velocities was then calculated using equation 4.25 [40]:

$$\dot{y} = l\,N \tag{4.25}$$

where N was the rate of rotation measured in revolutions per second. Using a thread pitch of 3.5mm on a single-thread leadscrew, and a translational velocity of 20mm/s, the required maximum angular speed for the system was calculated to be approximately 5.7 revolutions per second (342 rpm). The selected motor had to be able to operate at a speed of 342 rpm. The torque required to drive of the system was a function of the inertial torque of the leadscrew, and the torque required to move the load. The accelerating torque required for the system was described in terms of the total system inertia; the angular acceleration; and the load friction, using equation 4.26[40]

$$T_{acceleration} = \frac{2\pi J_{total} N}{60t} + T_f \tag{4.26}$$

The friction torque T_f was dependent on the weight (N) of the load being driven, the pitch (p) of the leadscrew; the efficiency (e) of the leadscrew, and the coefficient of friction between the leadscrew and the nut μ. This relationship was described using equation 4.27 [40].

$$T_f = \frac{\mu W p}{2\pi e} \tag{4.27}$$

The total inertia was the sum of the inertias of the load and the leadscrew shown in equation 4.28 and 4.29 [40].

$$J_{load} = \frac{m_{load}}{e}\left(\frac{p}{2\pi}\right)^2 \tag{4.28}$$

$$J_{leadscrew} = \frac{m_{leadscrew} R^2}{2} \tag{4.29}$$

The values for the inertial and frictional loading, along with the component properties for each axis, are shown in Table 4-7. The efficiency and friction coefficient of a leadscrew with lubrication was listed as 0.55 and 0.15 respectively [40]. Setting the minimum acceleration time to be half the minimum move distance of 30mm at maximum velocity (0.02m/s), this acceleration time was calculated to be approximately 0.75s. Using this time and a maximum required angular speed of 342 rpm, the following values were obtained when substituting in equations 4.27, 4.28, and 4.29.

	Efficiency	Load(N)	$T_{friction}$ (Nm)	J_{load} (kg-m^2)	$J_{leadscrew}$ (m^4)	Max. Torque required (Nm)
x-axis	0.55	30	$4.56*10^{-3}$	$1.69*10^{-6}$	$4.86*10^{-5}$	$6.96*10^{-3}$
y-axis	0.55	24	$3.65*10^{-3}$	$1.35*10^{-6}$	$4.86*10^{-5}$	$6.04*10^{-3}$
z-axis	0.55	10	$8.68*10^{-4}$	$1.84*10^{-7}$	$5*10^{-6}$	$1*10^{-3}$

Table 4-7: Calculated values of dynamic loading

4.1.1.11 Vibration analysis

A vibration analysis was performed for the x and y axes to determine the significance of the deflection caused during motion of the camera. This deflection would have contributed to errors in the positioning of the sensor at the time of image acquisition. Energy methods were used to determine the amplitude of the deflection. The analysis was applicable to both the x and the y-axis operation, using velocity and static deflections as parameters. The systems were modelled as a mass on a beam, with the axis slider being the mass and the leadscrew as the beam. The dynamic deflections were then in the vertical direction. Using v as velocity and δ as deflection, the energy of the system whilst in motion was modelled as the sum of the kinetic (due to motion) and potential (due to deflection) energies. This was given by equation 4.30 [41].

$$KE + PE = \frac{mv^2}{2} + \frac{k\delta^2}{2} \tag{4.30}$$

The energy present at the time that the slider was stopped was modelled using equation 4.31. k represented the stiffness of the system and X_o was used to represent the amplitude of the vibration.

$$KE + PE = \frac{k(X_o + \delta)^2}{2} \tag{4.31}$$

Using the principle of conservation of energy [41], and taking the stiffness of the system to be $k = \frac{mg}{\delta}$ [41], equations 4.30 and 4.31 were equated to yield equation 4.32.

$$X_o = \sqrt{\frac{\delta\left(v^2 + g\delta\right)}{g}} - \delta \tag{4.32}$$

Figure 4-24 shows the results from a simulation performed in Matlab which was used as a vibration analysis of the x-axis. The maximum value of the dynamic deflection was calculated to be approximately 0.83mm. This calculated value was the deflection that would have been experienced by the system if the stopping procedure was instantaneous. The value obtained was well within the allowable deflection tolerances.

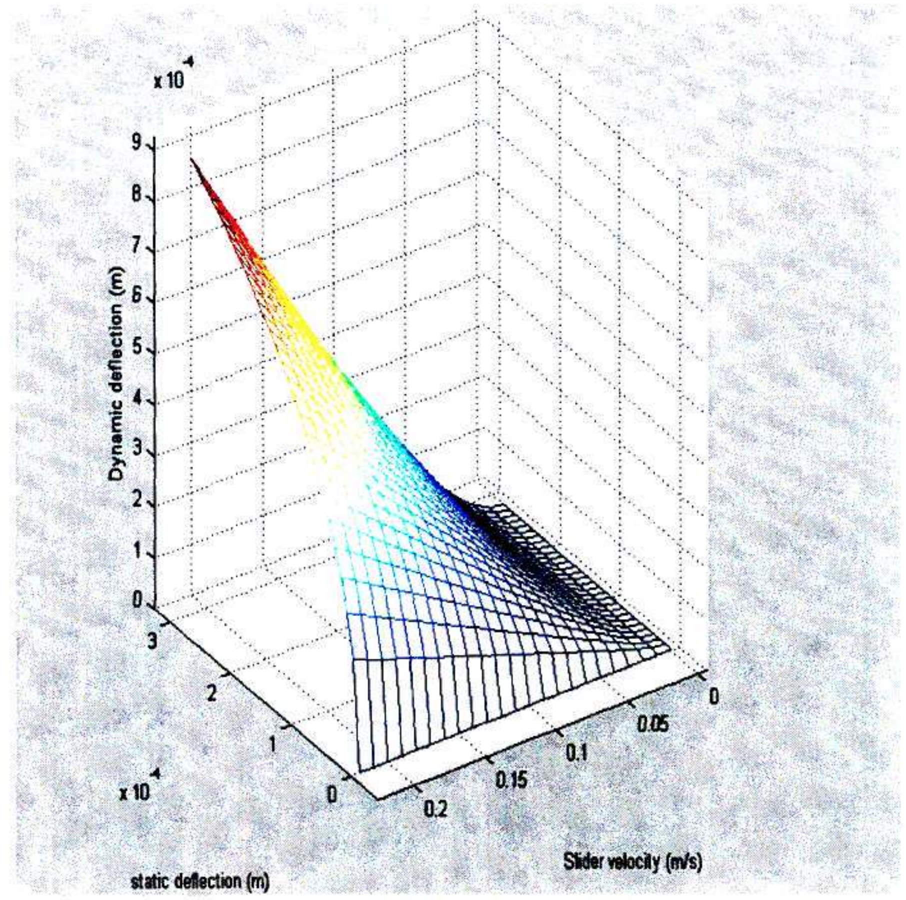


Figure 4-24: Dynamic x-axis deflection simulation in Matlab

A similar simulation was performed on the y-axis and the value for the dynamic deflection was calculated to be 0.72mm, which was also within the acceptable limit.

The critical speed of a shaft can be defined as the speed of the shaft at which harmonic vibration occurs. This speed for the leadscrew drive shafts were calculated using equation 4.33 [39].

$$critical\ speed = \frac{(1.21*10^8)*d}{L^2} \tag{4.33}$$

The critical speed was calculated to be 7195 rpm. This was well above the operating capacity of the motors, and so no further investigation was performed regarding vibration effects.

4.1.2 Materials handling system

The materials handling system was responsible for facilitating motion of the part during an inspection routine. The two subsystems designed for materials handling purposes were the PCS and CS.

4.1.2.1 Conveyor system

The conveyor system was required to move the part at a constant velocity during its motion through the machine. This motion had to be precise and consistent in order to synchronize the positioning of the sensor and the part at the required time intervals. The previous conveyor system used in the AVIS consisted of two parallel belts that were a fixed distance apart. One of the problems associated with this design was that often, the belts ran at slightly different speed and had different slipping characteristics. As a result, the part was rotated during its motion through the machine. This was unacceptable for the NCAIS as the unpredictable pose of the part could not be analytically solved as a function of the time. A new conveyor system was therefore required.

The conveyor system for the NCAIS was required to provide constant and smooth motion of a part, in adherence to the production rate. Many factors need to be considered when designing a conveyor system. These factors include belt width; drive selection; conveyor profile; and belt selection. A schematic drawing of the designed conveyor system is shown in figure 4-25.

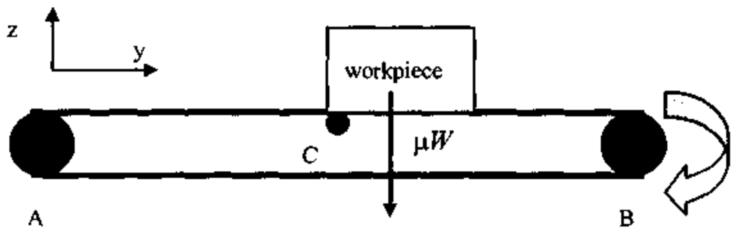


Figure 4-25: Schematic drawing on conveyor operation

The belt width was determined by the maximum width of the part to be transported. Given a predefined part width of 200mm, the width of the belt was selected to be 250mm. The constructed conveyor system is shown in Figure 4-26.

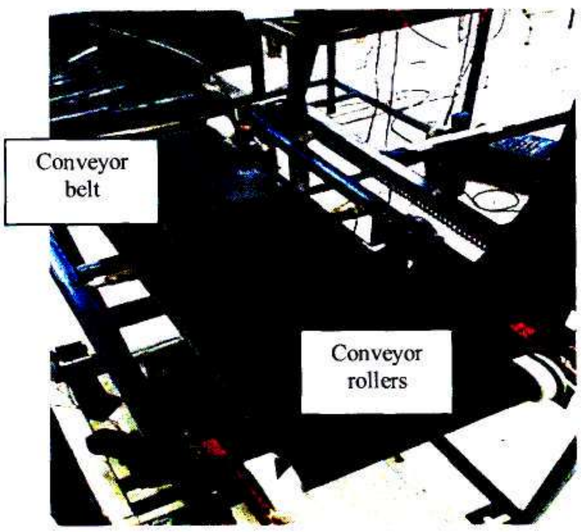


Figure 4-26: Constructed conveyor system

4.1.2.2 Part centralization system

The operation of the machine assumes that the part enters in a predictable pose. This would usually be an impractical assumption since there is always bound to be a small rotation of the part with respect to the assumed orientation, as well as some translation offset. A predictable pose can be assumed with implementation of a part centralisation system. The main function of this system is to ensure that the centre-axis of the part aligns with the centre of the machine with respect to the direction of product flow (y- axis of global frame), whilst maintaining the pre-defined orientation of the part. This concept is shown in Figure 4-27 and Figure 4-28. If the part enters the machine at some random pose, the acquired images would significantly vary from the reference image and this would lead to a less reliable inspection process.

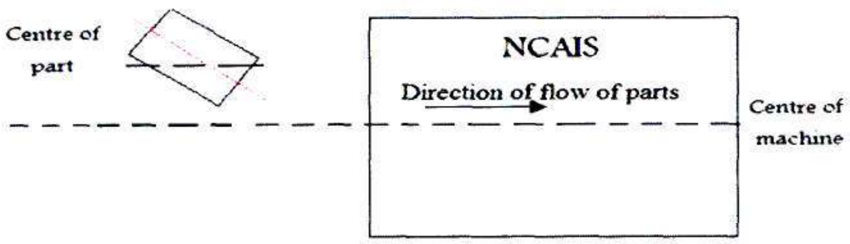


Figure 4-27. Undesired random pose of the part

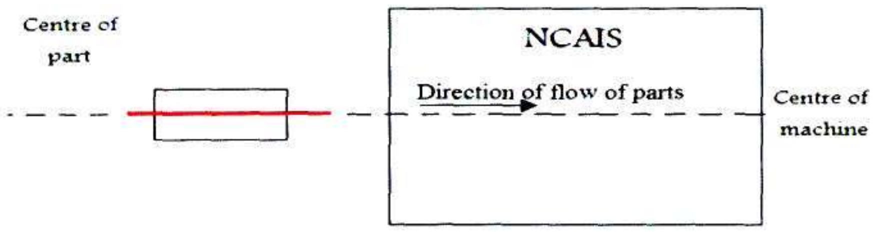


Figure 4-28: Desired predictable pose of the part

There were a number of conceptual designs that were considered for this application. The constraints for the designs were the speed of the product on the conveyor, as well as the dimensions of the conveyor-machine belt interface. The design of the system had to also account for the centralisation of parts and part families, which would vary in dimensions. This meant that the system could not be a fixed mechanism. These constraints were placed in order to ensure that the part was accurately posed in time, before entering the machine. The conveyor width was 320mm, and so the part centralisation mechanism had to operate within a range of 0mm – 320mm, taking into account the maximum part size of 200mm. The first concept considered was a belt system that could be fixed between the conveyor rollers and the machine belt. This system would have three conveyor belts. One of the belts would operate in the direction of the flow of parts. This would place the part onto the machine. The other two belts would act perpendicularly to the direction of flow so as to position the part to run along the central axis of the machine. Rail guides, which could vary their distance apart (d), would then be used to ensure alignment of the centre of the part and the machine. The converging ends of the guides would allow for initial directing of the part to the centre of the conveyor. This concept is shown in Figure 4-29. The conveyors would each have independent drive systems. The most likely selection of components for the drive systems would be belt and pulley systems, driven by separate motors. Each guide would use independent leadscrew assemblies, driven by motors with encoders for adjusting the distance between them to match the dimension of the part as it enters the machine.

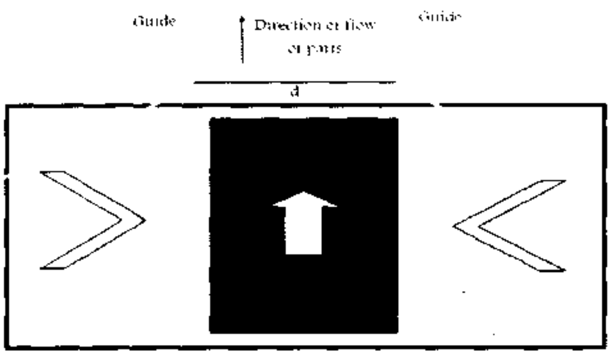


Figure 4-29: Conceptual design for Part Centralisation System

Due to the spatial CIM cell, this concept could not be implemented due to the allowable distance between the conveyor and the machine being too small. Physical design of such a system would therefore be impractical. The next concept that was considered was to implement rail guides, which mounted onto the machine itself. These guides would be driven by independent drive systems, similar to the concept discussed above. The consequence of implementing this system would be that the alignment of the part would only begin once the part has entered the machine. This would have a negative impact on the machine design since the inspection routine would only be able to commence once the part is aligned, and aligning the part after it enters the machine would delay the inspection routine. This would significantly reduce the time for the inspection routine. The concept was therefore considered unsuitable for this application.

The layout of the concept that was selected for the part centralisation system is shown below in Figure 4-30. In this design, the moveable rail guides are placed on the conveyor, between the barcode scanner and the machine. These guides would also protrude a small distance into the machine. The operation of this subsystem was such that when a part entered the section of the conveyor at which the guides are placed, the actuators moved the guides toward each other (by an equal distance). The dimensions of the part would determine the distance that the guides move. This motion would force the part to run along the centre axis of the conveyor, and hence along the centre of the machine. The curvature of the conveyor can be neglected at the conveyor-machine interface. The selected positioning of the guides would allow for the simultaneous orientation and alignment of the part prior to the part entering the machine. This allows for the inspection process to commence as soon as the part enters the machine, thereby reducing any unnecessary delays. The protrusion of the guides into the machine ensured that the part would remain in the reference pose whilst entering the machine without any significant response to disturbance at the machine-conveyor interface.

Figure 4-30: Constructed PCS

4.2 Electrical and electronic hardware

The electrical and electronic components of the system were responsible for performing the following tasks:

- Actuation of mechanical components
- Feedback for image acquisition; part identification; speed control; part tracking
- Electric power supply
- Communication
- Providing suitable lighting conditions

These tasks were all controlled by software in the host PC.

4.2.1 Actuation

4.2.1.1 Gantry and conveyor actuators

Actuation systems are elements of control systems which effect some action on a device, based on a control input [18]. The x and y-axes of the gantry system required high speed and high torque operation. The constraints considered were that the motors could not be very large as space was limited; the cost of the components must be kept to a minimum; and the actuators must be easy to control. Stepper motors were considered for selection as actuators; however the torque requirements at high speeds and the cost factors were not met by these motors. The method of actuation for the x and y axes of the gantry system, was DC motors. These motors were required to produce a torque of 6.96 Nmm and do so at a speed of 342rpm. The motors selected were a subassembly of a cordless drill [42], with its specifications given in Appendix C. The subassembly consisted of a DC motor coupled to a planetary gearbox, for torque amplification. A planetary gear system consists of small gears (referred to as planets) rotating at a fixed radius about a large central gear (referred to as the sun gear); and an outer ring (referred to as the annulus or ring) which meshes with the planet teeth [43]. The operation of this system involves the stationary annulus being used as a support, as the planet gears are rotated by the motor shaft. The planet gears then rotate the sun gear, which is seen as the output from the subassembly. The advantage of these gearing systems is that a large gear reduction can be achieved in a small volume. This system is shown below in Figure 4-31.

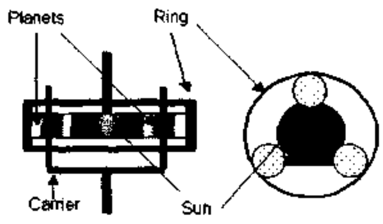


Figure 4-31: Diagram showing layout of planetary gearing system [43]

Due to the consistent low speed motion required by the conveyor, a DC motor that was commonly used as windscreen wiper motors was selected. The reason for this selection was that the conveyor was not as spatially constrained as the gantry system, and the low speed-high torque combination of this motor along with its size, resulted in very smooth and constant motion.

4.2.1.2 Pan- tilt actuators

The methods of actuation for the pan-tilt mechanism did not require translational motion due to a leadscrew. The rotation of the sensor was of utmost importance however, since any rotational errors would have lead to significant perspective errors in the acquired image. The actuator selected for the panning operation was a DC servo. DC servos are actuators that can be accurately positioned using a control signal, which are pulses that vary in length from

1ms to 2ms, in a 20ms cycle. The position of the servo shaft will remain constant (between 0 and 180 degrees), provided the input signal is constant. Any change in input will result in a change in the position of the shaft. These actuators consist of a dc motor; gears; control circuitry; and the casing for these components; shown in Figure 4-32.

Figure 4-32: Components of a servo [45]

The operation of a servo involves use of a potentiometer (in the control circuitry) to provide feedback about the rotation of the servo shaft. This value is compared to the input signal, and the motor stops when the error between the two values is approximately zero. The input signal is defined using Pulse Coded Modulation (PCM) [45], in which the length of the input pulse determines the rotation. The relationship between the input signal and the rotation can be considered linear in that a pulse length of 1.5ms will result in a position of 90 degrees, and a pulse length of 2ms will result in a rotation of 180 degrees. This is shown below in Figure 4-33.

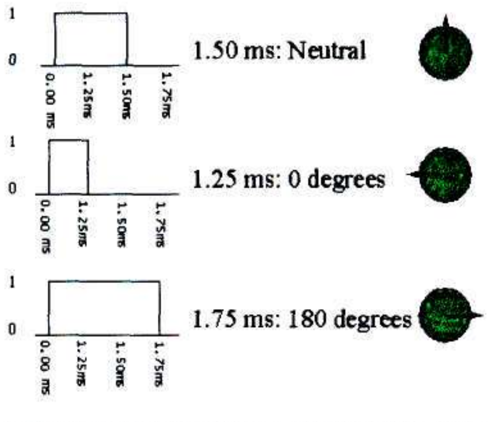


Figure 4-33: Rotation of the shaft as a function of the pulse length

The advantages of servos include high torque available in a low volume; reasonable time response; and low cost. The motor selection for the tilt operation of the camera was a straightforward DC motor. The reason for this selection was that the tilt angle of the camera was required to be at either 0 (for perpendicular image of side, front, and back faces) or 90 (for image of top face) degrees.

4.2.2 Sensor selection

Sensors were required for feedback of the following parameters in the system:

- Part identification
- Part inspection
- Motor speed

4.2.2.1 Part identification sensors

The requirements of the sensors for part identification system were:

- Rapid response for part identification process
- Must be able to provide information about the part pose
- Must be low cost

There are various types of part identification sensors; however the three most common sensors used for part identification are Radio Frequency Identification (RFID) sensors; barcode scanners; and imaging sensors (cameras). RFID involves remotely storing and retrieving data from parts, using devices RFID tags. These tags consist of an antenna, a capacitor, a microchip with a unique identification number, and sometimes a battery [46]. Communication is established between a reader and a tag, allowing for detection through non-metallic mediums at a distance. This form of identification is mainly used in security and livestock applications.

Part identification using barcode scanners involves the line-of-sight reading of a barcode by a barcode reader. A barcode consists of specific symbols, defined as a series of bars [45]. Barcodes often vary in aspect ratio (ratio of height to width), and variations in each bar denotes a unique alphanumeric character. Each barcode contains a fixed character to denote the starting and ending of the code. A barcode scanner consists of photo sensors which are able to detect the height and width of each bar, along with the spacing of that bar relative to the barcode. The most common industrial type of scanner is the Charge Coupled Device (CCD) barcode scanner. This imaging sensor converts the detected signals into electrical pulses within milliseconds [48].

Cameras can be used for part identification, but the required complexity and processing time required for identifying parts that vary only slightly in dimension and geometry is too significant a factor when compared to RFID and barcode scanning methods. Table 4-8 lists the comparison of barcode scanning systems and RFID systems [45].

Barcode scanning systems	**RFID systems**
Very low cost	RFID tags are significantly more expensive than barcodes
Barcode must be in line of sight of reader	Can read tags even with obstructions
Distance between barcode and reader is low (averaging approximately 20cm)	Read distance is high (averaging 2m)
Barcodes are large and sensitive to aspect ratios	Tags are small and can store varying amounts of data
Subject to degradation through handling	Robust against handing
Able to carry fixed limited amount of information	Tag data can be updated

Table 4-8: Comparison between Barcode scanning systems and RFID systems

The NCAIS operated on the assumption of a predictable part pose. Barcode scanning systems were able to provide this due to their line-of-sight operation. This attribute, coupled with the low cost involved with barcode scanners, led to the selection of barcode scanning sensors as the method of part identification. The barcode scanner selected was a PSC VS 1200 barcode scanner. This barcode scanner was designed as a vertical scanner with capabilities of dense scan patterns for optimal reading of damaged or poorly printed barcode labels [49]. Communication to this device via a computer is achieved using the serial communications port. The specifications for this device are shown in Appendix C.

4.2.2.2 Part inspection sensors

The most appropriate sensors for part inspection were image sensors (cameras). Image sensors are responsible for measuring and recording incident light and converting those recorded values into images. These sensors consist of various materials which generate a charge relative to the number of light photons striking them. The quality of the acquired image is essential to the operation of the vision system, and so selection of the appropriate sensor is crucial. An imaging sensor is divided into arrays of small areas called photosites, which convert measured light into a discrete value [22]. Images are considered arrays of these measured values. Many sensor characteristics need to be considered prior to selecting an imaging sensor. These characteristics include sensor resolution; aspect ratios; sensitivity; colour depth; fill rate; and frame rate.

Sensor resolution can be considered as the number of pixels available to describe an image. This description is also related to the Field Of View (FOV) of the sensor. Higher resolutions lead to higher amounts of information that require processing. The advantage of high resolution sensors is that more detailed images can be acquired. The aspect ratio of a sensor is defined as the number of pixels available to describe the height of an image, to the number of pixels available to describe the width of that image. Typical sensor aspect ratios lie between 1 and 1.5 [50]. The colour depth of an imaging sensor is the number of bits used by that sensor to distinguish between colours in an image. The sensitivity of a sensor is described by the speed at which that sensor is able to convert light into an image. Higher sensitivities are more appropriate in high speed environments. The fill rate is defined as the percentage of pixel area that is used for converting light into a voltage. The higher the fill rate, the more accurate and less susceptible a sensor is to noise. The frame rate of a sensor is the time allowed for light to enter the sensor's photosites, before the sensor shutter closes. The higher the frame rate, the more sensitive the sensor needs to be in order to produce an image. High frame rates are used to produce sharper images which are less susceptible to blurring.

The two most common types of imaging sensors available are Charge-Coupled Devices (CCDs) and Complimentary Metal Oxide Sensors (CMOS). The difference between these sensors is due to the way the charge is read out from the imaging sensor, resulting in different image formation processes. A CCD converts light into an image row by row of photosites, once the camera shutter is closed. Initially, the charges from the first row of pixels are transferred to a device called the read out register. This register is used for transferring data from the photosites to a signal amplifier, and then to an analogue to digital converter (usually a frame grabber). Once the data in the register has been read, the register is emptied and the values from the second row of photosites are stored in the register. The transfer of data is sequential and so each row is processed individually, however these values are eventually coupled to form the image data. These sensors are able to produce high quality, low noise images since they have characteristically high fill factors, and good pixel-to-pixel uniformity. Due to their complexity of design and the hardware required, implementing these sensors is generally associated with high capital costs. The operation of this type of sensor is shown in Figure 4-34 [22].

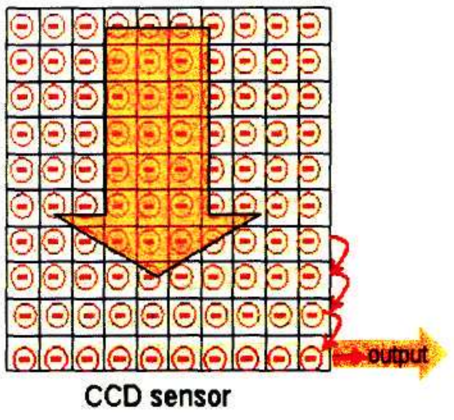


Figure 4-34: Operation of CCD imaging sensor [22]

CMOS sensors are increasing in popularity due to their low-cost and wide range of availability. These sensors have an amplifier at each photosite, and so each photosite is read individually as opposed to contiguous rows of photosites. Due to use of CMOS technology [22], additional control circuitry is generally added to these sensor chips, allowing for on-board digitization of images and light compensation circuitry. The implementation of amplifiers at each photosite decreases the fill factor of the sensor, making the sensor inappropriate under low lighting conditions. This is the reason for implementing light compensation circuitry. The heat generated at each photosite causes sensor noise which decreases the quality of the acquired image. These sensors have a low sensitivity and variations in photosite performance (due to methods of manufacture [50]), and this can result in noisy images being acquired. These sensors are generally small in size with high resolutions, allowing for use in intricate applications. The layout of a typical CMOS imaging sensor is shown in Figure 4-35[22].

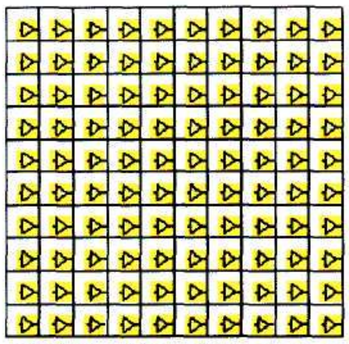


Figure 4-35: Layout of CMOS imaging sensor [22]

Table 4-9 lists the differences between CCD and CMOS sensors.

Imaging Sensor Type	Advantages	Disadvantages
CCD	High quality, low noise pictures	High power consumption
	Good pixel-to-pixel uniformity	Requires external hardware (frame grabbers)
	High sensitivity	High complexity of design
	High fill factor	High capital cost
CMOS	Low power consumption	Susceptible to noise
	Additional control circuitry can be added onto CMOS chip	Low sensitivity
	Random pixel readout capability	Low fill factor
	Low capital costs	
	Small in size	

Table 4-9: Differences between CMOS and CCD imaging sensors

The sensor required for the NCAIS was required to be:

- Low cost
- Small due to spatial constraints
- Low-weight

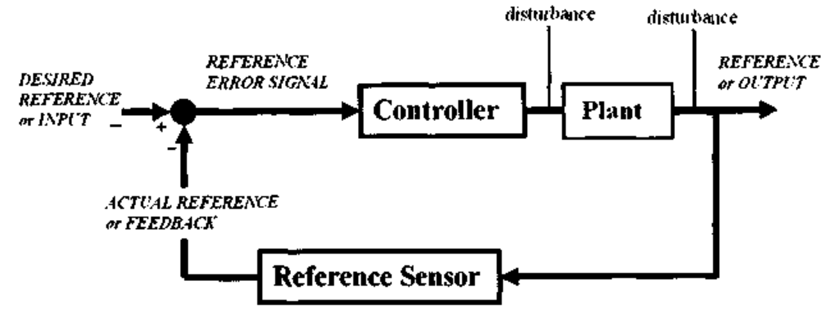


Figure 4-36: (b) Closed loop control components [51]

The required motion for each axis of the gantry structure was based on a constant spatial velocity to achieve the desired distance within the allocated time. Each axis was required to operate at different velocities for every point to point move. Operating different velocities meant that speed control was required, and this was achieved using Pulse Width Modulation (PWM) signals. The fact that the system was well defined in terms of its motion, allowed for the use of open loop control. The hardware for this motion control system consisted of motors; motor drivers; a power source, and a controller. The controller selected for generating the PWM signals was the ATMEGA32L microcontroller. This CMOS microcontroller is a high performance, low power; 8 bit controller based on enhanced AVR RISC (reduced instruction set computer) architecture [52]. The programming memory available for this chip was 32 kilobytes (KB), and serial communication to the computer was enabled via the Universally Synchronous/Asynchronous Receiver Transmitter (USART). This serial communication was required for receiving commands from the host PC and transmitting signals back to the host PC to indicate completion of the command. The microcontroller was equipped with four PWM outputs which meant that a single microcontroller was sufficient for speed control of the gantry system.

The actual speed control of the system was achieved by testing each axis of the machine under various pulse lengths and obtaining the characteristic information of pulse length versus speed. To prevent overshoot, the PWM signal was stopped before the total time required. PWM signals were generated at frequencies of 10kHz. PWM motor drivers were purchased from the electrical engineering department at the university. The circuit layout and diagrams are shown in Figure 4-37 (a) and (b). These motor drivers consisted of transistors and relays for facilitating fast PWM switching up to a frequency of 20kHz, with capacitors for smoothing the output signal to the motors.

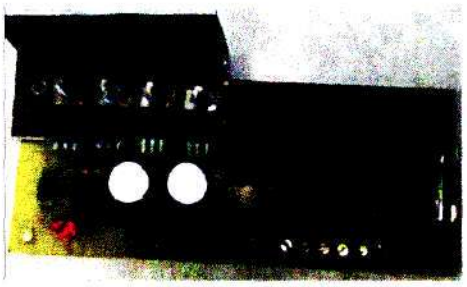


Figure 4-37(a): PWM motor driver

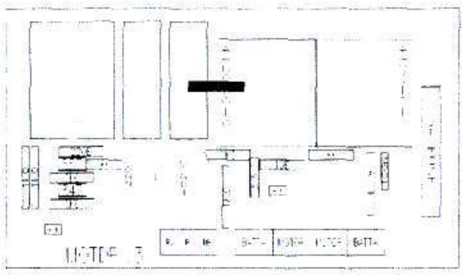


Figure 4-37 (b) PWM motor driver components

The layout of this system showing the components and signals is shown in Figure 4-38. The reference rotational speed was inputted from higher level operating software.

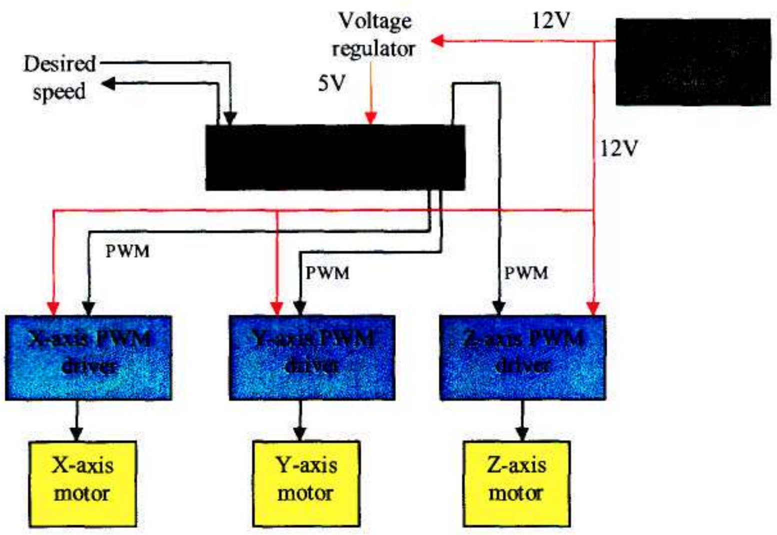


Figure 4-38: Layout of gantry system motor control circuit

The pan motor was also operated using a PWM signal to the input pins of the servo. The built-in control circuitry of the servo compared the actual position to the desired position, and executed the necessary control actions. The tilt motor was operated using a simple relay circuit and a limit switch for the two reference angles (0 and 90 degrees).

