IDENTIFICATION AND