4.2.4 Lighting system

Suitable lighting conditions are essential to the operation of the image acquisition process. The lighting source determines the quality of the reflected light received by the sensor, and hence affects the quality of the acquired image. It is therefore necessary to determine an appropriate method of lighting when considering implementation of a vision system.

4.2.4.1 Nature of light

Light can be considered to have a dual nature; namely a transverse wave (electromagnetic radiation) and a particle nature (photons). The wave nature of light is used to explain the propagation of light through various mediums. The particle nature of light is used when explaining the interaction of light and matter, resulting in a change of energy (as in an imaging sensor). Lighting is crucial to image acquisition process as this allows for high contrasts between features of interest and the backgrounds in images. Reflected light (received by the imaging sensor) is a function of the colour of the incident light; the colour of the part; and the geometry of the part. White light contains a continuum of frequencies [53]. Since colour filtering can be performed in software, white light was considered most suited for general lighting conditions of the NCAIS.

4.2.4.2 Lighting methods

Methods of lighting are dependant upon the lighting source and lighting type. Common lighting sources, used in vision systems, include fluorescent lamps; Light Emitting Diodes (LEDs); and quartz halogen. A comparison of these lighting sources is shown in Figure 4-39. For information on the detailed operation of these sources, see [53].

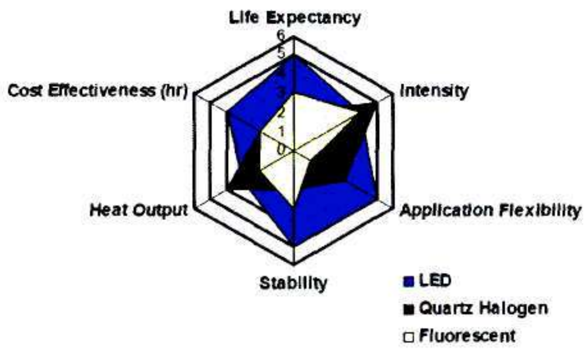


Figure 4-39: Comparison of three common machine vision lighting sources [53]

The purchased USB camera, for use as the imaging sensor of the NCAIS, had an on-board light differencing compensation circuit. This feature was used to compensate for slight variations in lighting conditions. The compensation of variations in lighting conditions led to an inaccurate image acquisition process. This inaccuracy, along with the design objectives, set the main requirements of the lighting system to be consistent; low-cost in operation; and have long life-expectancy. Fluorescent lamps operate at a frequency of 50Hz. This frequency is significantly larger than the frame rates allowed by USB cameras (12-24 frames per second), and so the flickers produced by fluorescent light sources can be detected by USB cameras. This source of lighting was therefore deemed unsuitable for use in the NCAIS, as the image acquisition process would have been inconsistent. From Figure 4-39, it was deduced that quartz halogen light sources are generally expensive when compared to the other sources, and do not have a very long life cycle. These factors, coupled with the stability characteristics, makes quartz halogen lighting sources unsuitable for the lighting system of the NCAIS. LEDs have a high life cycle; are low cost; and most importantly, offer very stable and consistent lighting. These characteristics make LEDs the most suitable lighting source for the NCAIS, and so use of only these sources of light will be further discussed.

Lighting types can be classified by the shape of the light source. The common shapes of LED lighting types are ring lights; bar lights; back lights; and spot lights. Ring-shaped lighting types are generally placed around the imaging sensor. This form of lighting type offers uniform light intensity over the entire ring, and can be used to highlight a circular area of the ROI that the imaging sensor is focusing upon. Bar lights consist of an array of LEDs that form a linear or bar shape. This setup allows for uniform lighting in the shape of a rectangular area. The lengths of these arrays range from 20cm-200cm [53]. Spotlights contain high intensity LEDs that are used to focus light onto a specific circular region. This type of lighting structure differs from ring type LEDs in that the light is not uniform around the sensor; and the light is more concentrated and focused on smaller regions than with ring lights. Backlighting systems differ from the other lighting types in that this type of lighting involves placing the light source behind the object. This type of lighting usually consists of a matrix of LEDs, and is used for inspecting transparent objects, and obtaining single-planed geometric profiles of reflective parts. Figure 4-40 (a) shows typical ring light; bar light; and spot light shapes. Figure 4-40 (b) shows the operation of a backlighting system.

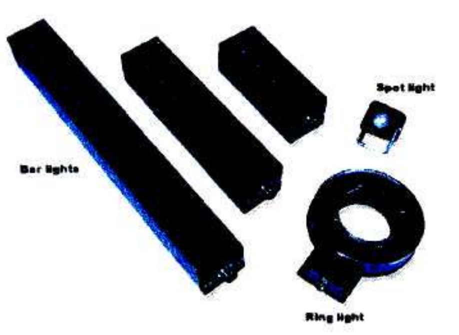


Figure 4-40 (a): Typical LED lighting arrangements [53]

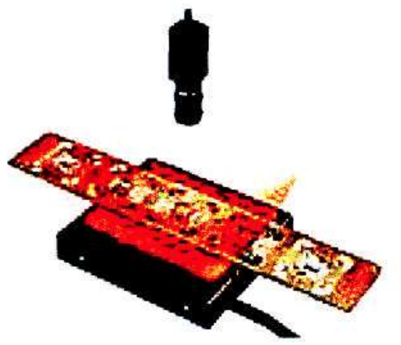


Figure 4-40(b): Backlighting system in operation [53]

4.2.4.3 Lighting position

The image of the surface of an object is greatly influenced by the position of the lighting source. The reason for this is that the light received by the imaging sensor is a function of the angle of the incident light, as well as the orientation of the camera relative to the reflected light. A significant factor when considering of angle of illumination is the glare caused by reflective surfaces. Glare can be defined as a relatively extreme light intensity level (much higher than the average intensity level), focused at a particular region in an image. This leads to inaccurate images being acquired. Reflected light (as perceived by the camera) can be divided into regular and diffuse reflections. Regular reflections consist of light waves that are reflected at the same angle as the source light; and are intensified on glossy areas (smooth, highly reflective glassy surfaces). Diffuse reflections consist of light waves that diffuse in random directions, and are intensified on non-glossy areas. In order to reduce the glare in an image, focus needs to be placed on capturing diffuse light waves. Figure4-41 (a) and (b) show the different sensor positions for reducing and increasing glare during image acquisition.

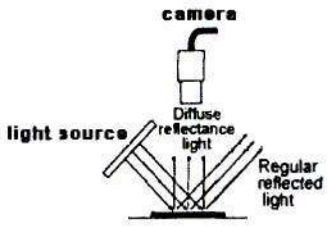


Figure 4-41(a): Optimal sensor positioning for reducing glare [53]

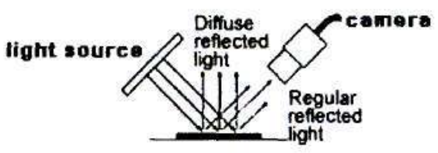


Figure 4-41(b): Sensor positioning that increases glare [53]

4.2.4.4 Lighting design

The advantage of ring lights is that the lighting conditions are consistent as perceived by the imaging sensor. The disadvantage of these lighting types is that glare is experienced when obtaining an image at a close distance at a uniform intensity. This leads to inaccuracies in the images acquired. This made the use of ring light LED structures unsuitable for the NCAIS. Backlighting was not a desirable lighting type as the conveyor and parts inspected were not transparent. Spotlights were considered, however another positioning system would have been required for positioning this light source, which would have complicated the kinematic model and SPS design. White LED arrays were selected for the lighting system design. These lights were placed on each corner the NCAIS frame, at the angle of the frames, as illustrated in Figure 4-42. The direction of the lighting was focused toward a rectangular area at the centre of the apparatus. Eight LEDs per array allowed for an intensity value of 800 lumens per array. This value was doubled when considering image acquisition of a face, since two arrays were active.

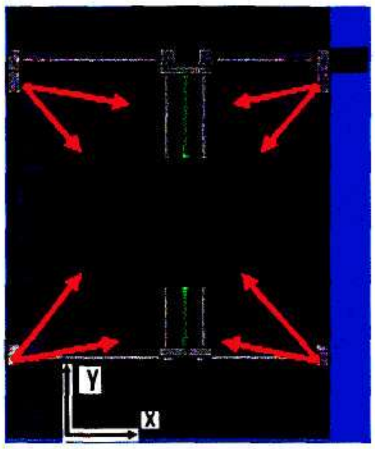


Figure 4-42: Top view of NCAIS showing direction of lighting

4.2.5 System layout

A host processor was used to perform overall process supervision, with motion controllers performing low-level control operations. A single board computer could have been used as a dedicated host processor; however a desktop PC was used due to the availability of this hardware by the university. Serial communication between the PC and the microcontroller was facilitated by using a MAX232 level shifter. The function of the MAX232 chip is to perform signal conditioning operations on the RS232 output from the PC and adapt the signal into TTL signals as used by the microcontroller. Communication between the host PC and the imaging sensor was facilitated via the USB communications port as no frame grabber was required. The output from the barcode scanner was via the RS232 serial communications protocol. Due to the PC having only one RS232 port, it was decided to purchase a USB to serial converter for signal conditioning purposes.

4.3 Software design

The system software was divided into high and low level operations. High level operations involved general system management. This level of software was used for coordinating and communicating with other levels of software for sequencing, processing, and decision making purposes. Low level software included software for motor control; image acquisition, and image processing. Different types of software were required for the operation of the NCAIS. Matlab V7 was selected for the general management of the system. OpenCV was selected for image acquisition and processing purposes. Low level motor control was performed using the ATMEGA32 microcontroller and CodeVision-AVR C compiler software. The software version of the kinematic model was written in C++ and complied with Borland C++ Builder.

4.3.1 Matlab software

The requirements of the system's high level software were to:

- Provide a system-user interface which can provide real-time user feedback during system operation
- Perform process control of system state, and system sequencing and operation
- Execute file handling operations
- Determine whether an inspected part was acceptable or not

The system-user interface was required to be simple to implement, and easy to use. The most practical user-friendly software interfaces are Graphical User Interfaces (GUIs). MatlabV7 allows for use of its Graphical User Interface Development Environment (GUIDE). This development environment makes use of object orientated programming techniques in which initializing system code is automatically generated for objects as an object is selected, which simplifies the GUI development.

The GUIs provided highly customizable options and settings for inspection routines, to the user. Using these GUIs, the desired inspection routine parameters were set by the user. These parameters included part family; part dimensions; conveyor speed; ROI; and required types of inspection. The GUI also allowed the user to repeat an inspection routine; edit an existing inspection routine; and perform an emergency stop of the system operation. The main GUI for the system is shown in Figure 4-43.

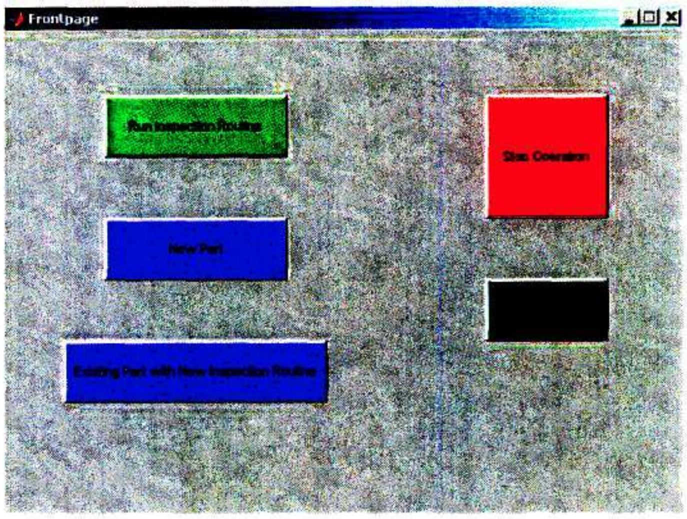


Figure 4-43: Layout of the main GUI for the system

The operation of the GUI involved calling of m-files that were written for performing specific tasks. These tasks included reading data from the barcode scanner; perform read/write file handling operations; system calibration; and decision making algorithms. The advantage of selecting MATLAB for use as the main software was that this software is able to access utility programs written in low-level languages. A utility program is defined as an independent program that produces an output on activation. Other high level languages such as Visual Basic (VB) are not able to interface with low-level languages as easily as MATLAB.

M-files were written to access and write to existing text files in the system database. A file convention was used to develop a standardized data structure for data transfer between MATLAB and the other softwares. Files that were stored included golden and background images of ROI as well as the data file which stored the routine parameters. The barcode characters of parts inspected were intended to be unique, and so these characters were used for identifying the parts inspected. Inspection could only be conducted on a part if the relevant file was located in the system database. Table 4-10 summarizes the information that was required by the system to register a new routine.

Part parameter	Data type
Bar code/part identifier	String (alpha numeric characters)
Part family	Integer (one of three part families)
Part dimensions	Integer
Conveyor speed	Integer
Number of ROI	Integer
ROI positions	Integer
Types of inspection	Integer
Inspection thresholds	Integer

Table 4-10 Summary of new part parameters

The kinematic model developed in C was used to output the sequence of actuator motions to MATLAB. The code for this model is shown in Appendix D. The sequencing of the system operation was performed by using special timer/clock functions in MATLAB. Outputs to the microcontroller software were used for sequencing the motion of each Utility file for image processing and acquisition algorithms were written using OpenCV, and these files were referenced by MATLAB whilst the inspection routine occurred. Based on the output from these utility programs, a decision was then made in MATLAB as to whether the part was acceptable or not.

4.3.2 OpenCV Software

OpenCV is an open source computer vision software libraries developed by Intel [54]. This software is based on the C++ programming language. The intended applications of this software include environments in which dynamic image processing with rapid response was required. Other software packages include image processing toolboxes. Popular software packages include Roborealm [55]; MATLAB; ERSP Vision; and Mathematica. OpenCV was most applicable for performing image processing operations during inspection, as this software was easily adaptable and operated in real time.

The requirements of this low-level software were to:

- Perform image acquisition when required by MATLAB

- Perform necessary image processing techniques as required by MATLAB
- Compare test and reference images and output result to MATLAB

Image acquisition was performed using the opencv.capture function. Due to the intended alignment of the centre of the ROI with the centre of the acquired image, some excess background information would have also been present in the image. This excess information would have increased the processing time, leading to inefficiencies in system operation. To minimize the effect of unnecessary background information in images, the acquired images were cropped. Cropping is a process wherein the outer rows and columns of an image are neglected, and the resulting image is a smaller subset of the acquired image. Cropped images contain less excess information and hence require less processing time than the original acquired images. Image enhancement and analysis operations were then performed on the acquired images to obtain data from the images for storage and comparison purposes. Once comparison between two images occurred, the output was then stored and accessed by MATLAB for decision making purposes. A flowchart of the sequence of operation for the OpenCV software, with the emphasised process for image analysis of test and reference parts, is shown in Figure 4-44.

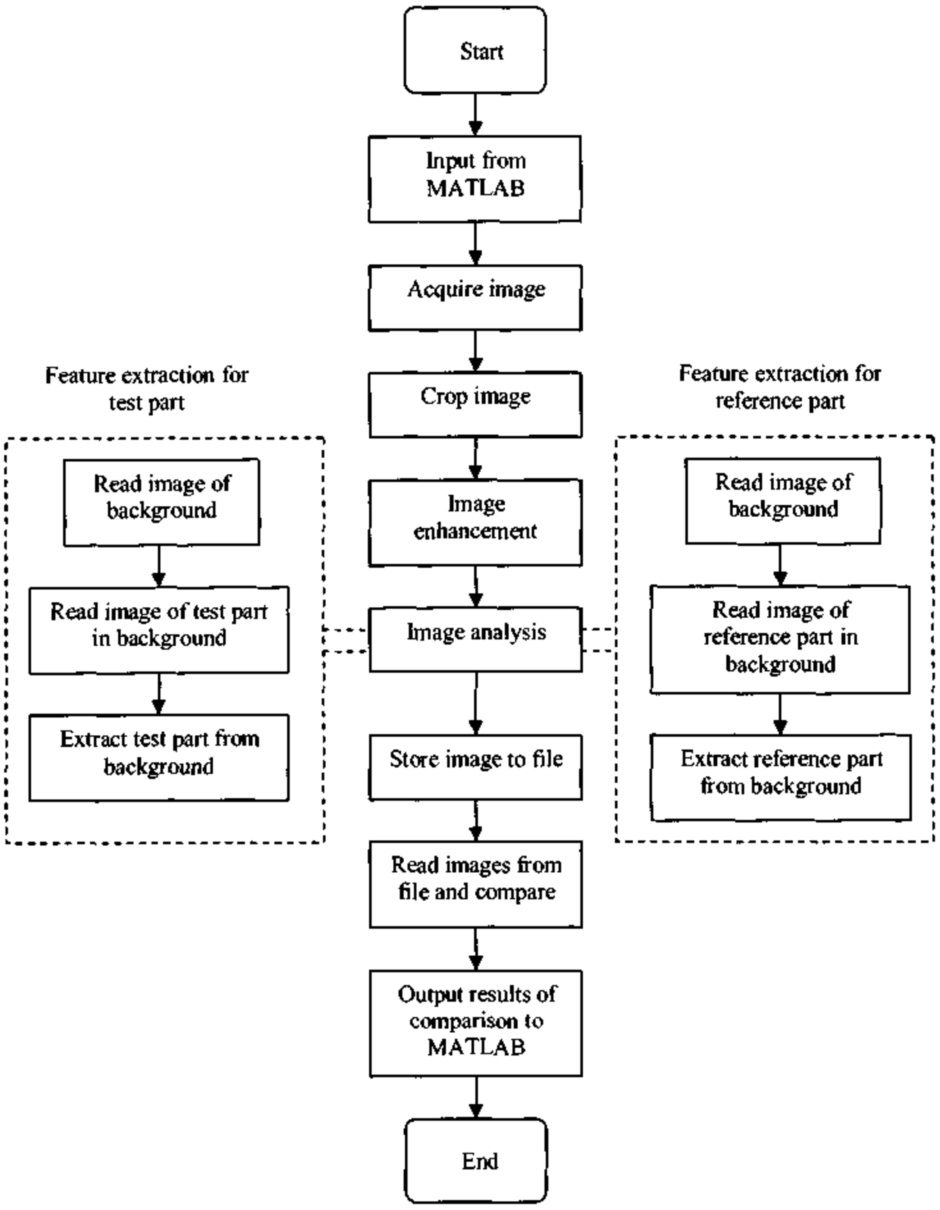

Start
Input from MATLAB
Acquire image
Crop image
Image enhancement
Image analysis
Store image to file
Read images from file and compare
Output results of comparison to MATLAB
End
Feature extraction for test part
Read image of background
Read image of test part in background
Extract test part from background
Feature extraction for reference part
Read image of background
Read image of reference part in background
Extract reference part from background

Figure 4-44: Sequence of operations required by the OpenCV software

4.3.2.1 Image data structure

Prior to detailing the actual operations of the utility programs, the method of defining the image structure used for the process needs to be established. The structure, used for mathematical representation of an image, consisted of three spatially discrete scalar functions of two independent variables. This representation was used to represent the image on each of the three primary colour channels; namely Red, Green, and Blue (RGB). The inspection process used true-colour image representation; in that each of the three channels supported 8-bit data structures (0-255 shades of a single colour). The intensity value of each channel was then represented as individual functions of the form:

$$I_r(x,y) = r$$
$$I_g(x,y) = g$$
$$I_b(x,y) = b$$

where $I_{r,g,b}$ are each 2-D representations of the intensities on the RGB channels respectively. The number of pixels in an image are used to define the size of that image. The representation of the size of an image for inspections was defined to be X in the x dimension, and Y in the y dimension. This placed the following mathematical constraints on the images of the system.

$$\left(r,g,b \in [0,255]; x \in [0, X-1]; y \in [0, Y-1]; x,y,r,g,b \in Z\right)$$

4.3.2.2 Image enhancement techniques

Image enhancement techniques are used for highlighting features of interest in an image; and to transform data that is obscured in some way in order to allow for a more reliable and efficient image analysis. Highlighting features of interest are particularly useful when considering that the inspection routines of the NCAIS were based on images of parts in a background. The isolation of the part from the background is essential for inspection purposes. This isolation was achieved by highlighting the pixels representing the part in the image. Obscured data may originate when poor image quality is achieved. The utility programs written for employing image enhancement techniques were image deconvolution; colour/grey-scale transforms; and histogram equalization.

4.3.2.2 (a) Image deconvolution

Image deconvolution is an operation commonly used to reconstruct or recover an image that has been degraded or blurred. Blurring in images is caused by relative motion during image acquisition; unsuitable lens focus; and unsuitable lighting conditions [56]. Due to relative motion between the sensor and the part, the images acquired during inspection routines were prone to blurring. This led to increased possibilities of inaccuracies in inspection routines, and therefore needed to be accommodated.

Blurred images can be modelled in the spatial domain as the convolution of the degradation system with additive noise. This model is represented by equation 4.34 [16].

$$g = H * f + n \tag{4.34}$$

where g is the blurred image; H is the distortion operator also referred to as the Point Spread Function (PSF); f is the desired image that has not been blurred; and n is the additive noise introduced during the image acquisition process [16]. The PSF describes the degree to which the vision system spreads a point of light in the spatial domain. The deconvolution of the blurred image with the PSF results in a deblurred image. For a more detailed explanation of the deconvolution process, see [56].

4.3.2.2 (b) Colour to Grey-scale conversion

In order to accurately compare two images, colour images are often converted to normalized grey-scale images. This conversion is performed using the equation 4.35 [16]. The intensity of each colour channel is considered as an independent value of a three dimensional matrix. The resulting grey-scale image then consists of one 8-bit channel.

$$G(x,y) = \sqrt{\frac{(I_r(x,y))^2 + (I_g(x,y))^2 + (I_b(x,y))^2}{3}} \tag{4.35}$$

4.3.2.2 (c) Histogram equalization

Histograms form the basis of numerous spatial domain processing techniques. Image histograms are used to characterize the distribution of the intensity values in an image. These histograms are represented as discrete functions of the form

$$h(r_k) = n_k$$

where r_k is the k^{th} discrete intensity level; and n_k is the number of pixels in the image having an intensity value of r_k. An 8-bit grey-scale image histogram involves representing the distribution of 256 levels over the image, with the index of the values ranging from 0-255. The numbers of pixels that have a certain intensity value are stored in an index counter. Having an image with high contrasts allows for easier and more reliable distinguishing between the features of interest and the background. A method of increasing the overall contrast in an image is histogram equalization. Histogram equalization is a statistical operation in which the intensity values are normalized and redistributed across the range of intensity values possible. The effect of histogram equalization on a grey-scale image is given by:

$$G_i = R\left[\frac{(G_{old} - G_{\min})}{G_{\max} - G_{\min}}\right]$$

where G_i is the grey-scale value of pixel i; R is the range of possible greyscale values (usually 256); G_{old} is the previous grey-scale value of pixel i; and $G_{\max}$ and $G_{\min}$ represent the maximum and minimum grey-scale value in the original image. Similar operations can be performed on each channel (RGB) for obtaining higher contrast in colour images. The OpenCV function cvCreateHist was used to read an image and then create the histogram of that image. The function cvNormalizeHist was used to perform histogram equalization on the histogram, and create a new normalized image which was stored in the same location as the original image.

4.3.2.3 Image analysis techniques

The software algorithms for image analysis were required to generate quantitative information about the image data produced by the image acquisition and enhancement processes. The main focus of the algorithms were to compute the statistical differences between the images of the reference part and the test part; and then quantify and output these differences to MATLAB. Prior to comparing the test and reference part attributes, these parts needed to be isolated from the background and aligned with each other to compute the differences. The isolation of the part from its background was achieved by feature extraction, and alignment of the test and reference parts was achieved by performing image registration. Image registration is the process of transformation of the test image into the reference image's scale, translational, and rotational spaces. The operations used for image registration were Log-Polar and Fourier correlation transforms.

4.3.2.3 (a) Feature extraction

The goal of this process was to subtract the background from the part. Following image acquisition and enhancement, a difference image computation was performed. For feature extraction purposes, the difference between two images may be defined as the absolute pixel-wise difference in the normalized greyscale for colour-invariant comparisons; and grey-scale of the absolute pixel-wise difference in each colour channel for colour-variant comparisons (calculated in a manner similar to actual grey-scale image G(x,y)). The mathematical representation of this differencing between two pixels with identical coordinates is represented by:

$$D_{(G,r,g,b)(1,2)}(x,y) = \left| I_{1(G,r,g,b)}(x,y) - I_{2(G,r,g,b)}(x,y) \right|$$

Based on this definition, the absolute pixel-wise difference metric for the entire image was defined as an indicator of the degree of direct correlation between two images. This image was given by equation 4.36 [16], with X and Y being the number of pixels in the x and y dimensions of the image.

$$\Delta_{i,j} = (X * Y)^{-1} \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} D_{(G,r,g,b)(i,j)}(x,y) \tag{4.36}$$

This difference yielded the isolation of the part from the background; however this process was prone to errors in that the background in the two images may have been misaligned. Log-polar and Fourier transforms were then implemented for improving system accuracy.

4.3.2.3 (b) Log-Polar transform

Image registration must be implemented when accounting for any inconsistencies in the image acquisition process. These inconsistencies may be present due to accumulated errors in the SPS, and initialization delays in system software. In the case of comparing two images that involve moving systems, rotational variances must be accounted for. The log-polar transform is a mathematical one-to-one mapping in 2D Cartesian space, used for eliminating rotational variances between two images of the same part. This transform is represented by:

$$(r,a) \leftrightarrow (x,y)$$

$$r = \log\left[\sqrt{(x-x_c)^2 + (y-y_c)^2}\right]$$

$$a = \tan^{-1}\left[\frac{(y-y_c)}{(x-x_c)}\right]$$

where x_c and y_c are the coordinates of the centre of the transformation. The centre of the transformation was set to be the centre of the image since the image resolution is best preserved at this point (due to convex lenses in the camera). The function of this transform is to convert rotation about the centre point as a translation along the a direction, and scaling about the centre point as a translation along the r direction. Using this transform, any scaling and rotational variances were converted into 2D translational errors. The cvLogPolar function was used for reading and transforming images; and then storing the transformed images. The code for this utility is shown in Appendix D.

4.3.2.3 (c) Fourier correlation

The second requirement of the registration process was to account for any translational inconsistencies between the reference and test images. Spectral analysis was the implemented method of eliminating errors in the log-polar transformed images. Images were transformed into the frequency domain using Fourier transforms. The Discrete Fourier Transform (DFT) of an image f(x,y) of size XY is defined below [16].

$$F(u,v) = \frac{1}{XY}\sum_{x=0}^{X-1}\sum_{y=0}^{Y-1} f(x,y)e^{-j2\pi(ux/X+vy/Y)}$$

Similarly, given $F(u,v)$ the inverse DFT of an image can be used to obtain f(x,y). The inverse DFT of an image is then given by:

$$f(x,y) = \sum_{u=0}^{X-1}\sum_{v=0}^{Y-1} F(u,v)e^{j2\pi(ux/X+vy/Y)}$$

These transforms can be calculated using one of many Fast Fourier Transform (FFT) algorithms available, which reduce the computational complexity of the transform. Although the Fourier transform operates on complex functions, the images used consisted of only real numbers, and hence by symmetry properties of the Fourier transform, the resulting inverses of transformed images are guaranteed to return purely real images.

The translational shift between two images was required to be calculated very quickly. Given two similar images, the amount of translational shift between them may be determined using the Fourier correlation theorem. This theorem states that the Fourier transform of a correlation integral is equal to the product of the complex conjugate of the Fourier transform of the first function (F*) and the Fourier transform of the second function (G) [16]. This is represented by the following equation:

$$C(f \bullet g) = F^{*}.G$$

The only difference between this and the convolution theorem [16] is in the presence of a complex conjugate, which reverses the phase and corresponds to the inversion of the argument u-x. Figure 4-45 illustrates that if the two functions f and g contain similar features, but at a different position, the correlation function will have a large value at a vector corresponding to the shift in the position of the feature.

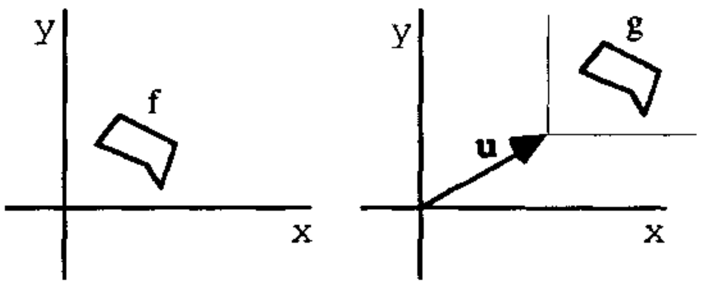


Figure 4-45: Resulting vector due to a Fourier correlation operation [16]

Using the obtained information from this operation, the images resulting from the log-polar transforms were aligned to eliminate any translational differences. The code for the utility program that performed this operation can be found in Appendix D.

4.3.2.3 (d) Image comparison

Following the feature extraction, log-polar transformation, and Fourier correlation operations; the inverse log-polar transform was required for returning the image back to its original form (Cartesian coordinates). Equation 4.36 was then used to perform a pixel-wise differencing operation to quantify the statistical difference between the reference and the test parts. Following the differencing algorithm, the resulting difference image was converted into a binary image to ensure that the differences were significant enough. This was constrained by a minimum preset threshold value of 5%. Following the binary transformation, it was possible that the areas showing differences in the images (difference regions) were due to noise in the system. To account for this problem, only differences higher than a certain number of pixels (5% or more of total number of pixels) were considered when computing the differences between the images. The utility program for comparing two images is shown in Appendix D.

4.3.3 Microcontroller software

The software package used for programming the microcontroller was CodeVision AVR, which offers the advantage of CodeWizard software that automatically generates the initializing code for the microcontroller as requested. This software is aimed solely for use on AVR microcontrollers programmed by the STK500 kit. The user simply enters the specific chip being used, the clock setting, the input/output port configuration, and the timer configurations; and the corresponding code is automatically generated and saved in a source file. The code implemented onto the microcontroller for motion control is shown in Appendix D.

4.3.4 Simulation of Kinematic model

A simulation of the kinematic model was written in C++. This model was designed to convert the ROI passed by MATLAB into moving coordinates. A sequence of the possible paths was then generated. Based on the convey speed input from MATLAB, the model calculated the sensor and part trajectories and checked for any possible collisions. Any collision-involved paths were rejected and the shortest possible path was selected as the optimal path. This sequence of motions was then returned to MATLAB which then performed the sequencing of this operation whilst communicating with the microcontroller software. An animation of the optimal path was also performed. The code for this simulation is shown in Appendix D.

4.4 Summary of chapter 4

This chapter serves to highlight the core elements of Mechatronic engineering for various subsystems design. The mechanical components were designed base on information provided in the mechanical design section. The electrical components were selected and designed considering mechanical components. Software was developed based on the requirements of the other two core elements. System analysis was performed both analytically and using simulations.

5. ASSEMBLY

The assembly phase of the project involved the physical integration of the subsystems to constitute the entire system. The assembly, along with the signal analysis of each subsystem, was performed prior to system integration.

5.1 Part Identification System

The PIS comprised of the barcode scanner linked up to the computer. No mechanical design was involved with this subsystem, apart from mounting the scanner onto the conveyor using the supplied bracket. The sequence of signal flow from the barcode scanner to the PC is shown in Figure 5-1.

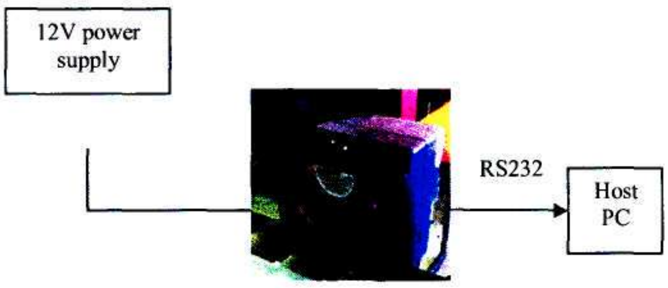


Figure 5-1: Signal flow in assembled PIS

5.2 Sensor Positioning System

The finalised constructed SPS consisted of the mechanical members used for positioning the sensor, the circuitry required for electronic control; the actuators (motors); and the software module for operating the different aspects of the subsystem. The component layout, including the sequence of the control signal of the SPS is shown in Figure 5-2.

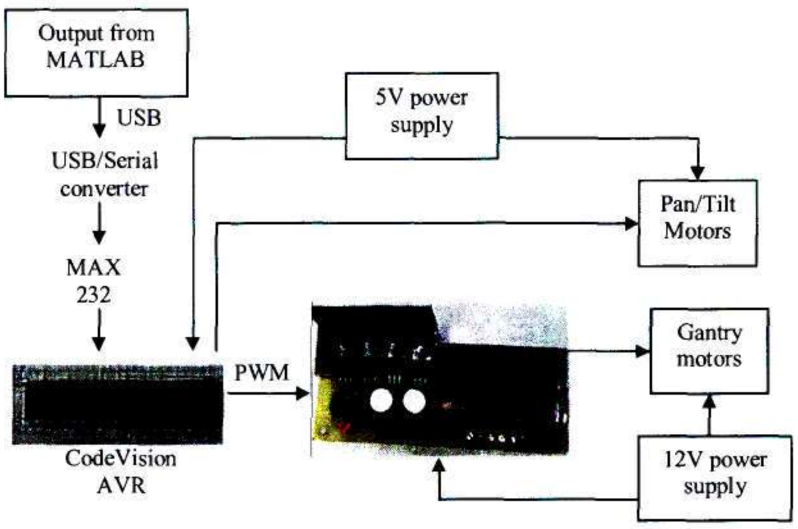


Figure 5-2: Component layout showing signal flow of assembled SPS

5.3 Materials Handling System

The assembled MHS included an operating conveyor system and part centralisation system. The layout for this system was almost identical to the layout shown in Figure 5-2, with the exception of conveyor and PCS motors as opposed to gantry and pan-tilt motors.

5.4 Vision System

The layout of the assembled VS consisted of OpenCV and MATLAB software; and the imaging sensor. The various utility programs were called by the MATLAB program when specific operations were required. The layout of this system is shown in Figure 5-3.

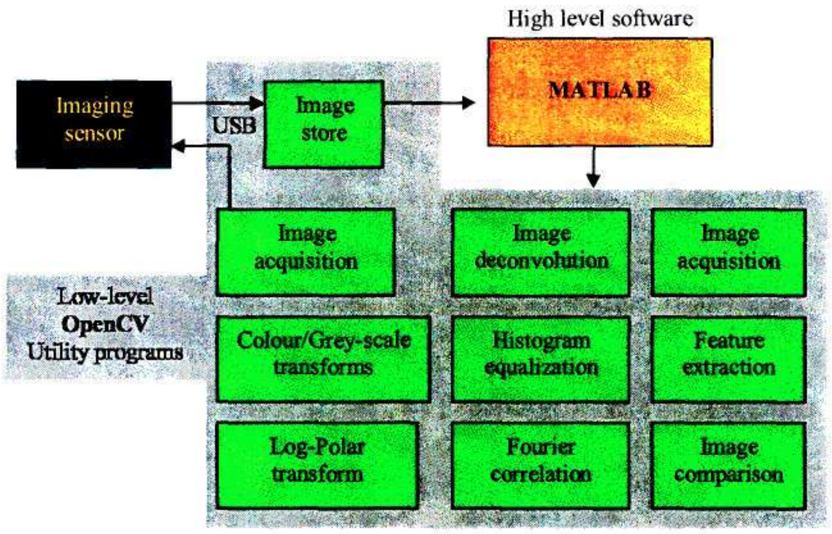


Figure 5-3: Layout of VS

5.5 System integration

The system integration can be explained by Figure 5-4. This diagram shows the interdependencies between the core elements of Mechatronic engineering. An example of this interdependency was the operation of the motors and encoders. The selection of the speed sensor was dependent upon the speed of the motor. The motor speed was dependent on the required speed of operation. The speed of operation was dependent upon the speed of the part and the mass of the mechanical components. The actuation of the motors was dependent upon the software algorithms and electronic hardware. The output from the software was dependent upon the feedback from the sensors.

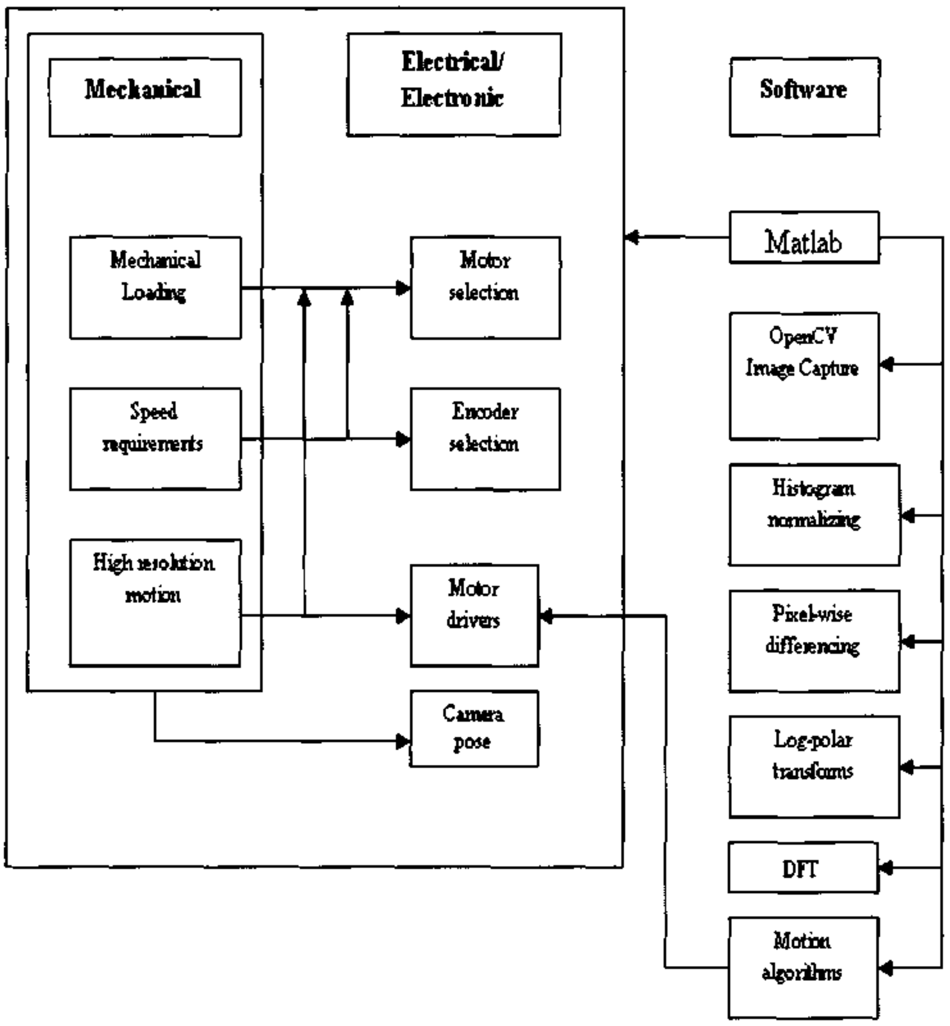


Figure 5-4: Illustration of the interdependencies of the core elements on each other

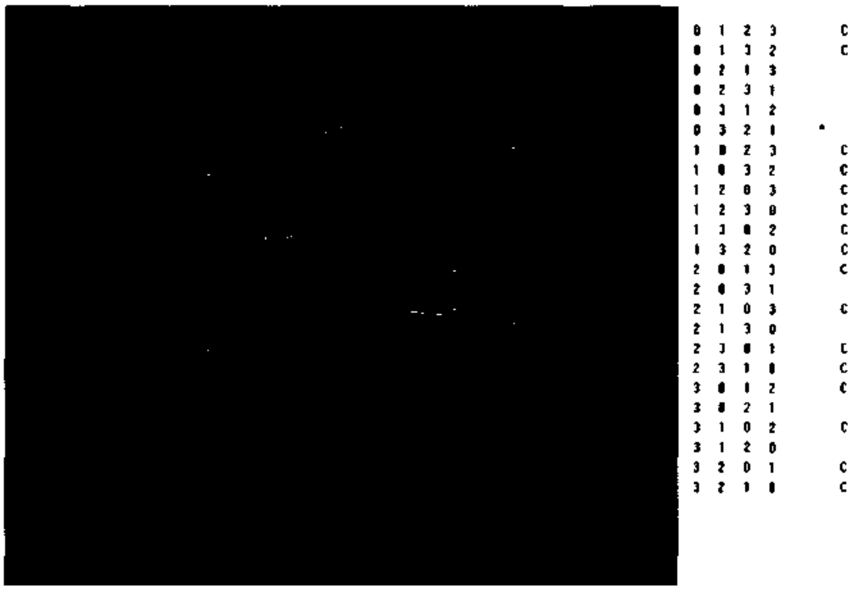
5.6 Summary of chapter 5

This chapter highlights the assembly phase of the design. All subsystems were assembled to function as a single unit. The interdependencies of each element of the Mechatronic engineering design were highlighted and discussed.

6. RESULTS AND DISCUSSION

6.1 Kinematics simulation results

The kinematic model was used to simulate the overall trajectory of the sensor relative to the moving part. This simulation shows the order of the paths that could have been followed. These paths were based on the positions of the ROI on the part; the velocity of the part; and the dimensions of the part. For example, the path (0 1 2 3) means that the trajectory of the sensor would be starting from ROI 0, going through to ROI 3.The ROI that were selected for the simulation were strategically placed at the furthest location from the camera, on each face. The camera field of view was able to accommodate adjacent ROI centres on the same image. Consequently, no more than two ROI were place on a single face of a part smaller than 200mmx200mmx200mm. The outputs form the simulation were the number of free paths and the shortest possible path. The green lines represent the part boundary space as a function of time. The purple line represents the trajectory of the sensor as a function of time. If a collision between the sensor and the part ever occurred whilst following a specific path, the path was labelled as colliding (marked with an adjacent c) and hence rejected. The paths that did not collide were then analysed, and the path that offered the shortest total distance was selected and highlighted with an asterisk as shown in Figure 6-1.



```
0 1 2 3       c
0 1 3 2       c
0 2 1 3
0 2 3 1
0 3 1 2
0 3 2 1     *
1 0 2 3       c
1 0 3 2       c
1 2 0 3       c
1 2 3 0       c
1 3 0 2       c
1 3 2 0       c
2 0 1 3       c
2 0 3 1
2 1 0 3       c
2 1 3 0
2 3 0 1       c
2 3 1 0       c
3 0 1 2       c
3 0 2 1
3 1 0 2       c
3 1 2 0
3 2 0 1       c
3 2 1 0       c
```

Figure 6-1: Simulation of kinematic model showing motion of part and sensor

The following results, shown in Table 6-1 were obtained for various part sizes, and the minimum possible inspection times indicate that higher conveyor speeds could have been facilitated for these inspection routine configurations. All dimensions were in mm. Four identical ROI were selected for each part.

Part dimensions (LxBxH)	Number of collision free paths	Number of colliding paths	Minimum inspection times
100x100x100	8	16	26s
100x150x100	4	20	27s
200x100x100	16	8	20s
200x100x200	8	16	22s

Table 6-1: Kinematic simulation results for randomly sized parts

6.2 Image processing algorithm results

The image processing algorithms were used to compare reference images against images of both acceptable and unacceptable parts. Images used included still images (for testing just the algorithm), and images which involved relative motion between the sensor and the part (for testing the algorithms whilst the machine was operational). The acquired images differed in that the features of interest in the images were at different poses, and the lighting conditions varied slightly. The OpenCV utility programs written for reducing translational and rotational non-conformities of the test images were then employed by Matlab.

The results of these image processing operations are shown in Figure 6-2(a) and (b). The component involved was a mounting bracket designed to fit onto a research platform in the CIM cell. A reference part was produced and an image of the top view of this component was acquired, as it passed along the conveyor belt at a speed of 0.01m/s. This part is shown in Figure 6-2(a). The same procedure was followed when a test part was manufactured, shown in Figure 6-2(b). The images shown in these figures are the corresponding images obtained after the feature extraction process. The ROI in this sample were the two holes, shown in green and red squares, used to house support bars in an assembly.

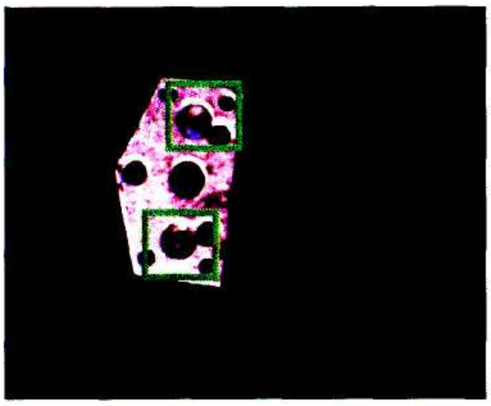


Figure 6-2(a): Reference image

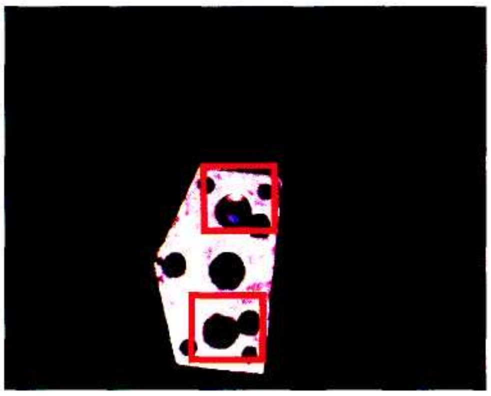


Figure 6-2(b): Test Image

Once the images were acquired and the features extracted, the rotational and translational non-conformities were eliminated using the log-polar and Fourier-correlation utility algorithms. The difference between images resulting from these operations is shown in Figure 6-3(a). The difference between the images after the filtering algorithm is shown in Figure 6-3(b). The figures show that the difference between the test part and the reference part is negligible, and the part was therefore classified acceptable.

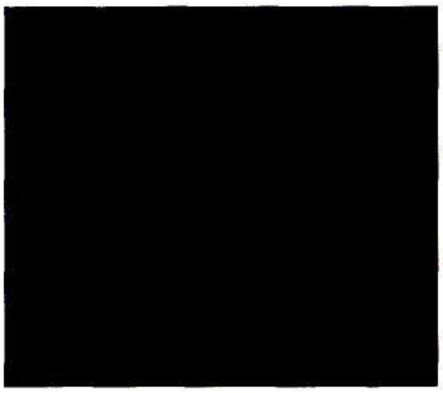


Figure 6-3(a): Difference of aligned images

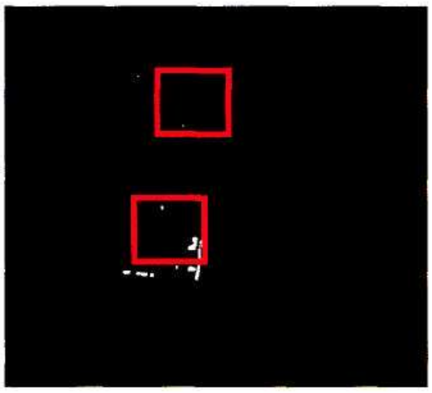


Figure 6-3(b): Image differences after cleaning

The same type of component was to be made, however the part was inspected prior to the drilling process on the ROI, and prior to an initial finishing process which ensured dimensional accuracy. This premature inspection routine was performed to verify that the algorithms would detect the non-conformities. The same reference image as shown in Figure 6-2(a) was used. The image acquired of the non-conforming part, after feature extraction, is shown in Figure 6-4(a). The operations used to inspect the previous part were again used to check the non-conforming part. The difference images (Figures 6-4(b) and (c)) indicated that the scale of the two parts was slightly off. This could have been due to different camera positions. The major differences highlighted in Figure 6-4(c) were at the location of the holes for the support bars. Based on the difference at this ROI, the part was classified unacceptable.

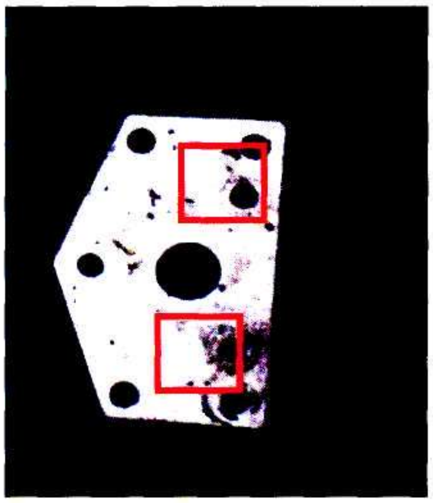


Figure 6-4(a): Non-conforming part

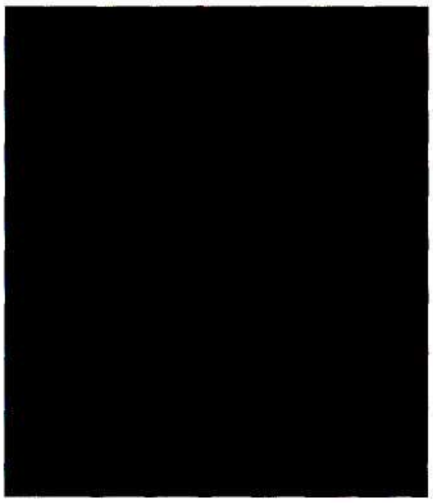


Figure 6-4(b): Difference images

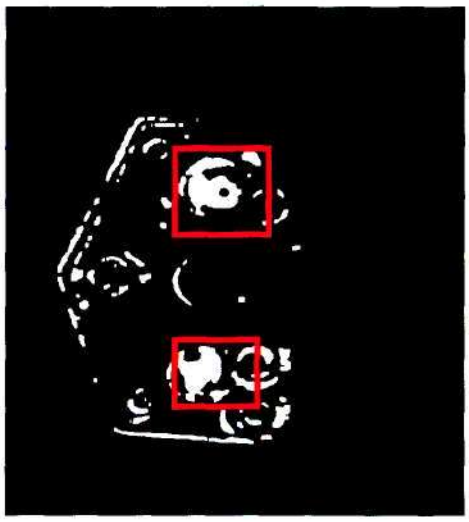


Figure 6-4(c): Cleaned difference image of non-conforming part

The time taken, from image acquisition till the image processing result completion, was observed as being less than a second for each ROI. This high speed operation justified the used of inspection based on machine vision.

6.3 System performance

The general performance of the system included analyzing the motion of the system coupled with the response times of the controlling and image processing software. During operation of the apparatus, it was observed that the technical effort required to ensure control; synchronization; and tracking, could have been significantly reduced by a more algorithmic and computational approach. Machine learning knowledge based systems would have been able to correlate 2D representations of 3D objects. This feature would have allowed for an increase in the degree of independence of the captured image from the physical pose of the camera, which would have then reduced the dexterity required by the sensor. This would have simplified the mechanical design of the system. The motion performance of the apparatus was tested with the system being tested initially in terms of each independent axis, and then as a whole for accuracy and repeatability.

6.3.1 X-axis

The x-axis drive system was operated with the pulse time equalling the duty cycle (full PWM signal) in order to quantify the maximum velocity of the axis. The maximum velocity was used since this velocity would result in the maximum overshoot. In this testing process, the drive system was operated alone at full speed, for a set of discrete times, ranging from 0.1s till 17.5s. The distance moved as a result of this operation was then measured. The results of the distance versus the operating times are shown in Figure 6-5. These results are shown in Appendix B. The equation of the graph was found to be of the form

$$y = 22.1x - 1.64$$

The straight line graph below confirmed that the motion of the x-axis involved a linear relationship between the distance travelled and the time that the motor was operated at full speed, which then verified a constant average maximum velocity of 22.1mm/s. The reason that the graph did not pass through the point (0,0) was due to friction and inertial effects.

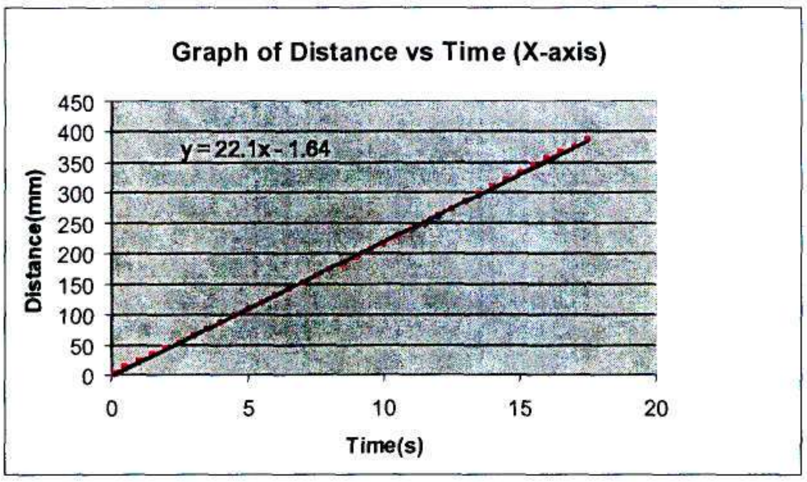


Figure 6-5: Graph used to obtain maximum x-axis velocity

Using this information, an estimated linear relationship between the desired speed and pulse length was established. This relationship was then used for acquiring an open loop model for the x-axis. A factor of concern was the performance of the x-axis when the y-axis slider was at different positions along the y-axis. The reason for this was that different slider positions would have varied the load distribution on the driving and support ends for motion along the x-axis. When the load was at y=0, the greatest deflection would have been experienced by the x-axis components. With the load at y=l, the smallest deflection would have been experienced by the x-axis. It was therefore decided to test the performance of the x-axis at extreme y-axis slider positions to see if any significant variations in performance occurred. The following results were obtained.

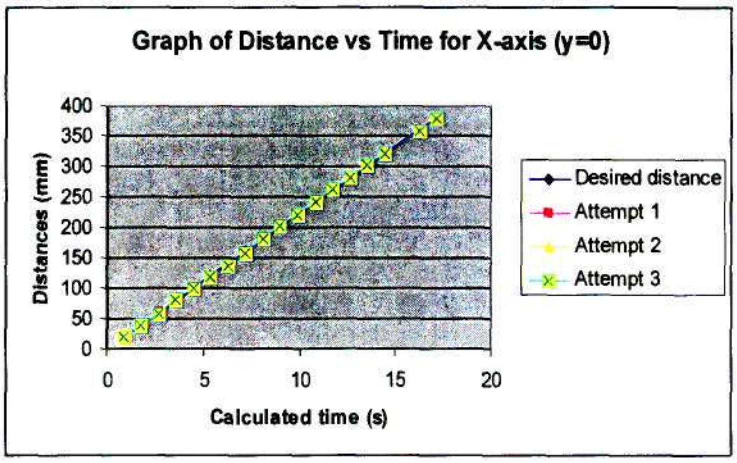


Figure 6-6(a): X-axis performance (y=0)

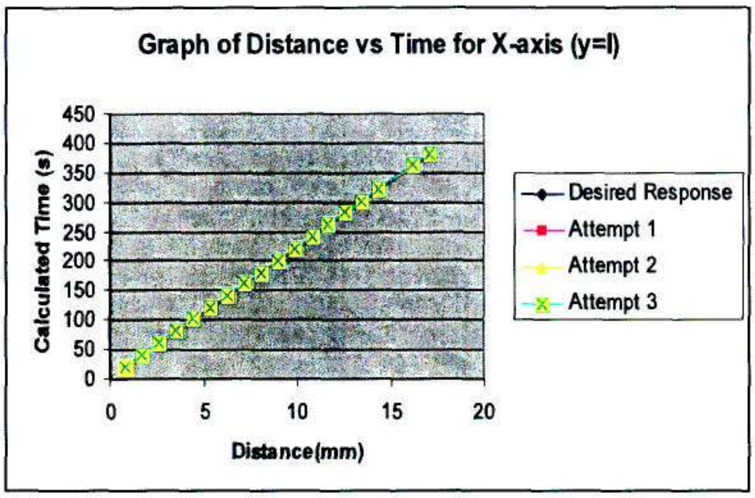


Figure 6.6(b): X-axis performance (y=l)

Figure 6-6(a) and (b) shows performance of the x-axis at y=0 and y=L. As shown in the graph, the slider position along the y-axis did not significantly affect the performance of the x-axis drive system. The errors for the x-axis positioning are shown in figure 6-7(a) and 6-7(b).

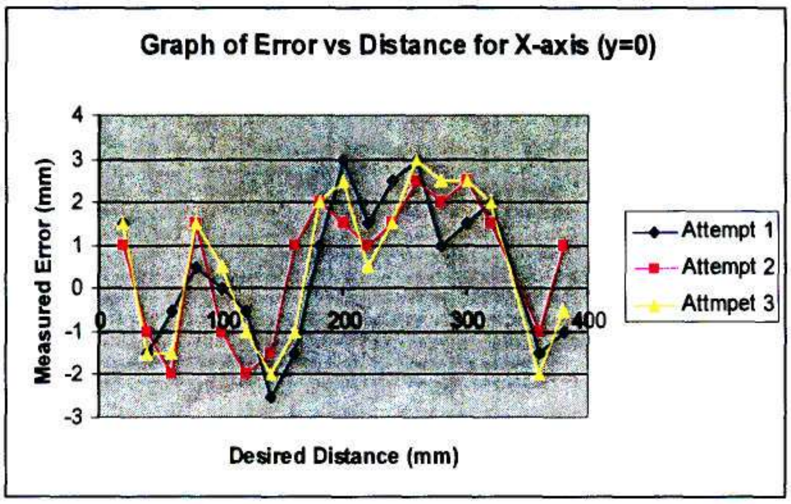


Figure 6-7(a): Measured error between actual and desired distances along the length of the structure (for y=0)

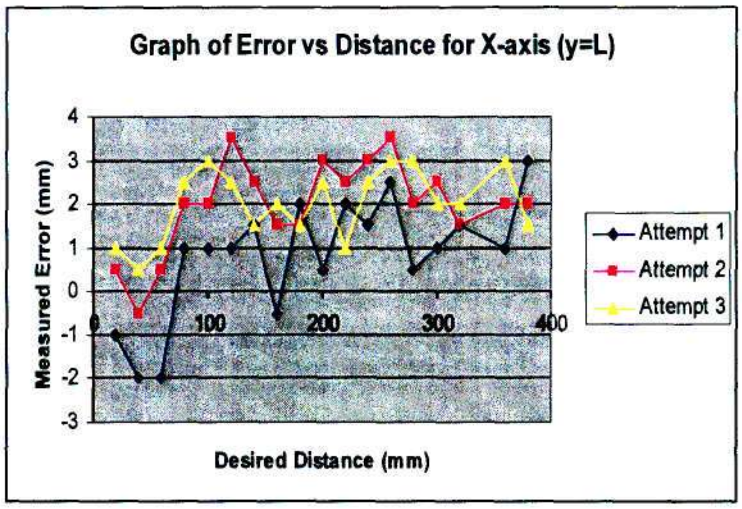


Figure 6-7(b): Measured error between actual and desired distances along the length of the structure (for y=L)

From both sets of x-axis performance results (y=0 and y=L), the maximum error obtained was 3.5mm, which was within the specifications of a maximum error of 8mm. These values indicate that the accuracy and repeatability of the x-axis, with various y-axis slider positions, was acceptable. The deflection of the x-axis under the worst loading condition (x=L/2; y=0) was measured to be 0.12mm. This value was below the minimum calculated value of 0.26mm. The difference between calculated and measured values was due to the fact that the static analysis accounted for each member sustaining the entire load alone; whereas in the measured value the load was shared amongst all the members accordingly.

6.3.2 Y-axis

The results for the motion of the y-axis are shown, Appendix B. The system was tested to quantify accuracy and repeatability, as well as estimate the losses due to friction and inertia, during operation of this axis. A graph of distance versus time, with the axis being driven at full pulse length, was plotted from the obtained results. The results confirmed that the distance travelled by the camera in the y-axis could be modelled as a linear function of time, and hence verified that the motor velocity could be modelled as being constant. The average velocity was estimated to be 22.92mm/s. This speed was acceptable as it was higher than the maximum part speed of 0.02m.s. A graph representing the motion characteristics of the y-axis is shown in Figure 6-8. The equation of the graph was found to be

$$y = 22.92x - 0.3208$$

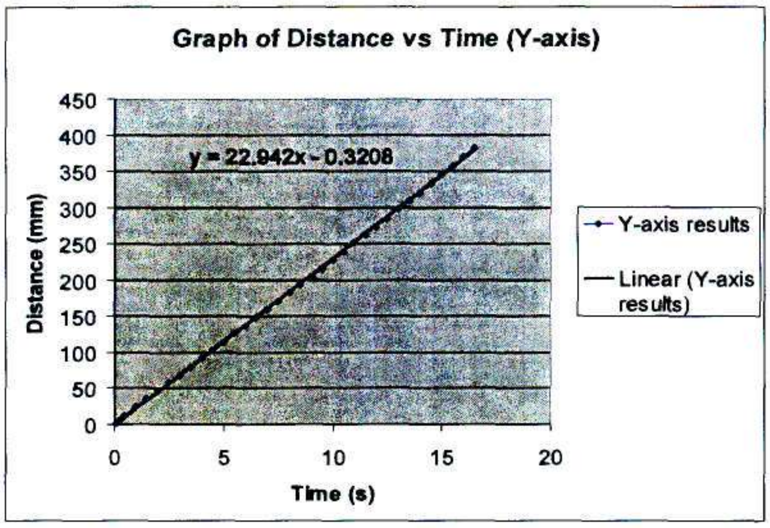


Figure 6-8: Y-axis motion characteristics

The motion was then tested for accuracy and repeatability by inputting the required move distance, and then measuring the actual travelled distances. Three attempts were made to obtain results in order to verify repeatability. The graph shown in Figure 6-9 relates the errors in the actual distances to the desired distances. It can be seen that the maximum errors in the motion were approximately 3.5mm which is less than the acceptable 8mm as specified. The largest errors occurred at the middle and toward the end of the y-axis structure.

The errors at the middle of the structure could have been as a result of the accumulation of inaccuracies in the estimated equation for the system. The undershoot toward the end of this structure, which influenced estimation of the open loop model, was most likely due to the structure having some form of deformation at these distances. Deformations could have been inherent during the manufacture of these components. These deformations would have increased the friction in the motion, leading to the travelled distance being less than the calculated distance.

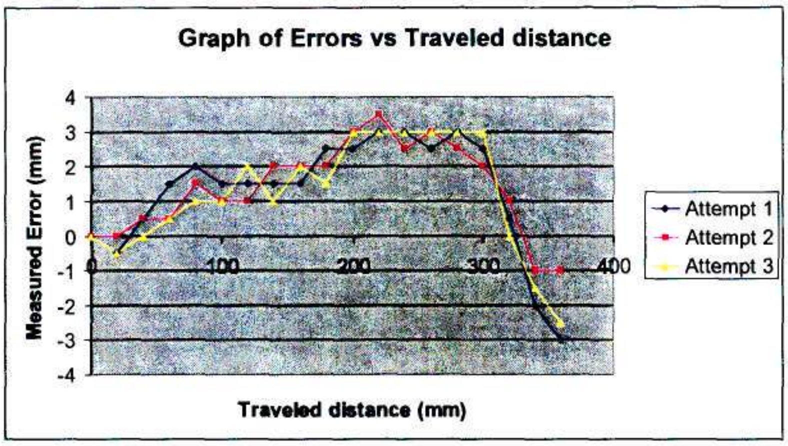


Figure 6-9: Graph showing measured errors of y-axis motion

The errors involved in positioning the y-axis were larger than the errors involved with the x-axis performance. This would have been largely due to the fact that the x-axis was rigidly fixed onto the frame of the NCAIS; whereas the y-axis structure was fixed onto a moving x-axis slider and a roller on the opposite side. Both these errors in positioning were acceptable despite a difference in the general performance of these axes. In order to obtain the accuracy of the axis, the percentage of the errors, for the different attempts, were plotted against the travelled distances. These results are shown in Figure 6-10.

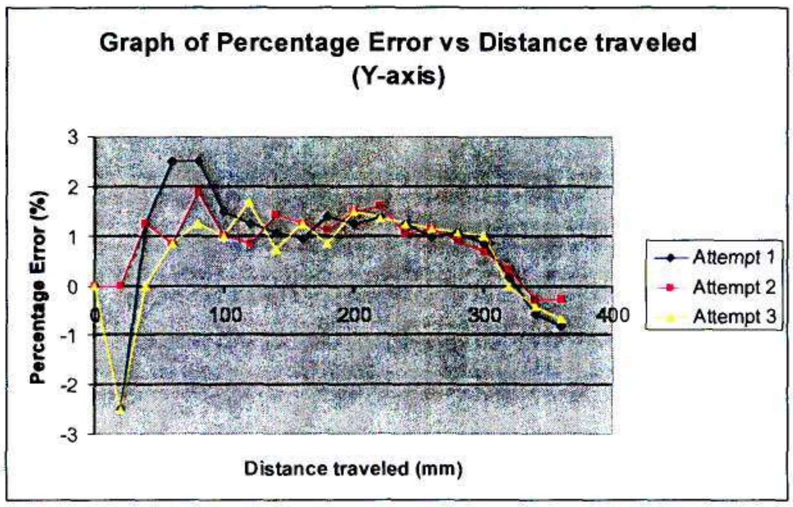


Figure 6-10: Accuracy and Repeatability of Y-axis

The axis errors showed an accuracy ranging from 97%-100%, which was essential for the motion of the y-axis. This high accuracy was crucial, as the greatest independent distance would have to be travelled by this axis, and so error accumulation had to be kept to a minimum. It can also be seen from the graph that the errors were within 2% of each other which indicated a high repeatability characteristic of the axis. The deflection of the y-axis under the heaviest loading condition (y=L/2) was measured to be 0.08mm. This value was below the minimum calculated value of 0.18 mm. The difference between calculated and measured values was the same as explained for the x-axis deflection.

6.3.3 Z-axis

The z-axis was tested similar to the x and y axes. The operation of the z-axis was limited to an operating distance of less than 100mm due to physical constraints. The motion of the z-axis did not require small movements due to the tilt function of the SPS. Consequently, only 4 test distances were used as only large movements were required. A graph showing the distance versus time of the results is shown in Figure 6-11. The equation of the graph was found to be

$$y = 5x - 5$$

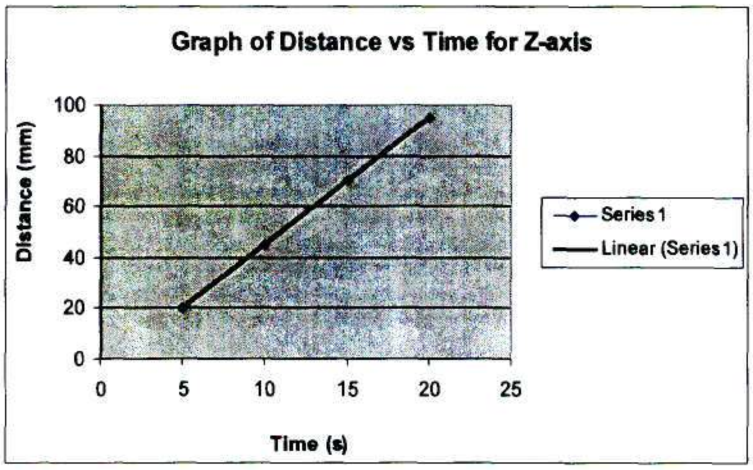


Figure 6-11: Motion performance of the Z-axis

Figure 6-12 shows that the performance of the z-axis was within the specifications of general accuracy and repeatability. This axis was not as significant in functioning as the other two axes. The reason for this is that the pan-tilt mechanism would allow access to ROI from the allowable heights of the system. The reason for implementing a z-axis was then to illustrate the principle of collision avoidance. This could have been achieved by operating the z-axis until its maximum height was achieved, and so no precise movement was necessary in the z-axis as was required in the other two axes. Implementation of the NCAIS in actual practical industrial applications would however require the increased functionality of this axis.

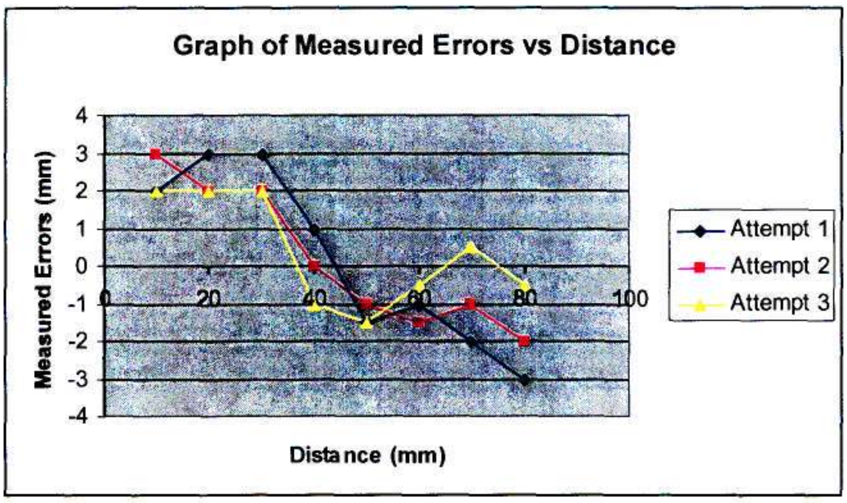


Figure 6-12: Graph showing errors of z-axis

6.3.4 Overall performance

The system was tested for overall three-dimensional spatial accuracy. Four random test points were selected as target points. The method of testing involved starting the camera at its datum point of coordinates (0, 0, 0), and then using the models to move the sensor to the desired location. The actual sensor position was then measured relative to the datum point. This test was conducted ten times for each of the four random target points which had coordinates (20, 20, 10); (35, 80, 5); (50, 50, 15); and (100, 70,10) respectively. The results of these tests are illustrated in the graph shown in Figure 6-13. This graph shows that the maximum spatial displacement (of 3mm) was acceptable for the accuracy of the sensor positioning.

The measured deflection of the camera was found to be within a 1mm radius and the change in tilt was estimated to be less than 1 degree. These measurements were made using a micrometer which was fastened at a fixed height. Due to the small magnitude in these measured values, the change in sensor pose due to deflections of the positioning system was therefore considered negligible.

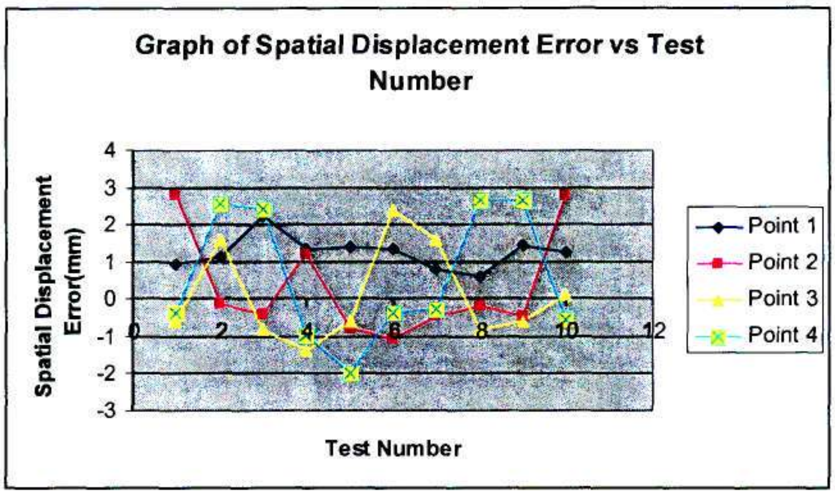


Figure 6-13: Three dimensional displacement errors involved with the sensor positioning process

Inspection of a support bracket was conducted in order to test system accuracy and repeatability. Figure 6-14(a) shows the reference part used for the inspection routine. Figure 6-14(b) shows the defective areas in the inspected part. This part was inspected with the conveyor operating at maximum speed. Four ROI were selected by the user in anticipation of defective ROI. Figure 6-15(a) and 6-15(b) show the machine whilst it was operating.

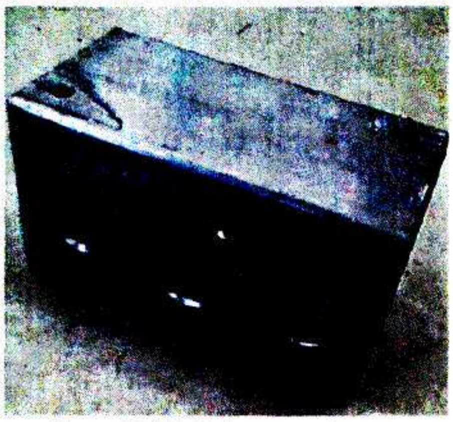


Figure 6-14(a): Reference part

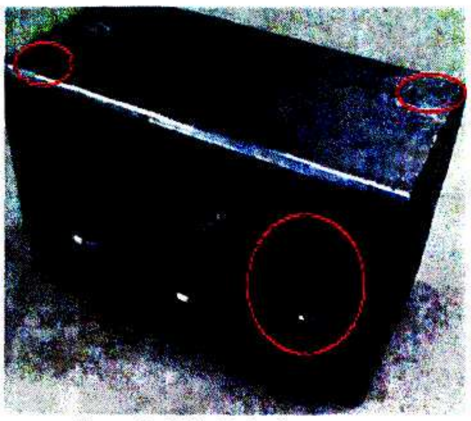


Figure 6-14(b): Defective part

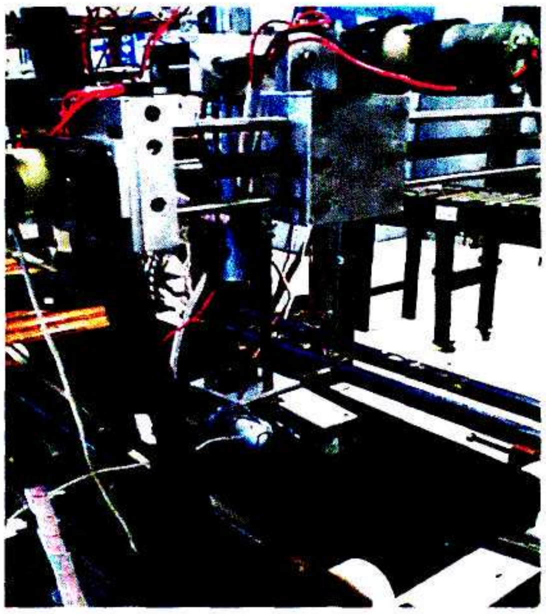


Figure 6-15(a): Operating NCAIS

Figure 6-15(b): Top-face ROI inspection

Figures 6-16(a) through (d) show the isolated images of the background, reference and test ROI, as well as the result after image comparison, from top to bottom respectively. The part was rejected due to significant differences between the reference and test images for the different ROI. The part would have been rejected even if only one ROI did not conform to the specifications. The location of the ROI that were defective was then stored and this information could have been used to identify flaws in a manufacturing process that were implemented for these ROI.

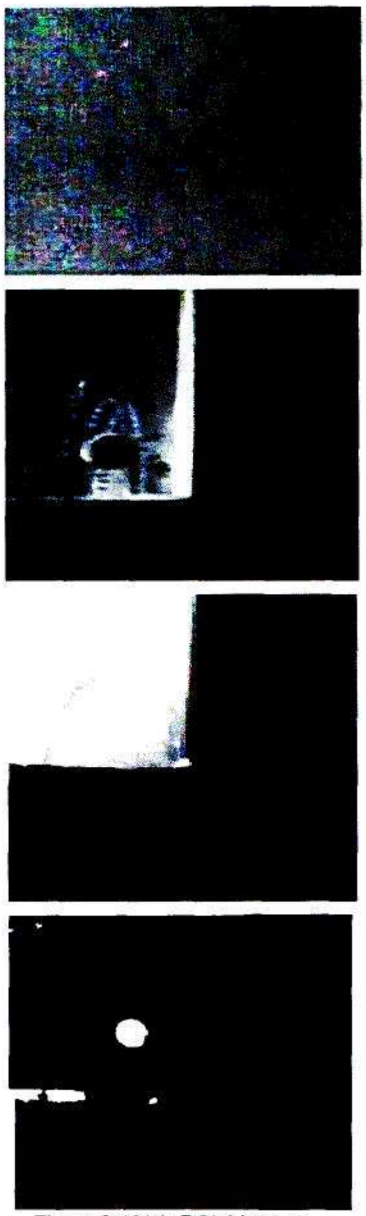


Figure 6-16(a): ROI 1 images

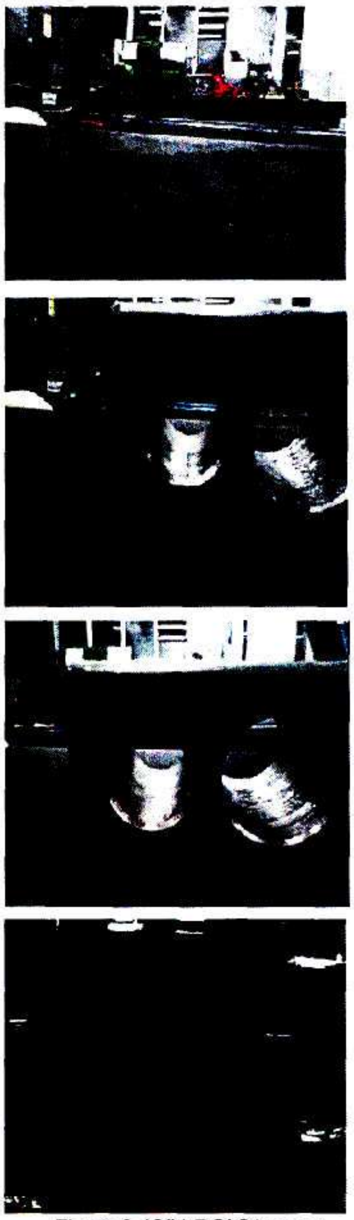


Figure 6-16(b) ROI 2 images

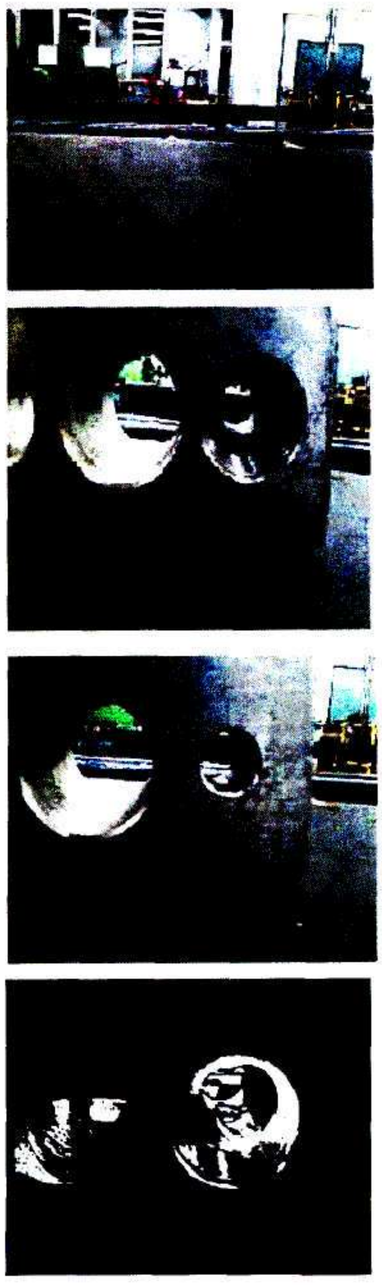


Figure 6-16(c): ROI 3 images

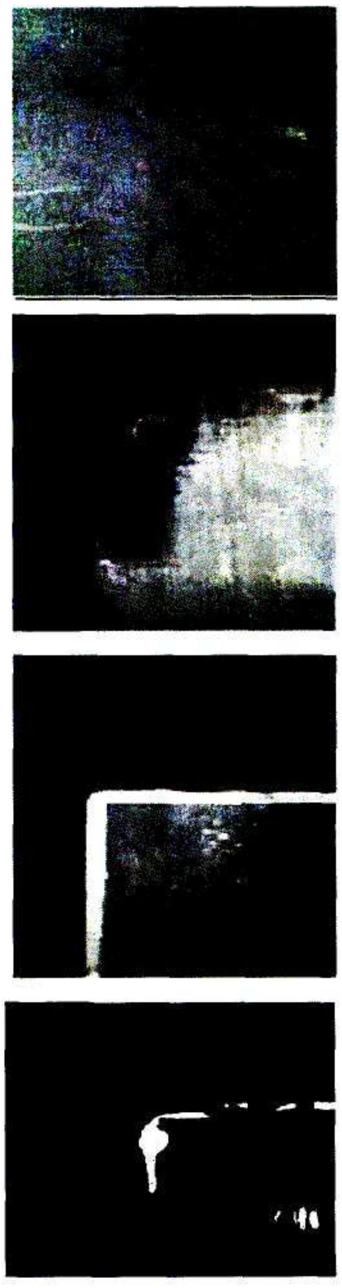


Figure 6-16(d) ROI 4 images

Table 6-2 shows the results of the reference part being inspected and the defective test part inspections. The reference part was inspected to test whether the system would give erroneous results when an acceptable part was inspected. The results of an inspection routine on an acceptable part are shown in Appendix A.

Attempt Number	Status of perfect part	Status of defective part
1	Acceptable	Unacceptable
2	Acceptable	Unacceptable
3	Acceptable	Unacceptable
4	Acceptable	Unacceptable
4	Acceptable	Unacceptable

Table 6-2: Repeatability of results for identical tests

The focus of the research was to maintain the maximum production rate of 0.02 m/s, in the CIM cell. This result was achieved for a particular part that resembled both a mass-produced custom part, and a part resulting from VAM (in which an Aluminium block would have value added to it by means of adding threaded holes at specified locations). The maximum production rate maintained was suitable for the research purposes of the CIM cell. Industrial applications may require faster, more accurate, and more precise quality control and inspection routines. This may be achieved by implementation of higher precision ball screws, better imaging sensors, and more specific imaging software. The results of the tests performed were specific to the operating parameters of the CIM cell; however the approaches of inspecting moving custom parts, and only key ROI on these parts, can be implemented in many industrial applications.

In order to consider the practical application of the apparatus, production rates involving multiple parts needed to be considered. The inspection routines were conducted within the 40 second time limit, averaging 35 seconds per inspection. An approximated average delay of 6 seconds (in addition to the inspection time) was required for the system to re-calibrate itself after each inspection routine for this particular product. This delay meant that the apparatus was prone to bottlenecking. A simulation was performed in Flexsim4 [57] to observe the operation of the apparatus in a manufacturing environment with production of multiple products. The model used for the simulation assumed that a part was produced every 30 seconds and a delay of 15 seconds to transport the part from the manufacturing process to the quality control apparatus. This 3D model is shown in Figure 6-17. The results from the simulation show that a bottleneck occurs at the NCAIS 600 seconds into operation under these conditions. After approximately 10000 seconds of operation, the bottleneck accumulates up to 5 items. This would be unacceptable to manufacturers since production rates are being hampered. Adding another inspection apparatus eliminated the bottleneck under these conditions and no bottleneck was observed after 500 000 seconds of operation.

The specified production rate could therefore be maintained, with delays in the processes, by implementing 2 parallel inspection processes.

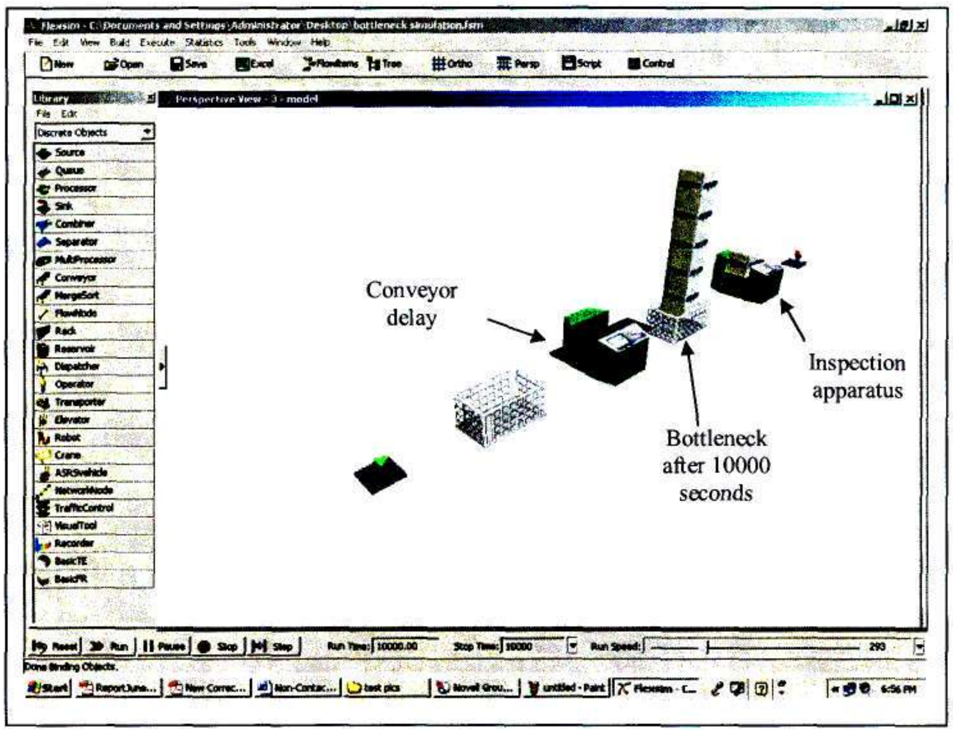


Figure 6-17: Graphical model used to simulate bottlenecking during operation

6.4 NCAIS suitability to industrial applications

An increased number of manufacturers will soon be forced to implement reconfigurable processes in all areas of manufacturing in order to maintain a competitive advantage. Amongst these areas is the requirement of quality control processes that must perform in-line inspection of moving parts. Manufacturing cells will have different throughputs and the inspection process must account for this. Furthermore, inspection requirements from each cell will differ and so the inspection process must be able to adapt for variations in inspection requirements without impacting significantly on production rates.

The NCAIS may be viewed as a prototype apparatus that can be implemented in such an environment. Figure 6-18 shows the typical conditions that the NCAIS will be exposed to in an industrial application. Some parts may require only drilling operations, whilst other parts may require drilling and turning operations. Manufacturing processes and layouts will have to be able to facilitate these requirements efficiently if high production rates are to be achieved. The differences in production rates for different parts must also be accounted for. Furthermore, inspection at various stages in the manufacture of a product must also be facilitated. The inspection process must then be able to facilitate inspections after drilling and turning. These inspections must not impede the production rates of the manufacturing process. The inspection process must also be able to measure any inconsistencies in parts as they progress through their manufacturing lifecycles. If there are any defective products being produced, then the manufacturing process must have the capability of early detection of these flawed products to minimise losses.

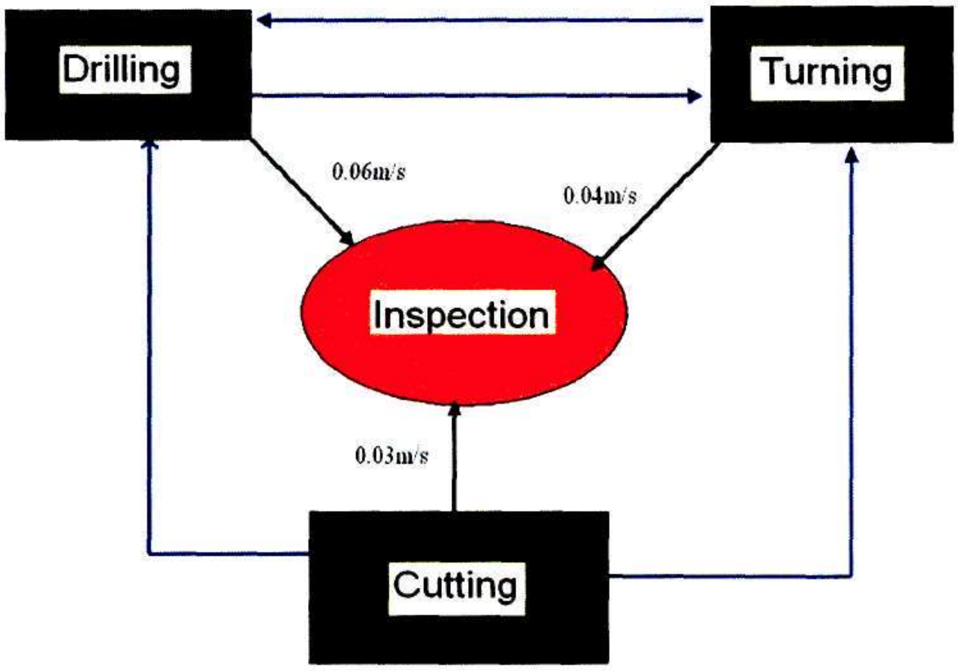


Figure 6-18: Typical operating sequence for part inspection in RME

6.5 Summary of chapter 6

This chapter listed the obtained results of the various subsystems. The image processing results were obtained and discussed. The algorithms used for checking the differences between the reference and test images proved to be sufficient in operation. The accuracy and repeatability of the positioning of the sensor was found to be within the given specifications.

7. CONCLUSION

The purpose of this project was to research, design, construct, assemble, and test an automated apparatus that would perform inspection of custom parts whilst defending production rates. Two methods of decreasing inspection times were conceptualised. The first method researched was to inspect only significant ROI. This method is effective and more accurate than inspecting an entire face of a large product. The implementation of ROI inspection meant that only features that required inspection were inspected. This increased overall system reliability and efficiency in terms of time. The second method that was researched was the inspection of ROI whilst the parts were moving. This allowed for in-line inspection, which required dynamic access to the various ROI. These two methods of decreasing inspection times required accurate and repeatable sensor positioning. Various methods of sensor manipulation were researched. Ultimately, a gantry structure with a pan-tilt mechanism was implemented due to its suitability to the application, as well as the suitability of this structure to be implemented into the existing AVIS framework.

The application of inspecting custom parts required a method of sensing that was diverse, time efficient, and low-cost. Non-contact sensing methods were preferred and in particular, visual inspection was implemented. Vision systems offer extreme diversity, and are low cost. The use of a vision system required research about the different types of vision sensors; the different types of lighting conditions; and the different types of software applications that were required for performing image processing operations. Inspection was based on template matching for which a reference part was required. Based on a predefined threshold, and comparison results between an inspected part and the reference part, a product was either accepted or rejected. CAD based inspections, involving 3D models of inspected parts, would have been more accurate and reliable, however the design and implementation of these models would have been required prior to the manufacturing process. Since the CIM cell did not implement such a design method, this form of inspection was not implemented.

The Mechatronic engineering approach of system integration was used to research, design, construct, assemble, and test an apparatus that performed automated inspection of ROI on moving custom parts. The core elements of Mechatronic engineering were considered simultaneously in the design of subsystems. Using the Mechatronic engineering philosophy, the functionality of each subsystem on all levels of the design was considered to be interdependent. This approach allowed for the successful and optimal merging of vision, sensor articulation, and control subsystems. Parts inspected were categorised into three part families. Mechanical sensor articulation components were merged with electronic hardware for feedback and control purposes. Imaging hardware was selected based on system constraints; the main constraints being spatial and cost. Motors were selected based on the torque requirements and required motion of the different applications such as conveyor operation and sensor positioning. Motor control hardware was implemented based on the characteristics of the motors. Software was developed in CVAVR to manage the low-level operation of the motors and sensors used for the inspection routine. OpenCV was used for image acquisition and processing, and formed the basis of the vision system software. Utility programs were written using this software for performing various image processing functions. These functions included log-polar and Fourier correlation algorithms for image registration; deconvolution operations for image deblurring; and image transformation algorithms for performing colour and greyscale image operations such as colour and intensity analyses. High level general system management was developed in MATLAB for coordinating the entire inspection process. This coordination involved the identification of the part that required inspection; loading the necessary inspection parameters; controlling the operation of the sensor articulation circuitry and mechanical members; synchronising the image acquisition process with the sensor positioning; implementing the appropriate image processing utility functions as required; as well as executing data transfers from sensors to the host PC and from the host PC to the slave microcontroller.

The constructed apparatus was tested in a CIM cell that operated at a maximum conveyor speed of 20mm/s. The apparatus was able to perform in-line inspection of custom parts which were moving at the maximum speed. Inspection of ROI increased the reliability of the inspection process, and also informed about the location of flaws if detected. The operation of the SPS using open loop control introduced some errors in the positioning of the imaging sensor. These errors were not as significant as anticipated since the image processing allowed for registration of images, which increased the tolerances on the positioning requirements of the sensor. The apparatus displayed characteristics of a reconfigurable machine. It was modular in its mechanical design of the x and y axes. The electronic control circuits were modular. The machine was able to reconfigure itself to obtain images at different angles and sensor positions. The customized flexibility lay in the fact that the system was only designed for three part families. Ability to account for varying throughputs and part dimensions allowed the apparatus to display some form of scalability. The NCAIS also displayed potential to be efficiently employed in VAM environments by use of ROI inspection. The speed of the CIM cell operation may not always be applicable to industrial applications. The prototype may however be modified to account for high speed operations. The mechanical modifications that can be implemented are the use of lead screws with a higher pitch. This would increase the translational speed as a function of the rotational input. The number of starting threads could also be increased. This would lead to a proportional increase in the translational velocity. For example, increasing the number of starting threads from one to two would double the translational velocity of the system. ROI inspection provided the capability of the system to provide feedback about the location of detected flaws. This information could have then been used to isolate inefficiencies in processes. Use of ROI inspection proved to be effective; however the manual selection of these ROI still required some form of human input in the process. In future, when implementing a CAD based model inspection system, an algorithm may be developed for identifying ROI, based on a stress analysis or predefined process information. Another deficiency in the system was the limitations of the implemented kinematic model. This model was limited to 8 ROI; however the path of the imaging sensor during inspection was dependent upon the speed of the part as well as the number of ROI. It was possible that the user selected ROI and part speeds that would only result in colliding paths which would have lead to rejection of the inspection routine itself. Future algorithms may implement machine learning techniques for mapping out a trajectory based on sensed information as opposed to predefined constraints. Even though research was aimed at one part family, the methods could have been extended to other part families, increasing system diversity. The research presented was based on the characteristics of the CIM cell that was used to test the apparatus, and was not focused on any particular manufacturing industry. Further research may involve more detailed analyses in support of a quality control apparatus for a specific manufacturing industry.

In conclusion, the following were achieved:

- Research, design, construction, implementation, and testing of an apparatus called the NCAIS for performing visual inspection of moving custom parts within RME
- Inspection of rectangular-volume custom parts travelling at the specified maximum speed of 20mm/s without stopping the part at any point in time during the inspection. This allowed for the maximum production rate in the CIM cell to be maintained
- Implementation of a kinematic model used for optimising the sensor trajectory to some extent. This allowed for incorporation of some form of intelligence to the machine
- All design specifications were met

8. REFERENCES

[1] Groover MP, "Fundamentals of Modern Manufacturing: Materials, Processes, and Systems", Prentice Hall, Upper Saddle River, New Jersey 07458, 1996.

[2] Mayor J.R.S (2000), "Automated Visual Inspection Apparatus for Flexible Manufacturing System", Chapters 1, 3. In PhD dissertation, Department of Mechanical Engineering, University of Natal

[3] Batchelor BG, Hill DA, Hodgson DC, " Automated Visual Inspection", IFS Publications Ltd, UK North-Holland, 1985.

[4] SY Nof, "Handbook of industrial robotics", 1985, p.1173-1240

[5] Zhang HC, Alting L, "Computerized Manufacturing Process Planning Systems" Springer, 1994.

[6] J Barhak, D Djurdjanovic, P Spicer, R Katz, "Integration of reconfigurable inspection with stream of variations methodology," International Journal of Machine Tools and Manufacture 45, 2005.

[7] Merhabi MG, Ulsoy AG, Koren Y, "Reconfigurable Manufacturing Systems: Key to Future Manufacturing", Journal of Intelligent Manufacturing, Volume 11, 2004.

[8] Tseng MT, Piller FT, "The Customer Centric Enterprise: Advances in Mass Customization and Personalization", Springer, 2003.

[9] G Bright, S Davrajh, "Intelligent quality control apparatus for inspection of moving custom parts", CARS&FOF Japan, July 2008, pp on CD

[10] http://industrialpaints.globalspec.com/LearnMore/Manufacturing_Process_ Equipment/Inspection_Tools_Instruments/Coordinate_Measuring_Machines_CMM, 16 May 2007

[11] Zhenhua Huang, AJ Shih, J Ni, "Three-dimensional optical measurements for automotive manufacturing," International Conference on Competitive Manufacturing, COMA 2007, p35-39

[12] http://www.lionprecision.com/inductive-sensors/index.html, 15 April 2007

[13] http://www.lionprecision.com/capacitive-sensors/index.html, 15 April 2007

[14] http://www.ferret.com.au/c/Micromax/Ultrasonic-sensors-provide-flexible-solutions-n686554, 15 April 2007

[15] Prasanthi Guda, Jin Cao, Jeannine Gailey and Ernest L, " Machine Vision Fundamentals", Prentice Hall, Center for Robotics Research, 2003, pp 6-61

[16] RC Gonzalez, "Digital Image Processing", Prentice Hall, New Jersey, 2002.

[17] http://www.machinevisiononline.org/public/articles/archivedetails.cfm?id=2870,12 June 2007.

[18] W Bolton, "Mechatronics electronic control systems in mechanical engineering", Addison Wesley Longman Limited, 1995.

[19] W Stadler, "Analytical Robotics and Mechatronics", McGraw Hill Inc, 1995.

[20] http://www.yaskawa.com/images/News/Mechatronics.jpg, 23 July 2007

[21] http://www.nationmaster.com/encyclopedia/Mechatronics-engineering, 16 August 2007

[22] Bolhouse V, "Fundamentals of Machine Vision", Robotic Industries Assn, 1999.

[23] DJ Carney, EJ Morris, PRH Place, "Identifying Commercial Off-the Shelf (COTS) Product Risks: The COTS Usage Risk Evaluation", Technical Report CMU/SEI TR 023, 2003.

[24] David Bradley, Derek Seward, David Dawson, and Stuart Burge, "Mechatronics and the design of intelligent machines and systems" Stanley Thornes, 2000.

[25] http://www.und.ac.za/und/mechatron/cim_cell.htm, 28 November

[26] http://virtualphysics.imrc.kist.re.kr/pjoint.gif, 12 july 2007

[27] IRB 340 – Flex-Picker Datasheet from www.abb.com/robots, 13 September 2007

[28] http://en.wikipedia.org/wiki/Serial_manipulator, 12 August 2007

[29] http://www.directindustry.com/industrial-manufacturer/cartesian-robot-63269.html, 11 October 2007

[30] http://www.americanrobot.com/images_products/gantry_5_t.jpg, 11 October 2007

[31] http://erc.engin.umich.edu/publications/KorenRMS.pdf, May 2007

[32] http://www.directindustry.com/industrial-manufacturer/actuator-67149.html, 12 August 2007

[33] http://www.secureusa.net/product%20pdf/Electrics%20over%20Hydraulics%20or%20Pneumatics.pdf, 20 November 2007

[34] http://www.boltscience.com/pages/screw3.htm

[35] JG Drotsky, "Strength of Materials for Technicians", Heinemann Publishers, Sandton, 2004.

[36] RC Stephens, "Strength of Materials: Theory and examples", Edward Arnold Inc, London, 1974

[37] http://www.aksteel.com/pdf/markets_products/stainless/austenitic/304_304L_Data_Bulletin.pdf, 23 November 2007

[38] http://www.ami.ac.uk/courses/topics/0123_mpm/index.html , 12 October 2007

[39] http://www.idcmotion.com/pdf/9009.pdf, 14 October 2007

[40] http://www.motioncontrolonline.org/files/public/Motion_Control_Basics.pdf,12 October 2007

[41] H McCallion,"Vibration of linear mechanical systems", Longman Group Ltd, 1973.

[42] http://www.toolpartsdirect.com/type/Cordless-Drill-Parts, 23 August 2008.

[43] http://www.dliengineering.com/vibman/planetarygears.htm, 11 December 2007

[44] http://www.seattlerobotics.org/guide/servos.html, 23 November 2007

[45] www.electrocom.com.au/pdfs/**RFID**%20vs%20**Barcodes**.pdf, 26 October 2007

[46] www.indiabusinessweek.com/Opinion/Editorial/**RFID**.html, 26 October 2007

[47] www.sensorcentral.com/vision/tech01, 25 February 2008

[48] http://www.barcodediscount.com/catalog/psc/vs1200.htm, 27 October 2007

[49] http://ezinearticles.com/?How-Does-A-Barcode-Scanner-Work?&id=377318,26 October 2007

[50] http://www.keyence.com/enews/vision/cv/expert_2.php?en=0811ja, 27 July 2008

[51] K Ogata, "Modern Control Engineering: Fourth edition", Prentice Hall Inc, Upper Saddle River, 2002

[52] www.atmel.com/dyn/resources/prod_documents/doc8162.pdf, 20 April 2008

[53] www.graftek.com/pdf/Marketing/**MachineVisionLighting**.pdf, 28 February 2008

[54] http://www.xpercept.com/opencv.htm, 3 March 20008

[55] http://www.roborealm.com/, 3 March 2008

[56] http://www-structmed.cimr.cam.ac.uk/Course/Convolution/convolution.html#corr_theorem, 23 March 2008

[57] http://www.automated@flexsim.com , 23 November 2008

[58] http://www.misumiusa.com/uploadedFiles/Technical/METRIC1805-1806.pdf, 25 February 2009

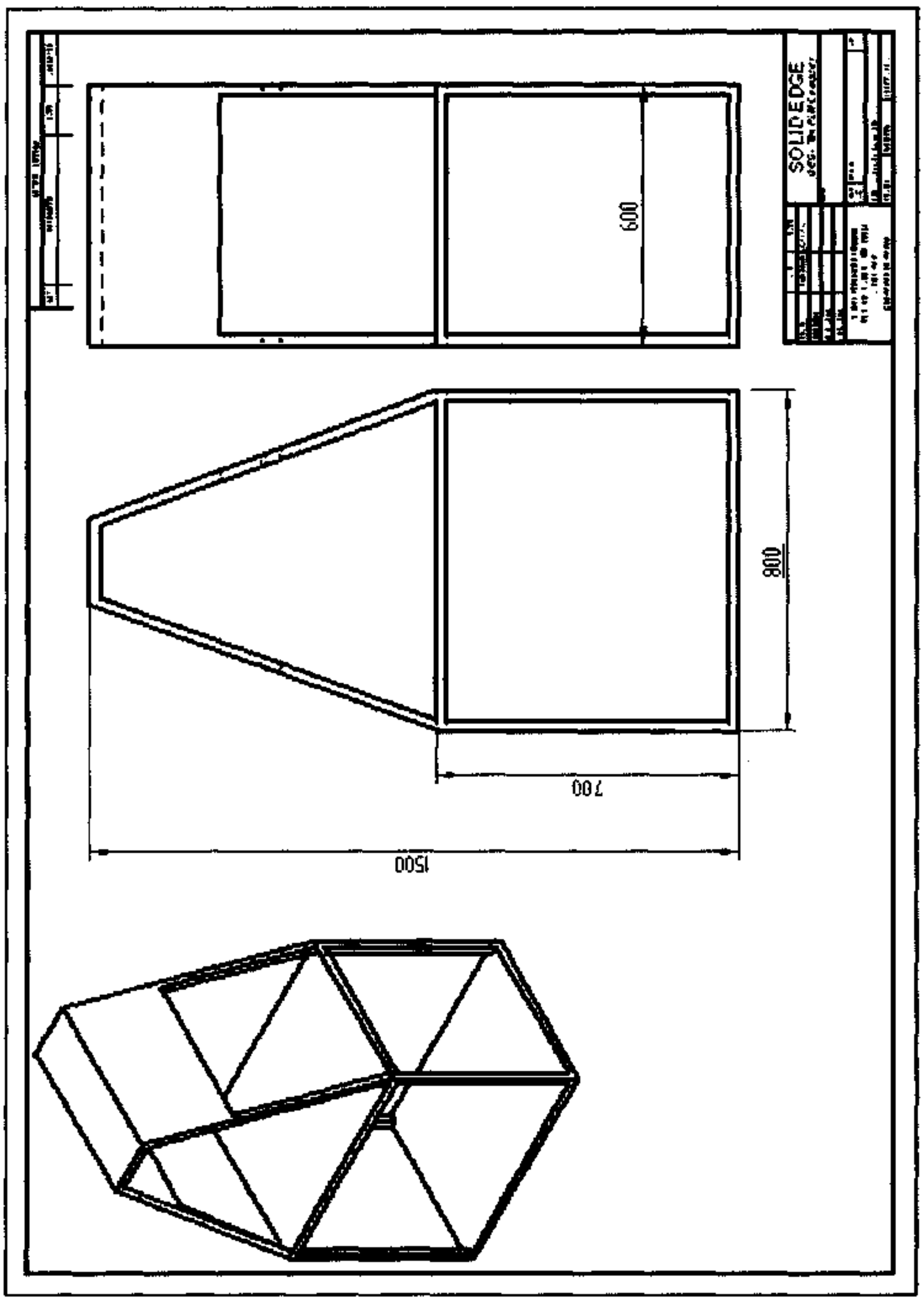
[59] T Philpot, " Mechanics of Materials- An integrated learning system", John Wiley and Sons, 111 River Street, Hoboken, New Jersey, 2008, Chapter 16

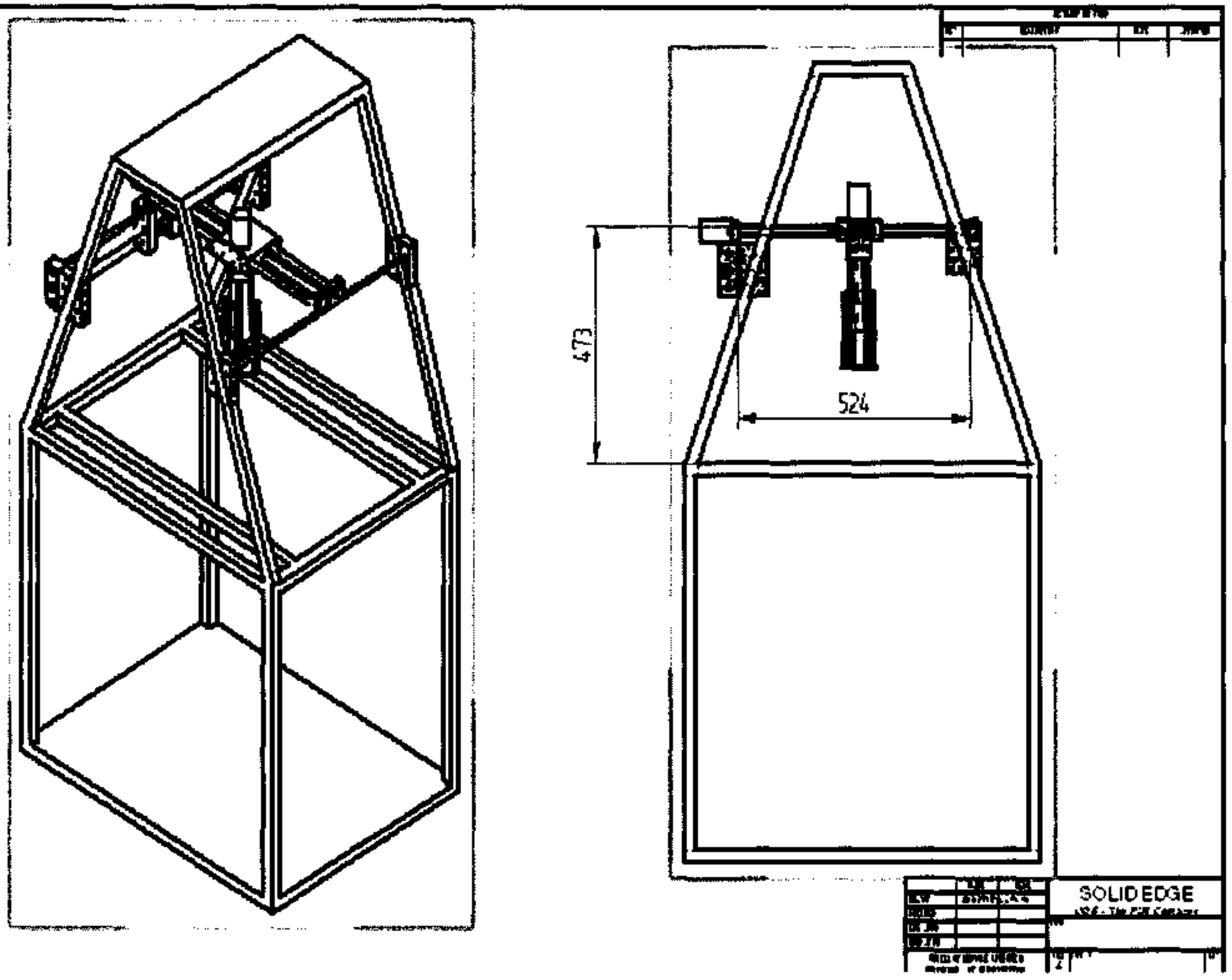
[60] http://www.ciao.com/Microsoft_LifeCam_VX_6000__15220321#productdetail, 25 February 2009

9. APPENDICES

9.1 Appendix A

9.1.1 System Drawings




473
524
SOLID EDGE

9.1.2 Inspection routine of acceptable part

An Inspection routine was performed on two identical boxes to test the apparatus for inspection of an acceptable part. These boxes were used for housing circuits. The two ROI for each of these parts were the face containing the connections, as well as the face containing the fuse as shown in Figure 9-1 (a) and (b). One box was used as the reference part, and the second box was considered the test part.

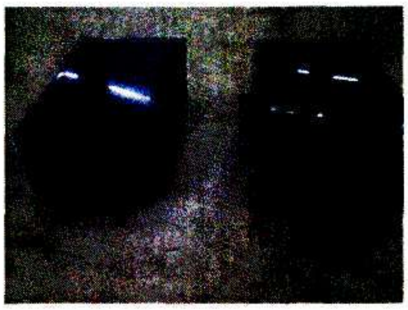


Figure 9-1(a): ROI 1 of the test and reference parts

Figure 9-1(b): ROI 2 on test and reference parts

The images acquired for ROI 1 are shown in Figure 9-2. The image processing involved with these images is discussed in section 6.2. Figure 9-3 shows the converted greyscale images of the original images, as well as the resulting difference between the two images after the comparison process from top to bottom respectively. A small difference between the images was found at the top left corner. This was most likely due to the difference in background as a result of imprecise camera positioning. Even though a difference between the images was obtained, the ROI was accepted due to the difference being insignificant.

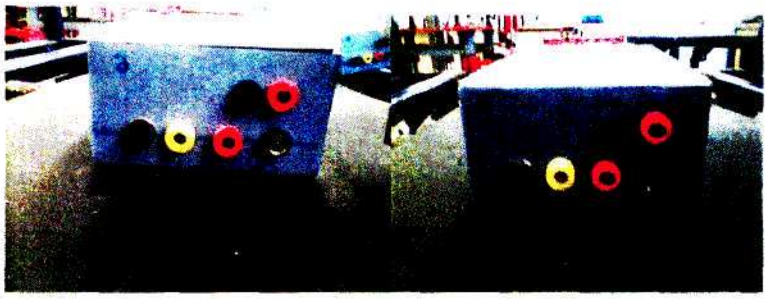


Figure 9-2: ROI 1 of reference and test part respectively

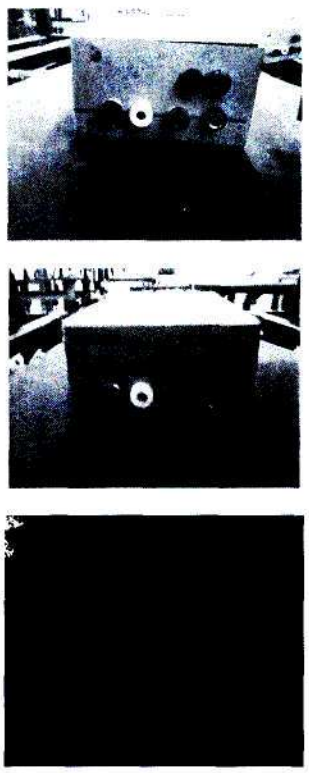


Figure 9-3: Greyscale images of the acquired images as well as their difference after the comparison process for ROI 1

The images acquired for ROI 2 are shown in Figure 9-4. The image processing involved with these images were the same as discussed in section 6.2. Figure 9-5 shows the converted greyscale images of the original images and the difference between the two images, in the same layout as shown for ROI 1. Differences between the test and reference images were most likely due to the different background portions in the images. Due to the insignificant difference between the images after comparison, this ROI was also accepted.

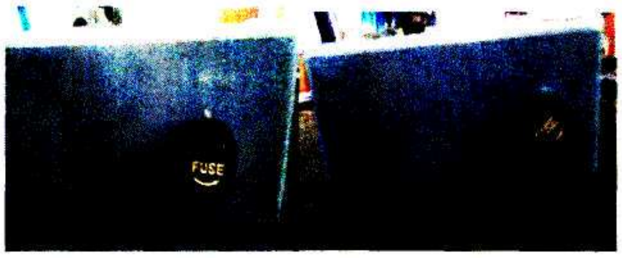


Figure 9-4: ROI 2 of reference and test part respectively

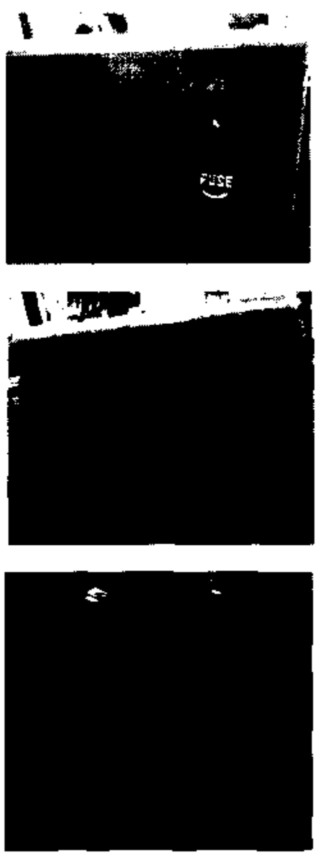


Figure 9-5: Greyscale images of the acquired images as well as their difference after the comparison process for ROI 2

It can be seen that the apparatus was capable of outputting a successful result for acceptable parts. The ROI were accepted even thought there were translational, rotational, and lighting inconsistencies present between the two images. The limits at which the inconsistencies became significant were not explored because the accuracy of the Sensor Positioning System (see section 6.3.4) was within tolerances, as specified in section 1.5. The accuracy of the system was sufficient for the scope of the research, however; more extensive and conclusive tests may be performed when the system is applied to a specific industrial application.

9.2 Appendix B

x-axis results

Desired distance	actual distance 1	actual distance 2	actual distance 3
20	21.50	21.00	21.50
40	38.50	39.00	38.50
60	59.50	58.00	58.50
80	80.50	81.50	81.50
100	100.00	99.00	100.50
120	119.50	118.00	119.00
140	137.50	138.50	138.00
160	158.50	161.00	159.00
180	181.00	182.00	182.00
200	203.00	201.50	202.50
220	221.50	221.00	220.50
240	242.50	241.50	241.50
260	263.00	262.50	263.00
280	281.00	282.00	282.50
300	301.50	302.50	302.50
320	322.00	321.50	322.00
360	358.50	359.00	358.00
380	379.00	381.00	379.50

	error	error	error
	1.50	1.00	1.50
	-1.50	-1.00	-1.50
	-0.50	-2.00	-1.50
	0.50	1.50	1.50
	0.00	-1.00	0.50
	-0.50	-2.00	-1.00
	-2.50	-1.50	-2.00
	-1.50	1.00	-1.00
	1.00	2.00	2.00
	3.00	1.50	2.50
	1.50	1.00	0.50
	2.50	1.50	1.50
	3.00	2.50	3.00
	1.00	2.00	2.50
	1.50	2.50	2.50
	2.00	1.50	2.00
	-1.50	-1.00	-2.00
	-1.00	1.00	-0.50
	average error	0.55	

percentage error	percentage error	percentage error
7.50	5.00	7.50
-3.75	-2.50	-3.75
-0.83	-3.33	-2.50
0.63	1.88	1.88
0.00	-1.00	0.50
-0.42	-1.67	-0.83
-1.79	-1.07	-1.43
-0.94	0.63	-0.63
0.56	1.11	1.11
1.50	0.75	1.25
0.68	0.45	0.23
1.04	0.63	0.63
1.15	0.96	1.15
0.36	0.71	0.89
0.50	0.83	0.83
0.63	0.47	0.63
-0.42	-0.28	-0.56
-0.26	0.26	-0.13

y-axis results

Desired distance (mm)	Actual distance 1	Actual distance 2	Actual distance 3
0	0	0	0
20	19.5	20	19.5
40	40.5	40.5	40
60	61.5	60.5	60.5
80	82	81.5	81
100	101.5	101	101
120	121.5	121	122
140	141.5	142	141
160	161.5	162	162
180	182.5	182	181.5
200	202.5	203	203
220	223	223.5	223
240	243	242.5	243
260	262.5	263	263
280	283	282.5	283
300	302.5	302	303
320	320.5	321	320
340	338	339	338.5
360	357	359	357.5

Error (mm)	Error (mm)	Error (mm)
0	0	0
-0.5	0	-0.5
0.5	0.5	0
1.5	0.5	0.5
2	1.5	1
1.5	1	1
1.5	1	2
1.5	2	1
1.5	2	2
2.5	2	1.5
2.5	3	3
3	3.5	3
3	2.5	3
2.5	3	3
3	2.5	3
2.5	2	3
0.5	1	0
-2	-1	-1.5
-3	-1	-2.5

percentage error	percentage error	percentage error
0.00	0.00	0.00
-2.50	0.00	-2.50
1.25	1.25	0.00
2.50	0.83	0.83
2.50	1.88	1.25
1.50	1.00	1.00
1.25	0.83	1.67
1.07	1.43	0.71
0.94	1.25	1.25
1.39	1.11	0.83
1.25	1.50	1.50
1.36	1.59	1.36
1.25	1.04	1.25
0.96	1.15	1.15
1.07	0.89	1.07
0.83	0.67	1.00
0.16	0.31	0.00
-0.59	-0.29	-0.44
-0.83	-0.28	-0.69

Overall system results

Attempt number	kinematic model distance	actual distance	error	percentage error
Point 1				
1.00	30.00	30.93	0.93	3.10
2.00	30.00	31.13	1.13	3.76
3.00	30.00	32.04	2.04	6.80
4.00	30.00	31.38	1.38	4.59
5.00	30.00	31.38	1.38	4.62
6.00	30.00	31.38	1.38	4.59
7.00	30.00	30.81	0.81	2.70
8.00	30.00	30.60	0.60	2.01
9.00	30.00	31.45	1.45	4.84
10.00	30.00	31.25	1.25	4.16
Point 2				
1.00	87.46	90.27	2.81	3.21
2.00	87.46	87.32	-0.14	-0.17
3.00	87.46	87.06	-0.41	-0.46
4.00	87.46	88.68	1.21	1.39
5.00	87.46	86.69	-0.78	-0.89
6.00	87.46	86.40	-1.06	-1.21
7.00	87.46	87.03	-0.44	-0.50
8.00	87.46	87.31	-0.16	-0.18
9.00	87.46	87.03	-0.44	-0.50
10.00	87.46	90.27	2.81	3.21
Point 3				
1.00	72.28	71.71	-0.57	-0.79
2.00	72.28	73.88	1.59	2.21
3.00	72.28	71.47	-0.82	-1.13
4.00	72.28	70.92	-1.37	-1.89
5.00	72.28	71.70	-0.58	-0.80
6.00	72.28	74.68	2.40	3.32
7.00	72.28	73.88	1.60	2.21
8.00	72.28	71.47	-0.81	-1.12
9.00	72.28	71.70	-0.59	-0.81
10.00	72.28	72.40	0.12	0.16
Point 4				
1.00	122.47	122.14	-0.34	-0.28
2.00	122.47	125.02	2.55	2.08
3.00	122.47	124.90	2.43	1.98
4.00	122.47	121.50	-0.98	-0.80
5.00	122.47	120.50	-1.97	-1.61
6.00	122.47	122.12	-0.36	-0.29
7.00	122.47	122.21	-0.26	-0.21
8.00	122.47	125.14	2.67	2.18
9.00	122.47	125.12	2.65	2.16
10.00	122.47	121.95	-0.52	-0.43

Desired x distance	Actual x distance	Desired y distance	Actual y distance	Desired z distance	Actual z distance
20.00	19.50	20.00	20.50	10.00	12.50
20.00	20.00	20.00	20.00	10.00	13.00
20.00	21.50	20.00	20.50	10.00	12.00
20.00	20.50	20.00	20.50	10.00	12.00
20.00	21.00	20.00	20.00	10.00	12.00
20.00	20.50	20.00	20.50	10.00	12.00
20.00	20.00	20.00	19.50	10.00	13.00
20.00	19.50	20.00	20.00	10.00	12.50
20.00	20.00	20.00	20.50	10.00	13.00
20.00	20.50	20.00	20.00	10.00	12.50
35.00	37.00	80.00	82.00	5.00	7.50
35.00	37.50	80.00	78.50	5.00	7.50
35.00	36.00	80.00	79.00	5.00	6.50
35.00	35.50	80.00	81.00	5.00	6.50
35.00	36.00	80.00	78.50	5.00	7.50
35.00	36.50	80.00	78.00	5.00	7.00
35.00	37.00	80.00	78.50	5.00	6.50
35.00	36.50	80.00	79.00	5.00	7.00
35.00	37.00	80.00	78.50	5.00	6.50
35.00	37.00	80.00	82.00	5.00	7.50
50.00	48.50	50.00	50.50	15.00	15.50
50.00	51.00	50.00	51.00	15.00	16.00
50.00	49.00	50.00	49.50	15.00	16.00
50.00	48.00	50.00	50.00	15.00	15.00
50.00	50.00	50.00	49.00	15.00	15.50
50.00	51.00	50.00	52.00	15.00	16.50
50.00	50.50	50.00	51.50	15.00	16.00
50.00	48.50	50.00	50.00	15.00	16.00
50.00	49.50	50.00	49.50	15.00	15.50
50.00	49.00	50.00	51.00	15.00	15.50
100.00	100.00	70.00	69.00	10.00	12.50
100.00	101.50	70.00	72.00	10.00	12.00
100.00	101.00	70.00	72.50	10.00	12.00
100.00	99.50	70.00	68.50	10.00	13.00
100.00	98.00	70.00	69.00	10.00	12.50
100.00	98.50	70.00	71.00	10.00	13.00
100.00	100.50	70.00	68.50	10.00	12.00
100.00	102.00	70.00	71.50	10.00	12.00
100.00	101.50	70.00	72.00	10.00	13.00
100.00	98.00	70.00	71.50	10.00	12.50

9.3 Appendix C- Product datasheets and specifications

Motor specifications [42]

Keyless chuck- 10 mm

No load speed- 0 to 400rpm, 0 to1250rpm

Max drilling capacities:

Wood- 20 mm

Steel- 10 mm

Torque settings- 24

Max Torque- 27Nm

Weight- 1.7 kg

Barcode scanner specifications [48]

VS1200 & VS1000
Vertical Scanners
PSC are designed specifcally to optimize performance in vertical applications.
Unlike many other so-called vertical scanners which are little more than horizontal scanners turned on their sides, the competitively priced VS1200 and VS1000 are "true vertical" scanners. Aggressive, fast, and accurate each has a large, symmetrical scan volume with a generous "sweet spot" to maximize productivity and ergonomics.
All VS models boast a small footprint and are easily installed onto any countertop with a simple stationary or flexible stand mount.
Models For The Codes You Read...
The VS1200 and the VS1000 are each available in two formats. To select the VS scanner that's right for your application, first determine the codes that you will need to read in your environment.
The VS1200 comes standard with UPC/EAN/JAN decoding and advanced EdgeTM software for reading torn or disfigured labels. With Edge disabled, the VS1200 reads the UPC family plus one additional industrial code, and reads poorly-printed labels with the help of AdaptiveTM software. Optionally, the VS1200 can be ordered with P2/P5 add-ons. Adaptive software is standard on the VS1000, which auto-discriminates between up to four codes including UPC/EAN/JAN with P2/P5 add-ons, and industrial symbologies such as ITF, Code 128, and Code 39.
...And How You Read Them
Examine your scanning environment—the VS model you choose will depend on available counter space, the method used to scan items, and the rate at which items must be scanned.
Both the VS1200 and the VS1000 are available in "Sweep" and "Presentation-style" scanning models. In higher volume checkout environments with sufficient room for a left-to-right or right-to-left scanning motion, choose a sweep configuration. In lower volume scanning environments with limited counter space, the presentation configurations' deep, dense scan pattern will provide optimal performance.
The VS1200 & VS1000: Optimized For Your Application
Your scanning requirements are unique, and your scanner should be too. With the VS1200 & VS1000, you can have an affordable vertical scanner that's just right for your environment—without sacrificing performance, ergonomics or valuable counter space.
Ideal for:
• Designed for POS applications such
as grocery, drug, variety and

9.4 Appendix D – System code

9.4.1 Full Source Code for Kinematics Model and Associated Interactive Viewer

This program calculates the optimal camera path for a part given the part type, dimensions, conveyor belt velocity, camera clearance radius, initial camera position, the number of ROI present and the ROI positions. Other parameters that may be varied are the dimensions of the Cartesian robot and the physical dimensions of the camera (assuming it is box shaped). The output is an interactive (rotation and scaling can be keyboard controlled) line animation of the motion of camera and part. The output may be written to disk and accessed from MATLAB. Output consists of order of camera target positions, physical co-ordinates of camera target positions and the times the target positions must be reached. The reference time is the instant the front of the part enters the Cartesian robot and the reference origin is the lower left corner [looking in the direction of part motion] of the conveyor belt plane (i.e. bottom plane) of the Cartesian robot framework. The standard Borland C/C++ Compiler version 5.5 (Available free from [9]) was used for compilation [No non-standard libraries are required]. The standalone Microsoft Windows application must be run from the root directory for the system [Same as used for MATLAB]. For graphical output, screen resolution must be 1024x768 or greater.

```
//MODEL CODE:
//---------------------------------------------------------------------------
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream.h>
#include <math.h>
//---------------------------------------------------------------------------
struct vec3d
{
 double x,y,z;
};
struct vec3db
{
 byte x,y,z;
};
struct complex
{
 double x,y;
};
//---------------------------------------------------------------------------
void Initialize(void);
void Draw(void);
bool CreateWindowXX(char,int,int,int);
WINAPI WinMain(HINSTANCE,HINSTANCE,LPSTR,int);
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
```

```
//*****************************************************************************
***********
vec3d StdVec(vec3d,vec3d);
vec3d Sum(vec3d,vec3d);
vec3d Multiply(vec3d,double);
double Length(vec3d);
vec3d Direction(vec3d);
double Dot(vec3d,vec3d);
vec3d Cross(vec3d,vec3d);
vec3d VectorProjection(vec3d,vec3d);
double ScalarProjection(vec3d,vec3d);
double VectorAngle(vec3d,vec3d);
double LinePtDist(vec3d,vec3d,vec3d);
//---------------------------------------------------------------------------
vec3d Rotate(vec3d,vec3d);
int RGBX(int,int,int);
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//*****************************************************************************
***********
#define FRONT 0
#define BACK 1
#define LEFT 2
#define RIGHT 3
#define TOP 4
#define NUMROI 4
vec3d pts[NUMROI];
vec3d opt[NUMROI];
vec3d tgt[NUMROI];
vec3d spt[NUMROI];
double times[NUMROI];
double ctims[NUMROI];
double maxx=0.6;
double maxy=0.8;
double maxz=0.45;
double xmax=0.2;
double ymax=0.2;
double zmax=0.2;
double camx=0.04;
double camy=0.07;
double camz=0.04;
double vset=0.04;
double speedy=0.1;
double radiusc=0.042;
double d=0.2;
vec3d vpart={0.0,0.01,0.0};
vec3d pinit;
vec3d cinit={0.0,maxy*0.5,maxz*0.5};
int roipts[NUMROI][3];
vec3d boxdim={0.05,0.3,0.25};
double totaltime=maxy/vpart.y;
int roip[][3]={{FRONT,2,2},{BACK,1,2},{RIGHT,1,2},{RIGHT,1,1},{BACK,0,0},{TOP,0,2}};
int frames=0;
//---------------------------------------------------------------------------
byte scr[800*600*3];
double scd[800*600];
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
```

```
const double PI=3.1415926539;
const double rad=PI/180.0;
const double deg=180.0/PI;
//-----------------------------------------------------------------------------------------
vec3d cam={0.0,-400.0,-200.0};
vec3d rot={-100.0,0.0,30.0};
const vec3d
zerovector={0.0,0.0,0.0},ivector={1.0,0.0,0.0},jvector={0.0,1.0,0.0},kvector={0.0,0.0,1.0};
//-----------------------------------------------------------------------------------------
double sa=0.0,ca=0.0,sb=0.0,cb=0.0,sc=0.0,cc=0.0;
//-----------------------------------------------------------------------------------------
//*************************************************************************************
***********
HDC hDC=NULL;
HWND hWnd=NULL;
HINSTANCE hInstance=NULL;
//*************************************************************************************
***********
vec3d Rotate(vec3d pt,vec3d cp)
{
 vec3d rtp;
 double x,y,z,x2,y2,z2;
 x=pt.x-cp.x;
 y=pt.y-cp.y;
 z=pt.z-cp.z;
 x2=(x*cc*cb)-(y*sc*cb)+(z*sb);
 y2=(x*cc*sb*sa)+(x*sc*ca)+(y*cc*ca)-(y*sc*sb*sa)-(z*cb*sa);
 z2=(x*sc*sa)-(x*cc*sb*ca)+(y*sc*sb*ca)+(y*sa*cc)+(z*cb*ca);
 rtp.x=x2+cp.x;
 rtp.y=y2+cp.y;
 rtp.z=z2+cp.z;
 return rtp;
}
//-----------------------------------------------------------------------------------------
vec3d Rotate3D(vec3d pt,vec3d cp,vec3d roto)
{
 double cas,sas,cbs,sbs,ccs,scs;
 vec3d rtp;
 double x,y,z,x2,y2,z2;
 cas=cos(roto.x*rad);
 sas=sin(roto.x*rad);
 cbs=cos(roto.y*rad);
 sbs=sin(roto.y*rad);
 ccs=cos(roto.z*rad);
 scs=sin(roto.z*rad);
 x=pt.x-cp.x;
 y=pt.y-cp.y;
 z=pt.z-cp.z;
 x2=(x*ccs*cbs)-(y*scs*cbs)+(z*sbs);
 y2=(x*ccs*sbs*sas)+(x*scs*cas)+(y*ccs*cas)-(y*scs*sbs*sas)-(z*cbs*sas);
 z2=(x*scs*sas)-(x*ccs*sbs*cas)+(y*scs*sbs*cas)+(y*sas*ccs)+(z*cbs*cas);
 rtp.x=x2+cp.x;
 rtp.y=y2+cp.y;
 rtp.z=z2+cp.z;
 return rtp;
}
//-----------------------------------------------------------------------------------------
int RGBX(int r,int g,int b)
```

```
{
 return r+(g*256)+(b*65536);
}
//*****************************************************************************
***********
vec3d StdVec(vec3d v1,vec3d v2)
{
 vec3d vr;
 vr.x=v2.x-v1.x;
 vr.y=v2.y-v1.y;
 vr.z=v2.z-v1.z;
 return vr;
}
//--------------------------------------------------------------------------------
vec3d Sum(vec3d v1,vec3d v2)
{
 vec3d vr;
 vr.x=v1.x+v2.x;
 vr.y=v1.y+v2.y;
 vr.z=v1.z+v2.z;
 return vr;
}
//--------------------------------------------------------------------------------
vec3d Multiply(vec3d v1,double fac)
{
 vec3d vr;
 vr.x=v1.x*fac;
 vr.y=v1.y*fac;
 vr.z=v1.z*fac;
 return vr;
}
//--------------------------------------------------------------------------------
double Length(vec3d v)
{
 double l;
 l=sqrt((v.x*v.x)+(v.y*v.y)+(v.z*v.z));
 return l;
}
//--------------------------------------------------------------------------------
vec3d Direction(vec3d v)
{
 return Multiply(v,1.0/Length(v));
}
//--------------------------------------------------------------------------------
double Dot(vec3d v1,vec3d v2)
{
 return ((v1.x*v2.x)+(v1.y*v2.y)+(v1.z*v2.z));
}
//--------------------------------------------------------------------------------
vec3d Cross(vec3d v1,vec3d v2)
{
 vec3d vr;
 vr.x=(v1.y*v2.z)-(v1.z*v2.y);
 vr.y=(v1.z*v2.x)-(v1.x*v2.z);
 vr.z=(v1.x*v2.y)-(v1.y*v2.x);
 return vr;
}
//--------------------------------------------------------------------------------
```

```
double ScalarProjection(vec3d u,vec3d v)
{
 return Dot(u,Direction(v));
}
//---------------------------------------------------------------------------------------
vec3d VectorProjection(vec3d u,vec3d v)
{
 return Multiply(v,Dot(u,v)/(Length(v)*Length(v)));
}
//---------------------------------------------------------------------------------------
double VectorAngle(vec3d v1,vec3d v2)
{
 double angle;
 if((Length(v1)!=0)&&(Length(v2)!=0))
 {
  angle=acos(Dot(v1,v2)/(Length(v1)*Length(v2)));
 }
 else
 {
  angle=0;
 }
 return angle;
}
//---------------------------------------------------------------------------------------
double LinePtDist(vec3d lor,vec3d lde,vec3d pt)
{
 vec3d v=StdVec(lor,lde);
 return Length(Cross(StdVec(lor,pt),v))/Length(v);
}
//---------------------------------------------------------------------------------------
//*********************************************************************************
***********
complex Complex(double r,double i)
{
 complex c={r,i};
 return c;
}
//---------------------------------------------------------------------------------------
double Mod(complex z)
{
 return sqrt(pow(z.x,2.0)+pow(z.y,2.0));
}
//---------------------------------------------------------------------------------------
double Arg(complex z)
{
 double ang=0.0,pan;
 if(z.x==0.0){if(z.y>0.0){ang=PI*0.5;} if(z.y<0.0){ang=PI*1.5;}}
 if(z.y==0.0){if(z.x>0.0){ang=0.0;} if(z.x<0.0){ang=PI;}}
 if((z.x!=0.0)&&(z.y!=0.0))
 {
  pan=atan(fabs(z.y)/fabs(z.x));
  if((z.x>0.0)&&(z.y>0.0)){ang=pan;}
  if((z.x<0.0)&&(z.y>0.0)){ang=PI-pan;}
  if((z.x<0.0)&&(z.y<0.0)){ang=PI+pan;}
  if((z.x>0.0)&&(z.y<0.0)){ang=(2.0*PI)-pan;}
 }
 return ang;
}
```

```
//---------------------------------------------------------------------------------
complex Add(complex z1,complex z2)
{
 complex sum;
 sum.x=z1.x+z2.x;
 sum.y=z1.y+z2.y;
 return sum;
}
//---------------------------------------------------------------------------------
complex Subtract(complex z1,complex z2)
{
 complex difference;
 difference.x=z1.x-z2.x;
 difference.y=z1.y-z2.y;
 return difference;
}
//---------------------------------------------------------------------------------
complex Multiply(complex z1,complex z2)
{
 complex product;
 product.x=(z1.x*z2.x)-(z1.y*z2.y);
 product.y=(z1.x*z2.y)+(z1.y*z2.x);
 return product;
}
//---------------------------------------------------------------------------------
complex Divide(complex z1,complex z2)
{
 complex quotient={0.0,0.0};
 double den=pow(Mod(z2),2.0);
 if(den!=0.0)
 {
  quotient.x=((z1.x*z2.x)+(z1.y*z2.y))/den;
  quotient.y=((z1.y*z2.x)-(z1.x*z2.y))/den;
 }
 return quotient;
}
//---------------------------------------------------------------------------------
complex Exp(complex z)
{
 double mag=exp(z.x);
 complex expo;
 expo.x=mag*cos(z.y);
 expo.y=mag*sin(z.y);
 return expo;
}
//---------------------------------------------------------------------------------
complex Log(complex z)
{
 double mag=Mod(z);
 complex loga={0.0,0.0};
 if(mag>0.0)
 {
  loga.x=log(mag);
  loga.y=Arg(z);
 }
 return loga;
}
//---------------------------------------------------------------------------------
```

```
complex Sin(complex z)
{
 complex i1={0.0,1.0},i2={0.0,2.0},ineg={0.0,-1.0};
 return Divide(Subtract(Exp(Multiply(z,i1)),Exp(Multiply(z,ineg))),i2);
}
//------------------------------------------------------------------------------
complex Cos(complex z)
{
 complex i1={0.0,1.0},r2={2.0,0.0},ineg={0.0,-1.0};
 return Divide(Add(Exp(Multiply(z,i1)),Exp(Multiply(z,ineg))),r2);
}
//------------------------------------------------------------------------------
complex Sinh(complex z)
{
 complex rpos={1.0,0.0},rneg={-1.0,0.0},r2={2.0,0.0};
 return Divide(Subtract(Exp(Multiply(z,rpos)),Exp(Multiply(z,rneg))),r2);
}
//------------------------------------------------------------------------------
complex Cosh(complex z)
{
 complex rpos={1.0,0.0},rneg={-1.0,0.0},r2={2.0,0.0};
 return Divide(Add(Exp(Multiply(z,rpos)),Exp(Multiply(z,rneg))),r2);
}
//------------------------------------------------------------------------------
complex Power(complex z1,complex z2)
{
 return Exp(Multiply(Log(z1),z2));
}
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
//**************************************************************************************
***********
//**************************************************************************************
***********
//**************************************************************************************
***********
void PlotScr(int x,int y,int r,int g,int b)
{
 if((x>=1)&&(x<=800)&&(y>=1)&&(y<=600))
  {
   scr[((((y-1)*800)+x-1)*3)+2]=r;
   scr[((((y-1)*800)+x-1)*3)+1]=g;
   scr[((((y-1)*800)+x-1)*3)+0]=b;
  }
}
//------------------------------------------------------------------------------
void DrawLineE(int x1,int y1,int x2,int y2,int r,int g,int b)
{
 double grad;
 int dx=x2-x1,dy=y2-y1;
 if(dx==0)
  {
   if(y2>=y1)
    {
     for(int i=y1;i<=y2;i++){PlotScr(x1,i,r,g,b);}
    }
   else
    {
```

```
    for(int i=y2;i<=y1;i++){PlotScr(x1,i,r,g,b);}
   }
  }
 if(dy==0)
 {
  if(x2>=x1)
  {
   for(int i=x1;i<=x2;i++){PlotScr(i,y1,r,g,b);}
  }
  else
  {
   for(int i=x2;i<=x1;i++){PlotScr(i,y1,r,g,b);}
  }
 }
 if((dx!=0)&&(dy!=0))
 {
  if(abs(dx)>=abs(dy))
  {
   grad=(double)dy/(double)dx;
   if(x1<=x2)
   {
        for(int i=x1;i<=x2;i++){PlotScr(i,y1+(grad*(i-x1)),r,g,b);}
   }
   else
   {
        for(int i=x2;i<=x1;i++){PlotScr(i,y2+(grad*(i-x2)),r,g,b);}
   }
  }
  else
  {
   grad=(double)dx/(double)dy;
   if(y1<=y2)
   {
        for(int i=y1;i<=y2;i++){PlotScr(x1+(grad*(i-y1)),i,r,g,b);}
   }
   else
   {
        for(int i=y2;i<=y1;i++){PlotScr(x2+(grad*(i-y2)),i,r,g,b);}
   }
  }
 }
}
//---------------------------------------------------------------------------
//*****************************************************************************************
***********
bool CollideRect(vec3d pots[],double radius)
{
 bool col=false;
 double t,maxt,xx,yy,zz;
 vec3d pt1,pt2,dir;
 double xmini=(0.5*(maxx-boxdim.x))-radius;
 double xmaxi=(0.5*(maxx+boxdim.x))+radius;
 double ymini=-boxdim.y-radius;
 double ymaxi=radius;
 double zmaxi=maxz+radius;
 for(int i=0;i<NUMROI;i++)
 {
  pt1=pots[i]; pt2=pots[i+1];
```

```
    dir=Direction(StdVec(pt1,pt2));
    maxt=Length(StdVec(pt1,pt2));
    if(dir.x!=0.0)
    {
     t=-(pt1.x-xmini)/dir.x;
     yy=pt1.y+(dir.y*t);
     zz=pt1.z+(dir.z*t);
     if((yy>=ymini)&&(yy<=ymaxi)&&(zz<=zmaxi)&&(t>=0.0)&&(t<=maxt)){col=true;}
     t=-(pt1.x-xmaxi)/dir.x;
     yy=pt1.y+(dir.y*t);
     zz=pt1.z+(dir.z*t);
     if((yy>=ymini)&&(yy<=ymaxi)&&(zz<=zmaxi)&&(t>=0.0)&&(t<=maxt)){col=true;}
    }
    if(dir.y!=0.0)
    {
     t=-(pt1.y-ymini)/dir.y;
     xx=pt1.x+(dir.x*t);
     zz=pt1.z+(dir.z*t);
     if((xx>=xmini)&&(xx<=xmaxi)&&(zz<=zmaxi)&&(t>=0.0)&&(t<=maxt)){col=true;}
     t=-(pt1.y-ymaxi)/dir.y;
     xx=pt1.x+(dir.x*t);
     zz=pt1.z+(dir.z*t);
     if((xx>=xmini)&&(xx<=xmaxi)&&(zz<=zmaxi)&&(t>=0.0)&&(t<=maxt)){col=true;}
    }
    if(dir.z!=0.0)
    {
     t=-(pt1.z-zmaxi)/dir.z;
     xx=pt1.x+(dir.x*t);
     yy=pt1.y+(dir.y*t);

if((xx>=xmini)&&(xx<=xmaxi)&&(yy>=ymini)&&(yy<=ymaxi)&&(t>=0.0)&&(t<=maxt)){col=tru
e;}
    }
  }
  return col;
}
//-----------------------------------------------------------------------------------------
int Factorial(int n)
{
 if(n<=0)
 {
  return 1;
 }
 else
 {
  return n*Factorial(n-1);
 }
}
//-----------------------------------------------------------------------------------------
void InitDynamics(void)
{
 for(int i=0;i<NUMROI;i++)
 {
  for(int j=0;j<3;j++)
  {
   roipts[i][j]=roip[i][j];
  }
 }
```

```
vec3d path[NUMROI+1];

int ssp[NUMROI];
int npos=Factorial(NUMROI),nfound;
int *seqind=new int[NUMROI*npos];
double *seqdis=new double[npos];
int *routeind=new int[npos];
bool exists;

vec3d vtemp;

double deltat,mindist=100000.0,tdist;
vec3d tgp,cup,normals[NUMROI];
int index,index2,route=0,maxind=0,tempind;
double tims[NUMROI],totdist,tottime,fact=1.0/3.0;
vec3d tgps[NUMROI];

double maxyv=-1000.0,maxxv=-1000.0,maxzv=-1000.0;

for(int i=0;i<NUMROI;i++)
{
 if(roipts[i][0]==FRONT)
 {
  normals[i]=jvector;
  pts[i].x=(((double)roipts[i][1]+0.5)*fact*boxdim.x)+(0.5*(maxx-boxdim.x));
  pts[i].z=((double)roipts[i][2]+0.5)*fact*boxdim.z;
  pts[i].y=0.0;
 }
 if(roipts[i][0]==BACK )
 {
  normals[i]=Multiply(jvector,-1.0);
  pts[i].x=(((double)roipts[i][1]+0.5)*fact*boxdim.x)+(0.5*(maxx-boxdim.x));
  pts[i].z=((double)roipts[i][2]+0.5)*fact*boxdim.z;
  pts[i].y=-boxdim.y;
 }
 if(roipts[i][0]==LEFT )
 {
  normals[i]=ivector;
  pts[i].y=(((double)roipts[i][1]+0.5)*fact*boxdim.y)-boxdim.y;
  pts[i].z=((double)roipts[i][2]+0.5)*fact*boxdim.z;
  pts[i].x=boxdim.x+(0.5*(maxx-boxdim.x));
 }
 if(roipts[i][0]==RIGHT)
 {
  normals[i]=Multiply(ivector,-1.0);
  pts[i].y=(((double)roipts[i][1]+0.5)*fact*boxdim.y)-boxdim.y;
  pts[i].z=((double)roipts[i][2]+0.5)*fact*boxdim.z;
  pts[i].x=0.0+(0.5*(maxx-boxdim.x));
 }
 if(roipts[i][0]==TOP  )
 {
  normals[i]=kvector;
  pts[i].x=(((double)roipts[i][1]+0.5)*fact*boxdim.x)+(0.5*(maxx-boxdim.x));
  pts[i].y=(((double)roipts[i][2]+0.5)*fact*boxdim.y)-boxdim.y;
  pts[i].z=boxdim.z;
 }
}
```

```
for(int i=0;i<NUMROI;i++)
{
 tgt[i]=opt[i]=zerovector;
}

for(int i=0;i<NUMROI*npos;i++){seqind[i]=0;}
for(int i=0;i<NUMROI;i++)
{
 for(int j=0;j<(npos/Factorial(NUMROI-i-1));j++)
 {
  nfound=0;
  for(int k=0;k<NUMROI;k++)
  {
        exists=false;
        for(int m=0;m<i;m++)
        {
         if(seqind[(j*Factorial(NUMROI-i)*NUMROI)+m]==k){exists=true; break;}
        }
        if(!exists)
        {
         ssp[nfound]=k;
         nfound++;
        }
  }
  for(int k=0;k<(Factorial(NUMROI-i)/Factorial(NUMROI-i-1));k++)
  {
        for(int m=0;m<Factorial(NUMROI-i-1);m++)
        {
         seqind[((((j*Factorial(NUMROI-i))+(k*Factorial(NUMROI-i-
1)))+m)*NUMROI)+i]=ssp[k];
        }
  }
 }
}

char *ch;
for(int i=0;i<npos;i++)
{
 for(int j=0;j<NUMROI;j++)
 {
  ch=itoa(seqind[(i*NUMROI)+j]," ",10);
  TextOut(hDC,820+(j*30),20+(i*20),ch,strlen(ch));
 }
}


for(int i=0;i<npos;i++)
{
 totdist=0.0;
 for(int j=0;j<NUMROI;j++)
 {
  if(j==0)
  {
        index=seqind[(i*NUMROI)+j];
        totdist+=Length(StdVec(cinit,Sum(pts[index],Multiply(normals[index],d))));
        tims[j]=Length(StdVec(cinit,Sum(pts[index],Multiply(normals[index],d))));
  }
```

```
   else
   {
         index=seqind[(i*NUMROI)+j-1];
         index2=seqind[(i*NUMROI)+j];
         totdist+=Length(StdVec(Sum(pts[index],Multiply(normals[index],d)),Sum(pts[index2],Multi
ply(normals[index2],d))));
         tims[j]=Length(StdVec(Sum(pts[index],Multiply(normals[index],d)),Sum(pts[index2],Multip
ly(normals[index2],d))));
   }
  }
  seqdis[i]=totdist;
  if(totdist<mindist)
  {
   mindist=totdist;
   route=i;
   vset=totdist/totaltime;
   for(int j=0;j<NUMROI;j++)
   {
         times[j]=tims[j]/vset;
         ctims[j]=0.0;
   }
   ctims[0]=times[0];
   for(int j=1;j<NUMROI;j++){ctims[j]=ctims[j-1]+times[j];}
   for(int j=0;j<NUMROI;j++)
   {
         index=seqind[(route*NUMROI)+j];
         tgt[j]=Sum(Sum(pts[index],Multiply(normals[index],d)),Multiply(vpart,ctims[j]));
         opt[j]=pts[index];
         spt[j]=Sum(pts[index],Multiply(normals[index],d));
   }
  }
 }
 for(int i=0;i<npos;i++){routeind[i]=i;}
 for(int i=npos-2;i>=0;i--)
 {
  for(int j=0;j<i;j++)
  {
   if(seqdis[j]>seqdis[j+1])
   {
         tempind=routeind[j];
         routeind[j]=routeind[j+1];
         routeind[j+1]=tempind;
   }
  }
 }

 path[0]=cinit;
 for(int i=0;i<npos;i++)
 {
  for(int j=1;j<=NUMROI;j++)
  {
   index=seqind[(i*NUMROI)+j-1];
   path[j]=Sum(pts[index],Multiply(normals[index],d));
  }
  if(CollideRect(path,radiusc)){ch="C"; TextOut(hDC,995,20+(i*20),ch,strlen(ch));}
 }

 ch="*";
 TextOut(hDC,970,20+(route*20),ch,strlen(ch));
}
//-----------------------------------------------------------------------------------------
void DrawBox(vec3d blist[])
{
 vec3d llist[24];
 vec3d pt1,pt2;
 int px1,py1,px2,py2;
 llist[ 0]=blist[0]; llist[ 1]=blist[1];
 llist[ 2]=blist[1]; llist[ 3]=blist[2];
 llist[ 4]=blist[2]; llist[ 5]=blist[3];
 llist[ 6]=blist[3]; llist[ 7]=blist[0];
 llist[ 8]=blist[4]; llist[ 9]=blist[5];
 llist[10]=blist[5]; llist[11]=blist[6];
 llist[12]=blist[6]; llist[13]=blist[7];
 llist[14]=blist[7]; llist[15]=blist[4];
 llist[16]=blist[0]; llist[17]=blist[4];
 llist[18]=blist[1]; llist[19]=blist[5];
 llist[20]=blist[2]; llist[21]=blist[6];
 llist[22]=blist[3]; llist[23]=blist[7];
 for(int i=0;i<12;i++)
 {
  pt1=llist[(i*2)+0];
  pt2=llist[(i*2)+1];
  pt1=Rotate(pt1,zerovector);
  pt2=Rotate(pt2,zerovector);
  px1=400+int(pt1.x); py1=300+int(pt1.y);
  px2=400+int(pt2.x); py2=300+int(pt2.y);
  DrawLineE(px1,py1,px2,py2,255,255,255);
 }
}
//-----------------------------------------------------------------------------------------
void Graphics(void)
{
 complex diff;
 vec3db white={255,255,255};
 vec3d blist[8];
 ca=cos(rot.x*rad);
 sa=sin(rot.x*rad);
 cb=cos(rot.y*rad);
 sb=sin(rot.y*rad);
 cc=cos(rot.z*rad);
 sc=sin(rot.z*rad);
 double sf=400.0,timenow,tht=0.0,phi=0.0;
 vec3d campt,velo,target,camroto={0.0,0.0,0.0};
 int stage=0;
 int px1,py1,px2,py2;
 vec3d pt1,pt2;

 for(int i=0;i<800*600;i++){scd[i]=100000.0;}
 for(int i=0;i<800*600*3;i++){scr[i]=0;}


 for(int i=0;i<NUMROI;i++)
 {
  if(i==0)
  {
   pt1=Multiply(cinit,sf);
   pt2=Multiply(tgt[i],sf);
  }
  else
  {
   pt1=Multiply(tgt[i-1],sf);
   pt2=Multiply(tgt[i+0],sf);
  }
  pt1=Rotate(pt1,zerovector);
  pt2=Rotate(pt2,zerovector);
  px1=400+int(pt1.x); py1=300+int(pt1.y);
  px2=400+int(pt2.x); py2=300+int(pt2.y);
  DrawLineE(px1,py1,px2,py2,255,0,255);
 }
 blist[0]=zerovector;
 blist[1]=Multiply(ivector,sf*maxx);
 blist[2]=Sum(Multiply(ivector,sf*maxx),Multiply(jvector,sf*maxy));
 blist[3]=Multiply(jvector,sf*maxy);
 blist[4]=Multiply(kvector,sf*maxz);
 blist[5]=Sum(Multiply(ivector,sf*maxx),Multiply(kvector,sf*maxz));

blist[6]=Sum(Sum(Multiply(ivector,sf*maxx),Multiply(jvector,sf*maxy)),Multiply(kvector,sf*maxz))
;
 blist[7]=Sum(Multiply(jvector,sf*maxy),Multiply(kvector,sf*maxz));
 DrawBox(blist);
 blist[0]=zerovector;
 blist[1]=Multiply(ivector,sf*boxdim.x);
 blist[2]=Sum(Multiply(ivector,sf*boxdim.x),Multiply(jvector,sf*boxdim.y));
 blist[3]=Multiply(jvector,sf*boxdim.y);
 blist[4]=Multiply(kvector,sf*boxdim.z);
 blist[5]=Sum(Multiply(ivector,sf*boxdim.x),Multiply(kvector,sf*boxdim.z));

blist[6]=Sum(Sum(Multiply(ivector,sf*boxdim.x),Multiply(jvector,sf*boxdim.y)),Multiply(kvector,sf
*boxdim.z));
 blist[7]=Sum(Multiply(jvector,sf*boxdim.y),Multiply(kvector,sf*boxdim.z));
 for(int i=0;i<8;i++){blist[i].x+=(maxx-boxdim.x)*0.5*sf; blist[i].y-=boxdim.y*sf;}
 for(int i=0;i<8;i++){blist[i]=Sum(blist[i],Multiply(vpart,frames*speedy*sf));}
 DrawBox(blist);
 blist[0]=zerovector;
 blist[1]=Multiply(ivector,sf*camx);
 blist[2]=Sum(Multiply(ivector,sf*camx),Multiply(jvector,sf*camy));
 blist[3]=Multiply(jvector,sf*camy);
 blist[4]=Multiply(kvector,sf*camz);
 blist[5]=Sum(Multiply(ivector,sf*camx),Multiply(kvector,sf*camz));

blist[6]=Sum(Sum(Multiply(ivector,sf*camx),Multiply(jvector,sf*camy)),Multiply(kvector,sf*camz))
;
 blist[7]=Sum(Multiply(jvector,sf*camy),Multiply(kvector,sf*camz));
 for(int i=0;i<8;i++){blist[i].x-=sf*camx*0.5; blist[i].y-=sf*camy*0.5; blist[i].z-=sf*camz*0.5;}

 timenow=frames*speedy;
 for(int i=1;i<NUMROI;i++)
 {
        if((timenow>ctims[i-1])&&(timenow<ctims[i])){stage=i;}
 }
 if(stage==0){pt1=cinit; pt2=spt[0];}else{pt1=spt[stage-1]; pt2=spt[stage];}
 target=Sum(opt[stage],Multiply(vpart,timenow));
 campt=Sum(Multiply(vpart,timenow),pt1);
 if(stage>0){timenow-=ctims[stage-1];}
 velo=Multiply(StdVec(pt1,pt2),1.0/times[stage]);
 campt=Sum(campt,Multiply(velo,timenow));
 diff.x=target.x-campt.x;
 diff.y=target.y-campt.y;
 tht=Arg(diff);
 diff.x=Mod(Complex(target.x-campt.x,target.y-campt.y));
 diff.y=target.z-campt.z;
 phi=Arg(diff);
 campt=Multiply(campt,sf);
 camroto.z=(tht+0.5*PI)*deg;
 camroto.x=phi*deg;
 for(int i=0;i<8;i++){blist[i]=Rotate3D(blist[i],zerovector,camroto);}

 if(frames<(100*ctims[NUMROI-1])){for(int i=0;i<8;i++){blist[i]=Sum(blist[i],campt);}}
 DrawBox(blist);

 for(int i=0;i<NUMROI;i++)
  {
   pt1=Multiply(Sum(opt[i],Multiply(vpart,frames*speedy)),sf);
   pt2=Multiply(opt[i],sf);
   pt1=Rotate(pt1,zerovector);
   pt2=Rotate(pt2,zerovector);
   px1=400+int(pt1.x); py1=300+int(pt1.y);
   px2=400+int(pt2.x); py2=300+int(pt2.y);
   DrawLineE(px1,py1,px2,py2,0,255,0);
  }
}
//-------------------------------------------------------------------------------------
//*************************************************************************************
***********
void Initialize(void)
{
 InitDynamics();
}
//-------------------------------------------------------------------------------------
void Draw(void)
{
 BITMAPINFO bi={40,800,600,1,24,0,0,0,0,0,0};
 SendMessageA(hWnd,WM_PAINT,0,0);
 Graphics();
 frames++;
 if(frames>=(maxy/(speedy*vpart.y))){frames=0;}
 StretchDIBits(hDC,2,2,800,600,0,0,800,600,&scr,&bi,DIB_RGB_COLORS,SRCCOPY);
}
//-------------------------------------------------------------------------------------
bool CreateWindowXX(char* title,int width,int height,int bits)
{
 HMENU hMenu,hSubMenu;

 UINT PixelFormat;
 WNDCLASS wc;
 DWORD dwStyle;

 hInstance=GetModuleHandle(NULL);
 wc.style=0;
 wc.lpfnWndProc=(WNDPROC)WndProc;
 wc.cbClsExtra=0;
 wc.cbWndExtra=0;
```

```
  wc.hInstance=hInstance;
  wc.hIcon=LoadIcon(NULL,IDI_WINLOGO);
  wc.hCursor=LoadCursor(NULL,IDC_ARROW);
  wc.hbrBackground=(HBRUSH)(COLOR_WINDOW+1);
  wc.lpszMenuName=NULL;
  wc.lpszClassName="Seffat";
  if(!RegisterClass(&wc))
  {
   MessageBox(NULL,"Failed To Register The Window
Class.","ERROR",MB_OK|MB_ICONEXCLAMATION);
   return FALSE;
  }

if(!(hWnd=CreateWindow("Seffat",title,WS_OVERLAPPEDWINDOW,0,0,width,height,NULL,NU
LL,hInstance,NULL)))
  {
   MessageBox(NULL,"Window Creation Error.","ERROR",MB_OK|MB_ICONEXCLAMATION);
   return FALSE;
  }
  static PIXELFORMATDESCRIPTOR pfd=
  {

sizeof(PIXELFORMATDESCRIPTOR),1,PFD_DRAW_TO_WINDOW|PFD_DOUBLEBUFFER,PF
D_TYPE_RGBA,bits,0,0,0,0,0,0,
   0,0,0,0,0,0,32,0,0,PFD_MAIN_PLANE,0,0,0,0
  };
  if(!(hDC=GetDC(hWnd)))
  {
   MessageBox(NULL,"Can't create a device
context.","ERROR",MB_OK|MB_ICONEXCLAMATION);
   return FALSE;
  }
  if(!(PixelFormat=ChoosePixelFormat(hDC,&pfd)))
  {
   MessageBox(NULL,"Can't find a suitable
pixelformat.","ERROR",MB_OK|MB_ICONEXCLAMATION);
   return FALSE;
  }
  if(!SetPixelFormat(hDC,PixelFormat,&pfd))
  {
   MessageBox(NULL,"Can't set the pixelformat.","ERROR",MB_OK|MB_ICONEXCLAMATION);
   return FALSE;
  }
  if(!DescribePixelFormat(hDC,PixelFormat,sizeof(PIXELFORMATDESCRIPTOR),&pfd))
  {
   MessageBox(NULL,"Can't Describe
PixelFormat","ERROR",MB_OK|MB_ICONEXCLAMATION);
   return FALSE;
  }

  ShowWindow(hWnd,SW_SHOW);
  UpdateWindow(hWnd);
  SetForegroundWindow(hWnd);
  SetFocus(hWnd);
  Initialize();
  return TRUE;
}
//-------------------------------------------------------------------------------------------
```

```
LRESULT CALLBACK WndProc(HWND hWnd,UINT uMsg,WPARAM wParam,LPARAM
lParam)
{
 switch (uMsg)
 {
  case WM_CLOSE:
  {
   PostQuitMessage(0);
   return 0;
  }
  case WM_KEYDOWN:
  {
   if(wParam==VK_ESCAPE)
   {
       PostQuitMessage(0);
       return 0;
   }
   if(wParam=='A')
   {
       rot.x++;
   }
   if(wParam=='S')
   {
       rot.x--;
   }
   if(wParam=='D')
   {
       rot.y++;
   }
   if(wParam=='F')
   {
       rot.y--;
   }
   if(wParam=='G')
   {
       rot.z++;
   }
   if(wParam=='H')
   {
       rot.z--;
   }
  }
 }
 return DefWindowProc(hWnd,uMsg,wParam,lParam);
}
//-------------------------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int
nCmdShow)
{
 MSG msg;
 bool done=false,ret=false;
 ret=CreateWindowXX("Radiosity by S.M.C.",1024,737,32);
 if(ret==false){done=true;}
 while(!done)
 {
  if(PeekMessage(&msg,NULL,0,0,PM_REMOVE))
  {
   if(msg.message==WM_QUIT)
```

```
    }
        done=true;
    }
    else
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
   }
   else
   {
    Draw();
    SwapBuffers(hDC);
   }
  }
 return(msg.wParam);
}
```

9.4.2 Image Processing Utility Source Code

This program performs takes image files and performs the described image processing operations as required. The input is the filenames of the image files and the operations that need to be performed. Currently, background subtraction, color verification, dimensional verification and assembly verifications are supported. In fact, dimensional verification and surface finish verifications are implicit in the process and need not be specified. Special ROI in the images may be specified and thresholds for rejection may be modified if required. The output is a binary result indicating whether the images have passed all the tests or not. In the case of background subtraction, the output is the foreground image with the background regions set to black. In the case of color verification the channel(s) that need to be verified must be specified as well. The standalone application requires Microsoft Windows XP with OpenCV Version 1.0 or higher to be installed. The OpenCV library is freely available from [10]. The compilation was performed with Dev C++ [version 4.9.9.2], a free C/C++ software development environment.

```
//-------------------------------------------------------------------------------
#include "cv.h"
#include "highgui.h"
//-------------------------------------------------------------------------------
IplImage *imag1=0,*imag2=0;
IplImage *smth1=0,*smth2=0;
IplImage *hist1=0,*hist2=0;
IplImage *logp1=0,*logp2=0;
IplImage *gray1=0,*gray2=0;
IplImage *ilpi1=0,*ilpi2=0;
IplImage *idiff=0,*ibind=0;
//-------------------------------------------------------------------------------
char *name1="colortest.jpg";
char *name2="colorgolden.jpg";
CvSize size={640,480};
//-------------------------------------------------------------------------------
int Coord(int ii,int jj,int sz)
{
 return ((jj-1)*sz)+ii-1;
}
//-------------------------------------------------------------------------------
void CleanBinaryImage(IplImage *imagi,CvSize sz,int thresh)
{
 bool one;
```

```
int sta,end,crd;
int xs=sz.width,ys=sz.height;

for(int i=1;i<=xs;i++)
{
 one=false;
 sta=0;end=0;
 if(imagi->imageData[Coord(i,1,xs)]>0)
 {
  one=true;
  sta=1;
 }
 for(int j=1;j<=ys;j++)
 {
  crd=Coord(i,j,xs);
  if(imagi->imageData[crd]==0)
  {
   if(one)
   {
    one=false;
    end=j;
    if((end-sta)<thresh)
    {
     for(int k=sta;k<end;k++)
     {
      imagi->imageData[Coord(i,k,xs)]=0;
     }
    }
   }
  }
  else
  {
   if(!one)
   {
    sta=j;
    one=true;
   }
  }
 }
}
for(int j=1;j<=ys;j++)
{
 one=false;
 sta=0;end=0;
 if(imagi->imageData[Coord(1,j,xs)]>0)
 {
  one=true;
  sta=1;
 }
 for(int i=1;i<=xs;i++)
 {
  crd=Coord(i,j,xs);
  if(imagi->imageData[crd]==0)
  {
   if(one)
   {
    one=false;
    end=i;
```

```
        if((end-sta)<thresh)
        {
         for(int k=sta;k<end;k++)
         {
          imagi->imageData[Coord(k,j,xs)]=0;
         }
        }
       }
      }
      else
      {
       if(!one)
       {
        sta=i;
        one=true;
       }
      }
     }
   }
}
//------------------------------------------------------------------------------
void AbsDifferenceImage(IplImage *i1,IplImage *i2,IplImage *op,CvSize sz,int flag)
{
 int c;
 int nx=sz.width;
 int ny=sz.height;
 for(int i=1;i<=nx;i++)
 {
  for(int j=1;j<=ny;j++)
  {
   c=Coord(i,j,nx);
   if(flag==1)
   {
    op->imageData[(c*3)+0]=abs(i1->imageData[(c*3)+0]-i2->imageData[(c*3)+0]);
    op->imageData[(c*3)+1]=abs(i1->imageData[(c*3)+1]-i2->imageData[(c*3)+1]);
    op->imageData[(c*3)+2]=abs(i1->imageData[(c*3)+2]-i2->imageData[(c*3)+2]);
   }
   if(flag==2)
   {
    op->imageData[c]=abs(i1->imageData[c]-i2->imageData[c]);
   }
  }
 }
}
//------------------------------------------------------------------------------
void Grayscale(IplImage *input,IplImage *output,CvSize sz)
{
 int c;
 int nx=sz.width;
 int ny=sz.height;
 for(int i=1;i<=nx;i++)
 {
  for(int j=1;j<=ny;j++)
  {
   c=Coord(i,j,nx);
   output->imageData[c]=byte(sqrt(
   pow(double(input->imageData[(c*3)+0]),2.0)+
   pow(double(input->imageData[(c*3)+1]),2.0)+
    pow(double(input->imageData[(c*3)+2]),2.0)
    )/sqrt(3.0));
   }
  }
}
//-----------------------------------------------------------------------------
void Convolve2D(IplImage *imagi,IplImage *imago,CvSize sz,double k)
{
 int ip,jp,im,jm,c1,c2,c3,c4,c5,c6,c7,c8,c9,rc,gc,bc;
 int xsz=sz.width,ysz=sz.height;
 for(int i=1;i<=xsz;i++)
 {
  for(int j=1;j<=ysz;j++)
  {
   im=i-1; if(im==0){im=1;}
   ip=i+1; if(ip==xsz+1){ip=xsz;}
   jm=j-1; if(jm==0){jm=1;}
   jp=j+1; if(jp==ysz+1){jp=ysz;}
   c1=Coord(im,jm,xsz);
   c2=Coord(im,j,xsz);
   c3=Coord(im,jp,xsz);
   c4=Coord(i,jm,xsz);
   c5=Coord(i,j,xsz);
   c6=Coord(i,jp,xsz);
   c7=Coord(ip,jm,xsz);
   c8=Coord(ip,j,xsz);
   c9=Coord(ip,jp,xsz);
   rc=(int)(-k*(
   imagi->imageData[(c1*3)+2]+imagi->imageData[(c2*3)+2]+imagi->imageData[(c3*3)+2]+
   imagi->imageData[(c4*3)+2]+imagi->imageData[(c6*3)+2]+
   imagi->imageData[(c7*3)+2]+imagi->imageData[(c8*3)+2]+imagi->imageData[(c9*3)+2]
   )+((1+(8*k))*imagi->imageData[(c5*3)+2]));
   gc=(int)(-k*(
   imagi->imageData[(c1*3)+1]+imagi->imageData[(c2*3)+1]+imagi->imageData[(c3*3)+1]+
   imagi->imageData[(c4*3)+1]+imagi->imageData[(c6*3)+1]+
   imagi->imageData[(c7*3)+1]+imagi->imageData[(c8*3)+1]+imagi->imageData[(c9*3)+1]
   )+((1+(8*k))*imagi->imageData[(c5*3)+1]));
   bc=(int)(-k*(
   imagi->imageData[(c1*3)+0]+imagi->imageData[(c2*3)+0]+imagi->imageData[(c3*3)+0]+
   imagi->imageData[(c4*3)+0]+imagi->imageData[(c6*3)+0]+
   imagi->imageData[(c7*3)+0]+imagi->imageData[(c8*3)+0]+imagi->imageData[(c9*3)+0]
   )+((1+(8*k))*imagi->imageData[(c5*3)+0]));
   if(rc<0){rc=0;} if(rc>255){rc=255;}
   if(gc<0){gc=0;} if(gc>255){gc=255;}
   if(bc<0){bc=0;} if(bc>255){bc=255;}
   imago->imageData[(c5*3)+2]=rc;
   imago->imageData[(c5*3)+1]=gc;
   imago->imageData[(c5*3)+0]=bc;
  }
 }
}
//-----------------------------------------------------------------------------
void FourierCorrelation(IplImage *im1,IplImage *im2,IplImage *cres)
{
 IplImage *bufr,*covt;
 IplImage *rel1,*img1,*cmp1,*rel2,*img2,*cmp2;
 IplImage *crel,*cimg;
 int dftm,dftn,mi,mj,c,c2,ii,jj;
```

```
unsigned int maxi=0;
CvMat *dft1,*dft2,*dftc,*filt,tmp;
double m,M,radius;
rel1=cvCreateImage(cvGetSize(im1),IPL_DEPTH_64F,1);
img1=cvCreateImage(cvGetSize(im1),IPL_DEPTH_64F,1);
cmp1=cvCreateImage(cvGetSize(im1),IPL_DEPTH_64F,2);
rel2=cvCreateImage(cvGetSize(im2),IPL_DEPTH_64F,1);
img2=cvCreateImage(cvGetSize(im2),IPL_DEPTH_64F,1);
cmp2=cvCreateImage(cvGetSize(im2),IPL_DEPTH_64F,2);
bufr=cvCreateImage(cvGetSize(im2),IPL_DEPTH_64F,1);
for(int i=1;i<=im2->width;i++)
{
 for(int j=1;j<=im2->height;j++)
 {
  c=Coord(i,j,im2->width);
  c2=Coord(im2->width-i+1,im2->height-j+1,im2->width);
  bufr->imageData[c]=im2->imageData[c2];
 }
}
for(int i=0;i<((im2->width)*(im2->height));i++){im2->imageData[i]=bufr->imageData[i];}
cvScale(im1,rel1,1.0,0.0);
cvScale(im2,rel2,1.0,0.0);
cvZero(img1);
cvZero(img2);
cvMerge(rel1,img1,NULL,NULL,cmp1);
cvMerge(rel2,img2,NULL,NULL,cmp2);

dftm=cvGetOptimalDFTSize(im1->height-1);
dftn=cvGetOptimalDFTSize(im1->width-1);
crel=cvCreateImage(cvSize(dftn,dftm),IPL_DEPTH_64F,1);
cimg=cvCreateImage(cvSize(dftn,dftm),IPL_DEPTH_64F,1);
covt=cvCreateImage(cvSize(dftn,dftm),IPL_DEPTH_8U,1);
dft1=cvCreateMat(dftm,dftn,CV_64FC2);
dft2=cvCreateMat(dftm,dftn,CV_64FC2);
dftc=cvCreateMat(dftm,dftn,CV_64FC2);
filt=cvCreateMat(dftm,dftn,CV_64FC2);
for(int i=1;i<=dftn;i++)
{
 for(int j=1;j<=dftm;j++)
 {
  ii=i-1; jj=j-1;
  if(i>((dftn+1)/2)){ii=dftn-i;}
  if(j>((dftm+1)/2)){jj=dftm-j;}
  radius=sqrt((ii*ii)+(jj*jj));
  c=Coord(i,j,dftn);
  if(radius>100.0)
  {
   filt->data.db[(c*2)+0]=0.0;
   filt->data.db[(c*2)+1]=0.0;
  }
  else
  {
   filt->data.db[(c*2)+0]=1.0;
   filt->data.db[(c*2)+1]=1.0;
  }
 }
}
cvGetSubRect(dft1,&tmp,cvRect(0,0,im1->width,im1->height));
```

```
cvCopy(cmp1,&tmp,NULL);
if(dft1->cols>im1->width)
{
 cvGetSubRect(dft1,&tmp,cvRect(im1->width,0,dft1->cols-im1->width,im1->height));
 cvZero(&tmp);
}

cvGetSubRect(dft2,&tmp,cvRect(0,0,im2->width,im2->height));
cvCopy(cmp2,&tmp,NULL);
if(dft2->cols>im2->width)
{
 cvGetSubRect(dft2,&tmp,cvRect(im2->width,0,dft2->cols-im2->width,im2->height));
 cvZero(&tmp);
}
cvDFT(dft1,dft1,CV_DXT_FORWARD,cmp1->height);
cvDFT(dft2,dft2,CV_DXT_FORWARD,cmp2->height);
cvMulSpectrums(dft1,dft2,dftc,0);
cvMulSpectrums(dft1,filt,dft1,0);
cvMulSpectrums(dft2,filt,dft2,0);
cvDFT(dftc,dftc,CV_DXT_INVERSE,cmp1->height);
cvDFT(dft1,dft1,CV_DXT_INVERSE,cmp1->height);
cvDFT(dft2,dft2,CV_DXT_INVERSE,cmp1->height);
cvSplit(dftc,crel,cimg,0,0);
cvSplit(dft1,rel1,img1,0,0);
cvSplit(dft2,rel2,img2,0,0);
cvPow(crel,crel,2.0);
cvPow(cimg,cimg,2.0);
cvAdd(crel,cimg,crel,NULL);
cvPow(crel,crel,0.5);
cvMinMaxLoc(crel,&m,&M,NULL,NULL,NULL);
cvScale(crel,crel,256.0/(M-m),256.0*(-m)/(M-m));
cvMinMaxLoc(rel1,&m,&M,NULL,NULL,NULL);
cvScale(rel1,rel1,256.0/(M-m),256.0*(-m)/(M-m));
cvMinMaxLoc(rel2,&m,&M,NULL,NULL,NULL);
cvScale(rel2,rel2,256.0/(M-m),256.0*(-m)/(M-m));
cvConvertScaleAbs(crel,covt,1,0);
//cvConvertScaleAbs(rel1,im1,1,0);
//cvConvertScaleAbs(rel2,im2,1,0);
for(int i=1;i<=dftn;i++)
{
 for(int j=1;j<=dftm;j++)
 {
  c=Coord(i,j,dftn);
  if((covt->imageData[c])>=maxi)
  {
   maxi=covt->imageData[c];
   mi=i;
   mj=j;
  }
 }
}
covt->imageData[Coord(mi,mj,dftn)]=0;
for(int i=1;i<=im2->width;i++)
{
 for(int j=1;j<=im2->height;j++)
 {
  c=Coord(i,j,im2->width);
  ii=im2->width-i+mi+1;
```

```
    jj=im2->height-j+mj+1;
    if(ii>im2->width ){ii-=im2->width; } if(ii<0){ii+=im2->width; }
    if(jj>im2->height){jj-=im2->height;} if(jj<0){jj+=im2->height;}
    c2=Coord(ii,jj,im2->width);
    cres->imageData[c]=im2->imageData[c2];
   }
  }
 for(int i=1;i<=im2->width;i++)
  {
   for(int j=1;j<=im2->height;j++)
   {
    c=Coord(i,j,im2->width);
    c2=Coord(im2->width-i+1,im2->height-j+1,im2->width);
    bufr->imageData[c]=im2->imageData[c2];
   }
  }
 for(int i=0;i<((im2->width)*(im2->height));i++){im2->imageData[i]=bufr->imageData[i];}
 cvNamedWindow("Fourier",CV_WINDOW_AUTOSIZE);
 cvShowImage("Fourier",covt);
 cvNamedWindow("Shifted",CV_WINDOW_AUTOSIZE);
 cvShowImage("Shifted",cres);
 cvSaveImage("fouriermap.jpg",covt);
 cvSaveImage("shifted.jpg",cres);
}
//---------------------------------------------------------------------------
void DoContours(IplImage* img)
{
 double area1=0.0;
 IplImage* imgc=0;
 CvSeq* contours=0;
 imgc=cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,3);
 CvMemStorage* storage=cvCreateMemStorage(0);
 cvFindContours(img,storage,&contours,sizeof(CvContour),
 CV_RETR_TREE,CV_CHAIN_APPROX_SIMPLE,cvPoint(0,0));
 contours=cvApproxPoly(contours,sizeof(CvContour),storage,CV_POLY_APPROX_DP,3,1);
 cvDrawContours(imgc,contours,CV_RGB(255,0,0),CV_RGB(0,255,0),
 1,3,CV_AA,cvPoint(0,0));
 area1=cvContourArea(contours,CV_WHOLE_SEQ);
 cvNamedWindow("Contours",CV_WINDOW_AUTOSIZE);
 cvShowImage("Contours",imgc);
 cvSaveImage("contour.jpg",imgc);
}
//---------------------------------------------------------------------------
bool CalcBinDiffMetr(IplImage* bim,double threshold)
{
 double tval=0.0,metr;
 bool res=true;
 CvSize sz=cvGetSize(bim);
 for(int i=0;i<sz.width*sz.height;i++)
  {
   if(bim->imageData[i]!=0){tval++;}
  }
 metr=tval/double(sz.width*sz.height);
 if(metr>threshold){res=false;}
 return res;
}
//---------------------------------------------------------------------------
bool ColorTest(int channel)
```

```
{
 double corr=0.0;
 bool result=true;
 IplImage *shftd=0;
 CvPoint2D32f cntr={size.width/2,size.height/2};
 imag1=cvLoadImage(name1);
 imag2=cvLoadImage(name2);
 smth1=cvCreateImage(size,IPL_DEPTH_8U,3);
 smth2=cvCreateImage(size,IPL_DEPTH_8U,3);
 gray1=cvCreateImage(size,IPL_DEPTH_8U,1);
 gray2=cvCreateImage(size,IPL_DEPTH_8U,1);
 hist1=cvCreateImage(size,IPL_DEPTH_8U,1);
 hist2=cvCreateImage(size,IPL_DEPTH_8U,1);
 idiff=cvCreateImage(size,IPL_DEPTH_8U,1);
 ibind=cvCreateImage(size,IPL_DEPTH_8U,1);
 shftd=cvCreateImage(size,IPL_DEPTH_8U,1);
 cvSmooth(imag1,smth1,CV_GAUSSIAN,5,0,0,0);
 cvSmooth(imag2,smth2,CV_GAUSSIAN,5,0,0,0);
 for(int i=0;i<size.width*size.height;i++)
 {
  gray1->imageData[i]=smth1->imageData[(i*3)+channel];
  gray2->imageData[i]=smth2->imageData[(i*3)+channel];
 }
 cvEqualizeHist(gray1,hist1);
 cvEqualizeHist(gray2,hist2);
 FourierCorrelation(gray1,gray2,shftd);
 AbsDifferenceImage(gray1,shftd,idiff,size,2);
 for(int i=0;i<size.width*size.height;i++)
 {
  if(idiff->imageData[i]>40){ibind->imageData[i]=255;}else{ibind->imageData[i]=0;}
 }
 CleanBinaryImage(ibind,size,2);
 cvNamedWindow("Raw 1",CV_WINDOW_AUTOSIZE);
 cvNamedWindow("Raw 2",CV_WINDOW_AUTOSIZE);
 cvNamedWindow("Difference",CV_WINDOW_AUTOSIZE);
 cvNamedWindow("Cleaned Difference",CV_WINDOW_AUTOSIZE);
 cvNamedWindow("Histogram Equalized 1",CV_WINDOW_AUTOSIZE);
 cvNamedWindow("Histogram Equalized 2",CV_WINDOW_AUTOSIZE);
 cvShowImage("Raw 1",gray1);
 cvShowImage("Raw 2",gray2);
 cvShowImage("Difference",idiff);
 cvShowImage("Cleaned Difference",ibind);
 cvShowImage("Histogram Equalized 1",hist1);
 cvShowImage("Histogram Equalized 2",hist2);
 cvSaveImage("bdiff.jpg",ibind);
 cvSaveImage("cdiff.jpg",idiff);
 DoContours(ibind);
 result=CalcBinDiffMetr(ibind,0.2);
 return result;
}
//------------------------------------------------------------------------------
bool AssemblyTest(void)
{
 double corr=0.0;
 bool result=true;
 IplImage *shftd=0;
 CvPoint2D32f cntr={size.width/2,size.height/2};
 imag1=cvLoadImage(name1);
 imag2=cvLoadImage(name2);
 smth1=cvCreateImage(size,IPL_DEPTH_8U,3);
 smth2=cvCreateImage(size,IPL_DEPTH_8U,3);
 gray1=cvCreateImage(size,IPL_DEPTH_8U,1);
 gray2=cvCreateImage(size,IPL_DEPTH_8U,1);
 hist1=cvCreateImage(size,IPL_DEPTH_8U,1);
 hist2=cvCreateImage(size,IPL_DEPTH_8U,1);
 idiff=cvCreateImage(size,IPL_DEPTH_8U,1);
 ibind=cvCreateImage(size,IPL_DEPTH_8U,1);
 shftd=cvCreateImage(size,IPL_DEPTH_8U,1);
 cvSmooth(imag1,smth1,CV_GAUSSIAN,5,0,0,0);
 cvSmooth(imag2,smth2,CV_GAUSSIAN,5,0,0,0);
 Grayscale(smth1,gray1,size);
 Grayscale(smth2,gray2,size);
 cvEqualizeHist(gray1,hist1);
 cvEqualizeHist(gray2,hist2);
 FourierCorrelation(gray1,gray2,shftd);
 AbsDifferenceImage(gray1,shftd,idiff,size,2);
 for(int i=0;i<size.width*size.height;i++)
 {
  if(idiff->imageData[i]>40){ibind->imageData[i]=255;}else{ibind->imageData[i]=0;}
 }
 CleanBinaryImage(ibind,size,5);
 cvNamedWindow("Raw 1",CV_WINDOW_AUTOSIZE);
 cvNamedWindow("Raw 2",CV_WINDOW_AUTOSIZE);
 cvNamedWindow("Difference",CV_WINDOW_AUTOSIZE);
 cvNamedWindow("Cleaned Difference",CV_WINDOW_AUTOSIZE);
 cvNamedWindow("Histogram Equalized 1",CV_WINDOW_AUTOSIZE);
 cvNamedWindow("Histogram Equalized 2",CV_WINDOW_AUTOSIZE);
 cvShowImage("Raw 1",gray1);
 cvShowImage("Raw 2",gray2);
 cvShowImage("Difference",idiff);
 cvShowImage("Cleaned Difference",ibind);
 cvShowImage("Histogram Equalized 1",hist1);
 cvShowImage("Histogram Equalized 2",hist2);
 cvSaveImage("bdiff.jpg",ibind);
 cvSaveImage("cdiff.jpg",idiff);
 DoContours(ibind);
 result=CalcBinDiffMetr(ibind,0.25);
 return result;
}
//-------------------------------------------------------------------------------
void SubtractBackground(char* fgndn,char* bgndn,int thresh1,int thresh2,IplImage* result)
{
 int c,rr,gg,bb;
 double magn;
 IplImage *fgnd=0,*bgnd=0;
 fgnd=cvLoadImage(fgndn);
 bgnd=cvLoadImage(bgndn);
 CvSize sz=cvGetSize(fgnd);
 IplImage* difr=cvCreateImage(sz,IPL_DEPTH_8U,3);
 IplImage* binr=cvCreateImage(sz,IPL_DEPTH_8U,1);
 result=cvCreateImage(sz,IPL_DEPTH_8U,3);
 AbsDifferenceImage(fgnd,bgnd,difr,sz,1);
 for(int i=1;i<=sz.width;i++)
 {
  for(int j=1;j<=sz.height;j++)
  {
    c=Coord(i,j,sz.width);
    rr=difr->imageData[(c*3)+2];
    gg=difr->imageData[(c*3)+1];
    bb=difr->imageData[(c*3)+0];
    magn=sqrt((rr*rr)+(gg*gg)+(bb*bb))/sqrt(3.0);
    if(magn>thresh1)
    {
     binr->imageData[c]=255;
     result->imageData[(c*3)+0]=fgnd->imageData[(c*3)+0];
     result->imageData[(c*3)+1]=fgnd->imageData[(c*3)+1];
     result->imageData[(c*3)+2]=fgnd->imageData[(c*3)+2];
    }
    else
    {
     binr->imageData[c]=0;
    }
   }
  }
 CleanBinaryImage(binr,sz,thresh2);
 for(int i=1;i<=sz.width;i++)
 {
  for(int j=1;j<=sz.height;j++)
  {
   c=Coord(i,j,sz.width);
   if(binr->imageData[c]==0)
   {
    result->imageData[(c*3)+0]=0;
    result->imageData[(c*3)+1]=0;
    result->imageData[(c*3)+2]=0;
   }
  }
 }
 DoContours(binr);
 cvNamedWindow("Background Subtraction",CV_WINDOW_AUTOSIZE);
 cvShowImage("Background Subtraction",result);
 cvSaveImage("backsub.jpg",result);
 cvSaveImage("AbsDiff.jpg",difr);
}
//----------------------------------------------------------------------------
int main(void)
{
 bool r1=true,r2=true;
 IplImage* rubbish=0;
 SubtractBackground("222.jpg","111.jpg",40,3,rubbish);
 r1=AssemblyTest();
 r2=ColorTest(2);
 cvWaitKey();
 return 0;
}
```

9.4.3 OpenCV Timed Video/Frame Capture Source Code

The following code captures timed frames from the video camera and saves them to disk. The requirements are the same as that of the image processing module in A1.2. Initialization may be customized and graphical output is also available.

```
#include "cv.h"
#include "highgui.h"

#include <stdlib>
#include <stdio>
#include <math>
#include <fstream>
#include <ctime>

#define NUMROI 10

void DrawFrame(void)
{
 cvGrabFrame(capture);
 frame=cvRetrieveFrame(capture);
 cvShowImage("Capture",frame);
}

int main(int argc,char** argv)
{
  randomize();
  double times[NUMROI];
  for(int i=0;i<NUMROI;i++){times[i]=random(10);}
  std::clock_t start,end;
  cvNamedWindow("Capture",CV_WINDOW_AUTOSIZE);
  capture = cvCreateCameraCapture(-1);
  start=std::clock();
  while (cvWaitKey(1000)==-1)
  {
   if((std::clock()-start)/(double)CLOCKS_PER_SEC>times[i])
   {
    DrawFrame();
    start=std::clock();
   }
  }
  cvWaitKey();
  return 0;
}
```

9.4.4 Example: Fast Fourier Transform (FFT)

The following code demonstrates the use of complex variable arithmetic to perform the Fast Fourier Transform (FFT). The algorithm used is a relatively common FFT algorithm known as the 'Divide and Conquer' algorithm or the Cooley-Tukey Algorithm. The input is a complex-number array with given dimensions and the output array is the Fourier transform of the input array. The array sizes along any dimension MUST be a power of two. Provision is made for the inverse transform as well. Both 1D and 2D transforms are coded. The computational complexity is of ***O(N log N)*** where *N* is the total number of elements in the array.

```
//****************************************************************************************
***********
#include <math>
```

```
#include <string>
#include <stdlib>
//*****************************************************************************
***********
struct complex
{
 double x,y;
};
//*****************************************************************************
***********
complex Complex(double r,double i)
{
 complex c={r,i};
 return c;
}
//-------------------------------------------------------------------------------
complex Add(complex z1,complex z2)
{
 complex sum;
 sum.x=z1.x+z2.x;
 sum.y=z1.y+z2.y;
 return sum;
}
//-------------------------------------------------------------------------------
complex Multiply(complex z1,complex z2)
{
 complex product;
 product.x=(z1.x*z2.x)-(z1.y*z2.y);
 product.y=(z1.x*z2.y)+(z1.y*z2.x);
 return product;
}
//-------------------------------------------------------------------------------
complex Exp(complex z)
{
 double mag=exp(z.x);
 complex expo;
 expo.x=mag*cos(z.y);
 expo.y=mag*sin(z.y);
 return expo;
}
//*****************************************************************************
***********
void DFT(int N,complex* src,complex* des)
{
 for(int k=0;k<N;k++)
  {
   des[k]=Complex(0.0,0.0);
   for(int n=0;n<N;n++)
   {
    des[k]=Add(des[k],Multiply(Exp(Complex(0.0,-2.0*PI*(1.0/(double)N)*double(k*n))),src[n]));
   }
  }
}
//-------------------------------------------------------------------------------
void FFT2(int N1,int N2,complex* src,complex* des,bool inv)
{
 int p,j,k;
 int log2n1=0; p=1;
```

```
while(p<N1){p*=2;log2n1++;}
int log2n2=0; p=1;
while(p<N2){p*=2;log2n2++;}
N1=1<<log2n1;
N2=1<<log2n2;
for(int i=0;i<(N1*N2);i++)
{
 des[i]=src[i];
}
for(int y=0;y<N2;y++)
{
 j=0;
 for(int i=0;i<N1-1;i++)
 {
  des[(N2*i)+y]=src[(N2*j)+y];
  k=N1/2;
  while(k<=j){j-=k;k/=2;}
  j+=k;
 }
}
complex t;
for(int x=0;x<N1;x++)
{
 j=0;
 for(int i=0;i<N2-1;i++)
 {
  if(i<j)
  {
   t=des[(N2*x)+i];
   des[(N2*x)+i]=des[(N2*x)+j];
   des[(N2*x)+j]=t;
  }
  k=N2/2;
  while(k<=j){j-=k;k/=2;}
  j+=k;
 }
}
for(int x=0;x<N1;x++)
{
 double CA=-1.0;
 double SA=0.0;
 int l1=1,l2=1;
 for(int l=0;l<log2n1;l++)
 {
  l1=l2;
  l2*=2;
  double u1=1.0;
  double u2=0.0;
  for(int j=0;j<l1;j++)
  {
   for(int i=j;i<N1;i+=l2)
   {
    int i1=i+l1;
    double t1=(u1*des[(N2*x)+i1].x)-(u2*des[(N2*x)+i1].y);
    double t2=(u1*des[(N2*x)+i1].y)+(u2*des[(N2*x)+i1].x);
    des[(N2*x)+i1].x=des[(N2*x)+i].x-t1;
    des[(N2*x)+i1].y=des[(N2*x)+i].y-t2;
    des[(N2*x)+i].x+=t1;
```

```
        des[(N2*x)+i].y+=t2;
       }
       double z=(u1*CA)-(u2*SA);
       u2=(u1*SA)+(u2*CA);
       u1=z;
      }
     SA=sqrt((1.0-CA)/2.0);
     if(!inv){SA=-SA;}
     CA=sqrt((1.0+CA)/2.0);
    }
  }
  for(int y=0;y<N2;y++)
  {
   double CA=-1.0;
   double SA=0.0;
   int l1=1,l2=1;
   for(int l=0;l<log2n2;l++)
   {
    l1=l2;
    l2*=2;
    double u1=1.0;
    double u2=0.0;
    for(int j=0;j<l1;j++)
    {
     for(int i=j;i<N1;i+=l2)
     {
      int i1=i+l1;
      double t1=(u1*des[(N2*i1)+y].x)-(u2*des[(N2*i1)+y].y);
      double t2=(u1*des[(N2*i1)+y].y)+(u2*des[(N2*i1)+y].x);
      des[(N2*i1)+y].x=des[(N2*i)+y].x-t1;
      des[(N2*i1)+y].y=des[(N2*i)+y].y-t2;
      des[(N2*i)+y].x+=t1;
      des[(N2*i)+y].y+=t2;
     }
     double z=(u1*CA)-(u2*SA);
     u2=(u1*SA)+(u2*CA);
     u1=z;
    }
    SA=sqrt((1.0-CA)/2.0);
    if(!inv){SA=-SA;}
    CA=sqrt((1.0+CA)/2.0);
   }
  }
  double dee;
  if(inv){dee=N1;}else{dee=N2;}
  for(int x=0;x<N1;x++)
  {
   for(int y=0;y<N2;y++)
   {
    des[(N2*x)+y].x/=dee;
    des[(N2*x)+y].y/=dee;
   }
  }
}
//-------------------------------------------------------------------------------------------
```

9.4.5 MATLAB Source Codes

Image Acquisition and Synchronization

```
function DelayFunc(timetodelay,steps)
 t=timer('TasksToExecute',steps,'TimerFcn','MyTimerFunction',...
 'Period',timetodelay,'executionMode','fixedRate');
 start(t);
 wait(t);
end

ntarget=10;
times=zeros(ntarget,1);
frame=zeros(480,640,3);
for i=1:ntarget
 times(i)=i;
end
vid = videoinput('winvideo',1,'RGB24_640x480')
for i=1:ntarget
   DelayFunc(1,times(i));
   frame=getsnapshot(vid);
   fnam=[num2str(i),'.jpg'];
   imwrite(frame,fnam,'JPEG');
end
delete(vid);
clear vid;
```

Serial Port Communication

```
obj1 = instrfind('Type', 'serial', 'Port', 'COM1', 'Tag', '');
if isempty(obj1)
   obj1 = serial('COM1');
else
   fclose(obj1);
   obj1 = obj1(1)
end
fopen(obj1);
out='';
while isempty(out)
 out=fscanf(obj1,'%c',9)
end
fclose(obj1);
delete(obj1);
```

Graphical User Interface Source Code

```
%-------------------------------------
function varargout = Cube_dimensions(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
               'gui_Singleton',  gui_Singleton, ...
               'gui_OpeningFcn', @Cube_dimensions_OpeningFcn, ...
               'gui_OutputFcn',  @Cube_dimensions_OutputFcn, ...
               'gui_LayoutFcn',  [] , ...
               'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function Cube_dimensions_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = Cube_dimensions_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function edit1_Callback(hObject, eventdata, handles)
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function pushbutton1_Callback(hObject, eventdata, handles)
Frontpage;
close Cube_dimensions
function pushbutton2_Callback(hObject, eventdata, handles)
side=get(handles.edit1,'string');
dim=str2num(side);
if isempty(dim)
   errordlg('Please enter a number for the side!','Bad Input','modal')
   return
end
if dim<50
   errordlg('The dimension that you have entered is too low (Minimum 50mm)','Too low','modal')
   return
end
if dim>200
   errordlg('The dimension that you have entered is too high (Maximum 200mm)','Too high','modal')
   return
end
Cube_faces;
%-------------------------------------
function varargout = Cube_faces(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
             'gui_Singleton',  gui_Singleton, ...
             'gui_OpeningFcn', @Cube_faces_OpeningFcn, ...
             'gui_OutputFcn',  @Cube_faces_OutputFcn, ...
             'gui_LayoutFcn',  [] , ...
             'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function Cube_faces_OpeningFcn(hObject, eventdata, handles, varargin)
figd= Cube_dimensions;
cubedata=guidata(figd);
sidedim=get(cubedata.edit1,'string');
side=str2num(sidedim);
set(handles.text2,'string',side);
handles.output = hObject;
```

```
guidata(hObject, handles);
function varargout = Cube_faces_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function checkbox1_Callback(hObject, eventdata, handles)
function checkbox2_Callback(hObject, eventdata, handles)
function checkbox3_Callback(hObject, eventdata, handles)
function checkbox4_Callback(hObject, eventdata, handles)
function checkbox5_Callback(hObject, eventdata, handles)
function pushbutton1_Callback(hObject, eventdata, handles)
chckval1=get(handles.checkbox1,'value');
chckval2=get(handles.checkbox2,'value');
chckval3=get(handles.checkbox3,'value');
chckval4=get(handles.checkbox4,'value');
chckval5=get(handles.checkbox5,'value');
x=0;
if chckval1
   x=x+1;
end
if chckval2
    x=x+1;
end
if chckval3
    x=x+1;
end
if chckval4
    x=x+1;
end
if chckval5
    x=x+1;
end
if not(x)
    errordlg('Please Select at least one face!','Bad Input','modal')
   return
end
Cube_ROI;
function pushbutton2_Callback(hObject, eventdata, handles)
Frontpage;
close Cube_faces
function varargout = Cube_ROI(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
              'gui_Singleton',  gui_Singleton, ...
              'gui_OpeningFcn', @Cube_ROI_OpeningFcn, ...
              'gui_OutputFcn',  @Cube_ROI_OutputFcn, ...
              'gui_LayoutFcn',  [] , ...
              'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function Cube_ROI_OpeningFcn(hObject, eventdata, handles, varargin)
f1=Cube_dimensions;
cube_dimension_data=guidata(f1);
a=get(cube_dimension_data.edit1,'string');
```

```
set(handles.text6,'string',a);
cubefig=Cube_faces;
cubefigdata=guidata(cubefig);
chk1=get(cubefigdata.checkbox1,'value');
chk2=get(cubefigdata.checkbox2,'value');
chk3=get(cubefigdata.checkbox3,'value');
chk4=get(cubefigdata.checkbox4,'value');
chk5=get(cubefigdata.checkbox5,'value');
if chk1
   set(handles.text1,'visible','on');
   set(handles.togglebutton1,'visible','on');
   set(handles.togglebutton2,'visible','on');
   set(handles.togglebutton3,'visible','on');
   set(handles.togglebutton4,'visible','on');
   set(handles.togglebutton5,'visible','on');
   set(handles.togglebutton6,'visible','on');
   set(handles.togglebutton7,'visible','on');
   set(handles.togglebutton8,'visible','on');
   set(handles.togglebutton9,'visible','on');
end
if chk2
   set(handles.text2,'visible','on');
   set(handles.togglebutton10,'visible','on');
   set(handles.togglebutton11,'visible','on');
   set(handles.togglebutton12,'visible','on');
   set(handles.togglebutton13,'visible','on');
   set(handles.togglebutton14,'visible','on');
   set(handles.togglebutton15,'visible','on');
   set(handles.togglebutton16,'visible','on');
   set(handles.togglebutton17,'visible','on');
   set(handles.togglebutton18,'visible','on');
end
if chk3
   set(handles.text3,'visible','on');
   set(handles.togglebutton19,'visible','on');
   set(handles.togglebutton20,'visible','on');
   set(handles.togglebutton21,'visible','on');
   set(handles.togglebutton22,'visible','on');
   set(handles.togglebutton23,'visible','on');
   set(handles.togglebutton24,'visible','on');
   set(handles.togglebutton25,'visible','on');
   set(handles.togglebutton26,'visible','on');
   set(handles.togglebutton27,'visible','on');
end
if chk4
   set(handles.text4,'visible','on');
   set(handles.togglebutton28,'visible','on');
   set(handles.togglebutton29,'visible','on');
   set(handles.togglebutton30,'visible','on');
   set(handles.togglebutton31,'visible','on');
   set(handles.togglebutton32,'visible','on');
   set(handles.togglebutton33,'visible','on');
   set(handles.togglebutton34,'visible','on');
   set(handles.togglebutton35,'visible','on');
   set(handles.togglebutton36,'visible','on');
end
if chk5
   set(handles.text5,'visible','on');
```

```
    set(handles.togglebutton37,'visible','on');
    set(handles.togglebutton38,'visible','on');
    set(handles.togglebutton39,'visible','on');
    set(handles.togglebutton40,'visible','on');
    set(handles.togglebutton41,'visible','on');
    set(handles.togglebutton42,'visible','on');
    set(handles.togglebutton43,'visible','on');
    set(handles.togglebutton44,'visible','on');
    set(handles.togglebutton45,'visible','on');
end
handles.output = hObject;
guidata(hObject, handles);
function varargout = Cube_ROI_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function togglebutton1_Callback(hObject, eventdata, handles)
function togglebutton2_Callback(hObject, eventdata, handles)
function togglebutton3_Callback(hObject, eventdata, handles)
function togglebutton4_Callback(hObject, eventdata, handles)
function togglebutton5_Callback(hObject, eventdata, handles)
function togglebutton6_Callback(hObject, eventdata, handles)
function togglebutton7_Callback(hObject, eventdata, handles)
function togglebutton8_Callback(hObject, eventdata, handles)
function togglebutton9_Callback(hObject, eventdata, handles)
function togglebutton10_Callback(hObject, eventdata, handles)
function togglebutton11_Callback(hObject, eventdata, handles)
function togglebutton12_Callback(hObject, eventdata, handles)
function togglebutton13_Callback(hObject, eventdata, handles)
function togglebutton14_Callback(hObject, eventdata, handles)
function togglebutton15_Callback(hObject, eventdata, handles)
function togglebutton16_Callback(hObject, eventdata, handles)
function togglebutton17_Callback(hObject, eventdata, handles)
function togglebutton18_Callback(hObject, eventdata, handles)
function togglebutton19_Callback(hObject, eventdata, handles)
function togglebutton20_Callback(hObject, eventdata, handles)
function togglebutton21_Callback(hObject, eventdata, handles)
function togglebutton22_Callback(hObject, eventdata, handles)
function togglebutton23_Callback(hObject, eventdata, handles)
function togglebutton24_Callback(hObject, eventdata, handles)
function togglebutton25_Callback(hObject, eventdata, handles)
function togglebutton26_Callback(hObject, eventdata, handles)
function togglebutton27_Callback(hObject, eventdata, handles)
function togglebutton28_Callback(hObject, eventdata, handles)
function togglebutton29_Callback(hObject, eventdata, handles)
function togglebutton30_Callback(hObject, eventdata, handles)
function togglebutton31_Callback(hObject, eventdata, handles)
function togglebutton32_Callback(hObject, eventdata, handles)
function togglebutton33_Callback(hObject, eventdata, handles)
function togglebutton34_Callback(hObject, eventdata, handles)
function togglebutton35_Callback(hObject, eventdata, handles)
function togglebutton36_Callback(hObject, eventdata, handles)
function togglebutton37_Callback(hObject, eventdata, handles)
function togglebutton38_Callback(hObject, eventdata, handles)
function togglebutton39_Callback(hObject, eventdata, handles)
function togglebutton40_Callback(hObject, eventdata, handles)
function togglebutton41_Callback(hObject, eventdata, handles)
function togglebutton42_Callback(hObject, eventdata, handles)
function togglebutton43_Callback(hObject, eventdata, handles)
function togglebutton44_Callback(hObject, eventdata, handles)
function togglebutton45_Callback(hObject, eventdata, handles)
function pushbutton1_Callback(hObject, eventdata, handles)
Frontpage;
close Cube_ROI
function pushbutton2_Callback(hObject, eventdata, handles)
Types_of_inspection_cube;
%-----------------------------------
function varargout = Cylinder_dimensions(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
              'gui_Singleton',  gui_Singleton, ...
              'gui_OpeningFcn', @Cylinder_dimensions_OpeningFcn, ...
              'gui_OutputFcn',  @Cylinder_dimensions_OutputFcn, ...
              'gui_LayoutFcn',  [] , ...
              'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function Cylinder_dimensions_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = Cylinder_dimensions_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function edit1_Callback(hObject, eventdata, handles)
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function edit2_Callback(hObject, eventdata, handles)
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function pushbutton1_Callback(hObject, eventdata, handles)
Frontpage;
close Cylinder_dimensions
function pushbutton2_Callback(hObject, eventdata, handles)
a=get(handles.edit1,'string');
num1=str2num(a);
if isempty(num1)
   errordlg('Please enter a number for the radius!','Bad Input','modal')
   return
end
if num1<50
   errordlg('The radius that you have entered is too low (Minimum 50mm)','Too low','modal')
   return
end
if num1>200
   errordlg('The radius that you have entered is too high (Maximum 200mm)','Too high','modal')
   return
end
b=get(handles.edit2,'string');
num2=str2num(b);
```

```
if isempty(num2)
   errordlg('Please enter a number for height!','Bad Input','modal')
   return
end
if num2<40
   errordlg('The height that you have entered is too low (Minimum 40mm)','Too low','modal')
   return
end
if num2>200
   errordlg('The height that you have entered is too high (Maximum 200mm)','Too high','modal')
   return
end
Cylinder_faces;
%-------------------------------------
function varargout = Cylinder_faces(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Cylinder_faces_OpeningFcn, ...
                   'gui_OutputFcn',  @Cylinder_faces_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function Cylinder_faces_OpeningFcn(hObject, eventdata, handles, varargin)
figurehandle=Cylinder_dimensions;
dimension_data=guidata(figurehandle);
a=get(dimension_data.edit1,'string');
b=get(dimension_data.edit2,'string');
set(handles.text1,'string',a);
set(handles.text2,'string',b);
handles.output = hObject;
guidata(hObject, handles);
function varargout = Cylinder_faces_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function pushbutton1_Callback(hObject, eventdata, handles)
Frontpage;
close Cylinder_faces
function pushbutton2_Callback(hObject, eventdata, handles)
chckval1=get(handles.checkbox1,'value');
chckval2=get(handles.checkbox2,'value');
chckval3=get(handles.checkbox3,'value');
chckval4=get(handles.checkbox4,'value');
chckval5=get(handles.checkbox5,'value');
x=0;
if chckval1
   x=x+1;
end
if chckval2
    x=x+1;
end
if chckval3
```

```
    x=x+1;
end
if chckval4
    x=x+1;
end
if chckval5
    x=x+1;
end
if not(x)
    errordlg('Please Select at least one face!','Bad Input','modal')
    return
end
Cylinder_ROI;
function checkbox1_Callback(hObject, eventdata, handles)
function checkbox2_Callback(hObject, eventdata, handles)
function checkbox3_Callback(hObject, eventdata, handles)
function checkbox4_Callback(hObject, eventdata, handles)
function checkbox5_Callback(hObject, eventdata, handles)
%-------------------------------------
function varargout = Cylinder_ROI(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
              'gui_Singleton',  gui_Singleton, ...
              'gui_OpeningFcn', @Cylinder_ROI_OpeningFcn, ...
              'gui_OutputFcn',  @Cylinder_ROI_OutputFcn, ...
              'gui_LayoutFcn',  [] , ...
              'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function Cylinder_ROI_OpeningFcn(hObject, eventdata, handles, varargin)
figurehandle=Cylinder_dimensions;
dimension_data=guidata(figurehandle);
a1=get(dimension_data.edit1,'string');
b1=get(dimension_data.edit2,'string');
set(handles.text5,'string',a1);
set(handles.text6,'string',b1);
gui=Cylinder_faces;
checkdata=guidata(gui);
chk1=get(checkdata.checkbox1,'value');
chk2=get(checkdata.checkbox2,'value');
chk3=get(checkdata.checkbox3,'value');
chk4=get(checkdata.checkbox4,'value');
chk5=get(checkdata.checkbox5,'value');
if chk1
   set(handles.text1,'visible','on');
   set(handles.togglebutton1,'visible','on');
   set(handles.togglebutton5,'visible','on');
end
if chk2
   set(handles.text2,'visible','on');
   set(handles.togglebutton12,'visible','on');
   set(handles.togglebutton13,'visible','on');
```

```
end
if chk3
   set(handles.text3,'visible','on');
   set(handles.togglebutton6,'visible','on');
   set(handles.togglebutton7,'visible','on');
end
if chk4
   set(handles.text4,'visible','on');
   set(handles.togglebutton10,'visible','on');
   set(handles.togglebutton11,'visible','on');
end
if chk5
   set(handles.text7,'visible','on');
   set(handles.togglebutton14,'visible','on');
   set(handles.togglebutton15,'visible','on');
end
handles.output = hObject;
guidata(hObject, handles);
function varargout = Cylinder_ROI_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function togglebutton1_Callback(hObject, eventdata, handles)
function togglebutton5_Callback(hObject, eventdata, handles)
function togglebutton6_Callback(hObject, eventdata, handles)
function togglebutton7_Callback(hObject, eventdata, handles)
function togglebutton10_Callback(hObject, eventdata, handles)
function togglebutton11_Callback(hObject, eventdata, handles)
function togglebutton12_Callback(hObject, eventdata, handles)
function togglebutton13_Callback(hObject, eventdata, handles)
function pushbutton1_Callback(hObject, eventdata, handles)
Types_of_inspection_Cylinder;
function pushbutton2_Callback(hObject, eventdata, handles)
Frontpage;
close Cylinder_ROI
function togglebutton14_Callback(hObject, eventdata, handles)
function togglebutton15_Callback(hObject, eventdata, handles)
%-------------------------------------
function varargout = Frontpage(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
              'gui_Singleton',  gui_Singleton, ...
              'gui_OpeningFcn', @Frontpage_OpeningFcn, ...
              'gui_OutputFcn',  @Frontpage_OutputFcn, ...
              'gui_LayoutFcn',  [] , ...
              'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function Frontpage_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = Frontpage_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function pushbutton1_Callback(hObject, eventdata, handles)
function pushbutton2_Callback(hObject, eventdata, handles)
close
function pushbutton3_Callback(hObject, eventdata, handles)
function pushbutton4_Callback(hObject, eventdata, handles)
select_part_family;
close Frontpage;
function pushbutton5_Callback(hObject, eventdata, handles)
%-----------------------------------
function varargout = getserialdata(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
              'gui_Singleton',  gui_Singleton, ...
              'gui_OpeningFcn', @getserialdata_OpeningFcn, ...
              'gui_OutputFcn',  @getserialdata_OutputFcn, ...
              'gui_LayoutFcn',  [] , ...
              'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function getserialdata_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = getserialdata_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function pushbutton1_Callback(hObject, eventdata, handles)
s = serial('COM1');
set(s,'BaudRate',4800);
fopen(s);
fprintf(s,'*IDN?')
out = fscanf(s);
fclose(s)
delete(s)
clear s
%-----------------------------------
function varargout = openfile(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
              'gui_Singleton',  gui_Singleton, ...
              'gui_OpeningFcn', @openfile_OpeningFcn, ...
              'gui_OutputFcn',  @openfile_OutputFcn, ...
              'gui_LayoutFcn',  [] , ...
              'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function openfile_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
```

```
function varargout = openfile_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function pushbutton1_Callback(hObject, eventdata, handles)
fid = fopen('fgetl.m');
while 1
   tline = fgetl(fid);
   if ~ischar(tline),  break,  end
   disp(tline)
end
fclose(fid);
%-----------------------------------
function varargout = Reactangular_volume_faces_for_inspection(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
              'gui_Singleton',  gui_Singleton, ...
              'gui_OpeningFcn', @Reactangular_volume_faces_for_inspection_OpeningFcn, ...
              'gui_OutputFcn',  @Reactangular_volume_faces_for_inspection_OutputFcn, ...
              'gui_LayoutFcn',  [] , ...
              'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function   Reactangular_volume_faces_for_inspection_OpeningFcn(hObject,   eventdata,   handles,
varargin)
figurehandle=Rectangle_dimensions;
dimension_data=guidata(figurehandle);
a=get(dimension_data.edit1,'string');
b=get(dimension_data.edit2,'string');
c=get(dimension_data.edit3,'string');
set(handles.text3,'string',a);
set(handles.text4,'string',b);
set(handles.text5,'string',c);
handles.output = hObject;
guidata(hObject, handles);
function   varargout   =   Reactangular_volume_faces_for_inspection_OutputFcn(hObject,   eventdata,
handles)
varargout{1} = handles.output;
function pushbutton1_Callback(hObject, eventdata, handles)
Frontpage;
close Reactangular_volume_faces_for_inspection
function pushbutton2_Callback(hObject, eventdata, handles)
chckval1=get(handles.checkbox1,'value');
chckval2=get(handles.checkbox2,'value');
chckval3=get(handles.checkbox3,'value');
chckval4=get(handles.checkbox4,'value');
chckval5=get(handles.checkbox5,'value');
a=0;
if chckval1
   a=a+1;
end
 if chckval2
   a=a+1;
 end
 if chckval3
   a=a+1;
 end
 if chckval4
   a=a+1;
 end
 if chckval5
   a=a+1;
 end
 if not(a)
   errordlg('Please Select at least one face!','Bad Input','modal')
   return
 end
Rectangle_ROI;
function checkbox1_Callback(hObject, eventdata, handles)
function checkbox2_Callback(hObject, eventdata, handles)
function checkbox3_Callback(hObject, eventdata, handles)
function checkbox4_Callback(hObject, eventdata, handles)
function checkbox5_Callback(hObject, eventdata, handles)
%---------------------------------------
function varargout = Rectangle_dimensions(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
             'gui_Singleton',  gui_Singleton, ...
             'gui_OpeningFcn', @Rectangle_dimensions_OpeningFcn, ...
             'gui_OutputFcn',  @Rectangle_dimensions_OutputFcn, ...
             'gui_LayoutFcn',  [] , ...
             'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function Rectangle_dimensions_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = Rectangle_dimensions_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function edit1_Callback(hObject, eventdata, handles)
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function edit2_Callback(hObject, eventdata, handles)
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function edit3_Callback(hObject, eventdata, handles)
function edit3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function pushbutton1_Callback(hObject, eventdata, handles)
Frontpage;
```

```
close Rectangle_dimensions
function pushbutton2_Callback(hObject, eventdata, handles)
num1=get(handles.edit1,'string');
a=str2num(num1);
num2=get(handles.edit2,'string');
b=str2num(num2);
num3=get(handles.edit3,'string');
c=str2num(num3);
if isempty(a) %makes sure that the user enters numbers only for each dimension
   errordlg('Please enter numbers only!','Bad Input','modal')
   return
end
if isempty(b)
  errordlg('Please enter numbers only!','Bad Input','modal')
return
end
if isempty(c)
  errordlg('Please enter numbers only!','Bad Input','modal')
return
end
if (a<50)
    errordlg('The minimum value for a is 50mm. Please enter a higher value','Bad Input','modal')
   return
elseif (a>200)
    errordlg('The maximum value for a is 200mm. Please enter a lower value','Bad Input','modal')
   return
end
if (b<30)
    errordlg('The minimum value for b is 30mm. Please enter a higher value','Bad Input','modal')
   return
elseif (b>300)
    errordlg('The maximum value for b is 300mm. Please enter a lower value','Bad Input','modal')
   return
end
if (c<50)
    errordlg('The minimum value for c is 50mm. Please enter a higher value','Bad Input','modal')
   return
elseif (a>180)
    errordlg('The maximum value for c is 180mm. Please enter a lower value','Bad Input','modal')
   return
end
Reactangular_volume_faces_for_inspection;
%------------------------------------
function varargout = Rectangle_ROI(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
              'gui_Singleton',  gui_Singleton, ...
              'gui_OpeningFcn', @Rectangle_ROI_OpeningFcn, ...
              'gui_OutputFcn',  @Rectangle_ROI_OutputFcn, ...
              'gui_LayoutFcn',  [] , ...
              'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
```

```
end
function Rectangle_ROI_OpeningFcn(hObject, eventdata, handles, varargin)
figurehandle=Rectangle_dimensions;
dimension_data=guidata(figurehandle);
a1=get(dimension_data.edit1,'string');
b1=get(dimension_data.edit2,'string');
c1=get(dimension_data.edit3,'string');
a=str2num(a1);
b=str2num(b1);
c=str2num(c1);
set(handles.text6,'string',a);
set(handles.text7,'string',b);
set(handles.text8,'string',c);
figh=Reactangular_volume_faces_for_inspection;
figdata=guidata(figh);
chk1=get(figdata.checkbox1,'value');
chk2=get(figdata.checkbox2,'value');
chk3=get(figdata.checkbox3,'value');
chk4=get(figdata.checkbox4,'value');
chk5=get(figdata.checkbox5,'value');
if chk1
   set(handles.text1,'visible','on');
   set(handles.togglebutton65,'visible','on');
   set(handles.togglebutton66,'visible','on');
   set(handles.togglebutton67,'visible','on');
   set(handles.togglebutton68,'visible','on');
   set(handles.togglebutton69,'visible','on');
   set(handles.togglebutton70,'visible','on');
   set(handles.togglebutton71,'visible','on');
   set(handles.togglebutton72,'visible','on');
   set(handles.togglebutton73,'visible','on');
end
if chk2
   set(handles.text2,'visible','on');
   set(handles.togglebutton92,'visible','on');
   set(handles.togglebutton93,'visible','on');
   set(handles.togglebutton94,'visible','on');
   set(handles.togglebutton95,'visible','on');
   set(handles.togglebutton96,'visible','on');
   set(handles.togglebutton97,'visible','on');
   set(handles.togglebutton98,'visible','on');
   set(handles.togglebutton99,'visible','on');
   set(handles.togglebutton100,'visible','on');
end
if chk3
   set(handles.text3,'visible','on');
   set(handles.togglebutton101,'visible','on');
   set(handles.togglebutton102,'visible','on');
   set(handles.togglebutton103,'visible','on');
   set(handles.togglebutton104,'visible','on');
   set(handles.togglebutton105,'visible','on');
   set(handles.togglebutton106,'visible','on');
   set(handles.togglebutton107,'visible','on');
   set(handles.togglebutton108,'visible','on');
   set(handles.togglebutton109,'visible','on');
end
if chk4
   set(handles.text4,'visible','on');
```

```
    set(handles.togglebutton110,'visible','on');
    set(handles.togglebutton111,'visible','on');
    set(handles.togglebutton112,'visible','on');
    set(handles.togglebutton113,'visible','on');
    set(handles.togglebutton114,'visible','on');
    set(handles.togglebutton115,'visible','on');
    set(handles.togglebutton116,'visible','on');
    set(handles.togglebutton117,'visible','on');
    set(handles.togglebutton118,'visible','on');
end
if chk5
    set(handles.text5,'visible','on');
    set(handles.togglebutton119,'visible','on');
    set(handles.togglebutton120,'visible','on');
    set(handles.togglebutton121,'visible','on');
    set(handles.togglebutton122,'visible','on');
    set(handles.togglebutton123,'visible','on');
    set(handles.togglebutton124,'visible','on');
    set(handles.togglebutton125,'visible','on');
    set(handles.togglebutton126,'visible','on');
    set(handles.togglebutton127,'visible','on');
end
handles.output = hObject;
guidata(hObject, handles);
function varargout = Rectangle_ROI_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function pushbutton1_Callback(hObject, eventdata, handles)
counter1=0;
counter2=0;
counter3=0;
counter4=0;
counter5=0;
t65=get(handles.togglebutton65,'value');
t66=get(handles.togglebutton66,'value');
t67=get(handles.togglebutton67,'value');
t68=get(handles.togglebutton68,'value');
t69=get(handles.togglebutton69,'value');
t70=get(handles.togglebutton70,'value');
t71=get(handles.togglebutton71,'value');
t72=get(handles.togglebutton72,'value');
t73=get(handles.togglebutton73,'value');
if t65
    counter1=counter1+1;
end
if t66
    counter1=counter1+1;
end
if t67
    counter1=counter1+1;
end
if t68
    counter1=counter1+1;
end
if t69
    counter1=counter1+1;
end
if t70
    counter1=counter1+1;
```

```
end
if t71
   counter1=counter1+1;
end
if t72
   counter1=counter1+1;
end
if t73
   counter1=counter1+1;
end
t92=get(handles.togglebutton92,'value');
t93=get(handles.togglebutton93,'value');
t94=get(handles.togglebutton94,'value');
t95=get(handles.togglebutton95,'value');
t96=get(handles.togglebutton96,'value');
t97=get(handles.togglebutton97,'value');
t98=get(handles.togglebutton98,'value');
t99=get(handles.togglebutton99,'value');
t100=get(handles.togglebutton100,'value');
if t92
   counter2=counter2+1;
end
if t93
   counter2=counter2+1;
end
if t94
   counter2=counter2+1;
end
if t95
   counter2=counter2+1;
end
if t96
   counter2=counter2+1;
end
if t97
   counter2=counter2+1;
end
if t98
   counter2=counter2+1;
end
if t99
   counter2=counter2+1;
end
if t100
   counter2=counter2+1;
end

t101=get(handles.togglebutton101,'value');
t102=get(handles.togglebutton102,'value');
t103=get(handles.togglebutton103,'value');
t104=get(handles.togglebutton104,'value');
t105=get(handles.togglebutton105,'value');
t106=get(handles.togglebutton106,'value');
t107=get(handles.togglebutton107,'value');
t108=get(handles.togglebutton108,'value');
t109=get(handles.togglebutton109,'value');
if t101
   counter3=counter3+1;
```

```
end
if t102
   counter3=counter3+1;
end
if t103
   counter3=counter3+1;
end
if t104
   counter3=counter3+1;
end
if t105
   counter3=counter3+1;
end
if t106
   counter3=counter3+1;
end
if t107
   counter3=counter3+1;
end
if t108
   counter3=counter3+1;
end
if t109
   counter3=counter3+1;
end
t110=get(handles.togglebutton110,'value');
t111=get(handles.togglebutton111,'value');
t112=get(handles.togglebutton112,'value');
t113=get(handles.togglebutton113,'value');
t114=get(handles.togglebutton114,'value');
t115=get(handles.togglebutton115,'value');
t116=get(handles.togglebutton116,'value');
t117=get(handles.togglebutton117,'value');
t118=get(handles.togglebutton118,'value');
if t110
   counter4=counter4+1;
end
if t111
   counter4=counter4+1;
end
if t112
   counter4=counter4+1;
end
if t113
   counter4=counter4+1;
end
if t114
   counter4=counter4+1;
end
if t115
   counter4=counter4+1;
end
if t116
   counter4=counter4+1;
end
if t117
   counter4=counter4+1;
end
```

```
if t118
   counter4=counter4+1;
end
t119=get(handles.togglebutton119,'value');
t120=get(handles.togglebutton120,'value');
t121=get(handles.togglebutton121,'value');
t122=get(handles.togglebutton122,'value');
t123=get(handles.togglebutton123,'value');
t124=get(handles.togglebutton124,'value');
t125=get(handles.togglebutton125,'value');
t126=get(handles.togglebutton126,'value');
t127=get(handles.togglebutton127,'value');
if t119
   counter5=counter5+1;
end

if t120
   counter5=counter5+1;
end
if t121
   counter5=counter5+1;
end
if t122
   counter5=counter5+1;
end
if t123
   counter5=counter5+1;
end
if t124
   counter5=counter5+1;
end
if t125
   counter5=counter5+1;
end
if t126
   counter5=counter5+1;
end
if t127
   counter5=counter5+1;
end

total=counter1+counter2+counter3+counter4+counter5;
if (total>8)
   errordlg('The maximum number of ROIs allowed is 8','Bad Input','modal')
return
end
if (total<1)
   errordlg('Please select at least one ROI','Bad Input','modal')
return
end
Types_of_inspection_Rectangle;

function pushbutton2_Callback(hObject, eventdata, handles)

Frontpage;
close Rectangle_ROI
%--------------------------------------
function varargout = Resolution_for_ROI(varargin)
```

```
gui_Singleton = 1;
gui_State = struct('gui_Name',      mfilename, ...
            'gui_Singleton',  gui_Singleton, ...
            'gui_OpeningFcn', @Resolution_for_ROI_OpeningFcn, ...
            'gui_OutputFcn',  @Resolution_for_ROI_OutputFcn, ...
            'gui_LayoutFcn',  [] , ...
            'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function Resolution_for_ROI_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = Resolution_for_ROI_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
%-------------------------------------
function varargout = select_part_family(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',      mfilename, ...
            'gui_Singleton',  gui_Singleton, ...
            'gui_OpeningFcn', @select_part_family_OpeningFcn, ...
            'gui_OutputFcn',  @select_part_family_OutputFcn, ...
            'gui_LayoutFcn',  [] , ...
            'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function select_part_family_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = select_part_family_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function radiobutton1_Callback(hObject, eventdata, handles)
a=get(hObject,'value');
if a
   Rectangle_dimensions;
   set(handles.radiobutton2,'value',0);
   set(handles.radiobutton3,'value',0);
end
function pushbutton1_Callback(hObject, eventdata, handles)
function pushbutton2_Callback(hObject, eventdata, handles)
function radiobutton2_Callback(hObject, eventdata, handles)
b=get(hObject,'value');
if b
   Cylinder_dimensions;
   set(handles.radiobutton1,'value',0);
   set(handles.radiobutton3,'value',0);
end
```

```
function radiobutton3_Callback(hObject, eventdata, handles)
c=get(hObject,'value');
if c
   Cube_dimensions;
   set(handles.radiobutton2,'value',0);
   set(handles.radiobutton1,'value',0);
end
function pushbutton3_Callback(hObject, eventdata, handles)
Frontpage;
close select_part_family;
%-------------------------------------
function varargout = Types_of_inspection(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
               'gui_Singleton',  gui_Singleton, ...
               'gui_OpeningFcn', @Types_of_inspection_OpeningFcn, ...
               'gui_OutputFcn',  @Types_of_inspection_OutputFcn, ...
               'gui_LayoutFcn',  [] , ...
               'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function Types_of_inspection_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = Types_of_inspection_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function checkbox1_Callback(hObject, eventdata, handles)
function checkbox2_Callback(hObject, eventdata, handles)
function checkbox3_Callback(hObject, eventdata, handles)
function edit2_Callback(hObject, eventdata, handles)
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function pushbutton1_Callback(hObject, eventdata, handles)
Frontpage;
close Types_of_inspection
function pushbutton2_Callback(hObject, eventdata, handles)
%-------------------------------------
function varargout = Types_of_inspection(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
               'gui_Singleton',  gui_Singleton, ...
               'gui_OpeningFcn', @Types_of_inspection_OpeningFcn, ...
               'gui_OutputFcn',  @Types_of_inspection_OutputFcn, ...
               'gui_LayoutFcn',  [] , ...
               'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
   gui_mainfcn(gui_State, varargin{:});
end
function Types_of_inspection_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = Types_of_inspection_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function checkbox1_Callback(hObject, eventdata, handles)
function checkbox2_Callback(hObject, eventdata, handles)
function checkbox3_Callback(hObject, eventdata, handles)
function edit2_Callback(hObject, eventdata, handles)
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function pushbutton1_Callback(hObject, eventdata, handles)
Frontpage;
close Types_of_inspection
function pushbutton2_Callback(hObject, eventdata, handles)
%-------------------------------------
function varargout = Types_of_inspection(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
              'gui_Singleton',  gui_Singleton, ...
              'gui_OpeningFcn', @Types_of_inspection_OpeningFcn, ...
              'gui_OutputFcn',  @Types_of_inspection_OutputFcn, ...
              'gui_LayoutFcn',  [] , ...
              'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function Types_of_inspection_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = Types_of_inspection_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function checkbox1_Callback(hObject, eventdata, handles)
function checkbox2_Callback(hObject, eventdata, handles)
function checkbox3_Callback(hObject, eventdata, handles)
function edit2_Callback(hObject, eventdata, handles)
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function pushbutton1_Callback(hObject, eventdata, handles)
Frontpage;
close Types_of_inspection
function pushbutton2_Callback(hObject, eventdata, handles)
%-------------------------------------
function varargout = Types_of_inspection(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
            'gui_Singleton',  gui_Singleton, ...
            'gui_OpeningFcn', @Types_of_inspection_OpeningFcn, ...
            'gui_OutputFcn',  @Types_of_inspection_OutputFcn, ...
            'gui_LayoutFcn',  [] , ...
            'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
function Types_of_inspection_OpeningFcn(hObject, eventdata, handles, varargin)
rect_dim=Rectangle_dimensions;
dim_info=guidata(rect_dim);
dim1=str2num(get(dim_info.edit1,'string'));
dim2=str2num(get(dim_info.edit2,'string'));
dim3=str2num(get(dim_info.edit3,'string'));
set(handles.text50,'string',dim1);
set(handles.text51,'string',dim2);
set(handles.text52,'string',dim3);
roi=Rectangle_ROI;
roi_info=guidata(roi);
handles.output = hObject;
guidata(hObject, handles);
function varargout = Types_of_inspection_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function checkbox1_Callback(hObject, eventdata, handles)
function checkbox2_Callback(hObject, eventdata, handles)
function checkbox3_Callback(hObject, eventdata, handles)
function edit2_Callback(hObject, eventdata, handles)
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function pushbutton1_Callback(hObject, eventdata, handles)
Frontpage;
close Types_of_inspection
function pushbutton2_Callback(hObject, eventdata, handles)
%--------------------------------------
```

9.4. 6 Microcontroller code

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.24.5 Evaluation
Automatic Program Generator
© Copyright 1998-2005 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com
e-mail:office@hpinfotech.com

Project : Davrajh16PWM
Version :
Date    : 2008/08/16
Author  : Freeware, for evaluation and non-commercial use only
Company :
Comments:


Chip type          : ATmega16
Program type       : Application
Clock frequency    : 4.000000 MHz
Memory model       : Small
External SRAM size : 0
Data Stack size    : 256
*****************************************************/

//#include <mega16.h>
#include <mega32.h>
//#include "delay.h"
// Standard Input/Output functions
/*
const unsigned char achMotor1[4] = {'M','T','1','\0'};
const unsigned char achMotor2[4] = {'M','T','2','\0'};
const unsigned char achMotor3[4] = {'M','T','3','\0'};
*/

unsigned char uchSpeed3 = 0;
unsigned int uiPWM0 = 0;
unsigned int uiPWM1 = 0;
unsigned int uiPWM2 = 0;
unsigned char uchDelay = 0;


void USART_Transmit( unsigned char data )
{
/* Wait for empty transmit buffer */
while ( !( UCSRA & (1<<5)) )
;
/* Put data into buffer, sends the data */
UDR = data;
}

unsigned char USART_Receive( void )
{
/* Wait for data to be received */
while ( !(UCSRA & (1<<7)) )
;
/* Get and return received data from buffer */
```

```
return UDR;
}
#if 0
void SPI_MasterInit(void)
{
/* Set MOSI and SCK output, all others input */
DDRB = (1<<5)|(1<<7);
/* Enable SPI, Master, set clock rate fck/16 */
SPCR = (1<<6)|(1<<4)|(1<<0);
}

void SPI_MasterTransmit(char cData)
{
/* Start transmission */
SPDR = cData;
/* Wait for transmission complete */
while(!(SPSR & (1<<7)))
;
}
#define ADC_VREF_TYPE 0x20
// Read the 8 most significant bits
// of the AD conversion result
unsigned char read_adc(unsigned char adc_input)
{
ADMUX=adc_input|ADC_VREF_TYPE;
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCH;
}
#endif
// Declare your global variables here

void main(void)
{
unsigned char afMotor[3];
unsigned char  achInput[4] = {0,0,0,0};
unsigned char chtemp = 0;
unsigned char uchSpeed1 = 0;
unsigned char uchSpeed2 = 0;

// Declare your local variables here



afMotor[0] = 0;
afMotor[1] = 0;
afMotor[2] = 0;
// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0xFF;
DDRB=0xFF;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD =0xFF;
DDRD =0xFF;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0= 0x6A ; //0x62;
//0x02|0x40|0x20;
TCNT0=0x00;
OCR0=0xFA;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x01;
TCCR1B=0x02;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0xFA;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x7A;
TCNT2=0x00;
OCR2=0xF0;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 9600
UCSRA=0x00;
UCSRB=0x18;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x19;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
#if 0
ACSR=0x80;
SFIOR=0x00;

// ADC initialization
// ADC Clock frequency: 125.000 kHz
// ADC Voltage Reference: AREF pin
// ADC Auto Trigger Source: None
// Only the 8 most significant bits of
// the AD conversion result are used
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x85;
#endif
//SPI_MasterInit();
while (1)
    {
      PORTB = 0x0F ;
      chtemp = USART_Receive();
#if 0
      USART_Transmit(USART_Receive());
      if (uchDelay == 1)
      {
        OCR0 = (uiPWM0 * 100)/255;
        OCR1AL = (uiPWM1 * 100)/255;
        OCR2 = (uiPWM1 * 100)/255;
        delay_ms((unsigned char)(chtemp*10)); /* centi seconds */
        uchDelay = 0;
      }
      if (afMotor[0])
      {
        uchSpeed1 = chtemp;
        uiPWM0 = ((int)uchSpeed1*100)/209;
        afMotor[0] = 0;
      }
      else if (afMotor[1])
```

```
        {
          uchSpeed2 = chtemp;
          uiPWM1 = ((int)uchSpeed2*100)/229;
          afMotor[1] = 0;
        }
        else if (afMotor[2])
        {
          uchSpeed3 = chtemp;
          uiPWM2 = ((int)uchSpeed3*100)/50;
          afMotor[2] = 0;
          uchDelay = 1;
        }

        if (chtemp == 'M' )
        {
          achInput[0] = chtemp;
          continue;
        }
        else if ((chtemp == 'T')&&(achInput[0] == 'M'))
        {
          achInput[1] = chtemp;
          continue;
        }
        else if ((achInput[0] == 'M')&&(achInput[1] == 'T'))
        {
         if ((chtemp == 0)||(chtemp == 1)||(chtemp == 2))
         {
           afMotor[chtemp] = 1;
         }
         else
         {
           achInput[0] = 0;
           achInput[1] = 0;
           achInput[2] = 0;
           achInput[3] = 0;
         }
         continue;
        }

  //     USART_Transmit(USART_Receive());
  //     PORTB = cVariable ;
  //     cVariable += 10;
  //     PORTB &= 0xFA;
  //     OCR0 = cVariable;
  //     OCR1AL = cVariable;
  //     OCR1BL = cVariable;
  //     OCR2 = cVariable;
#endif
/*
#asm
        nop;
        nop;
        nop;
        nop;
#endasm */
    // Place your code here
    };
}
```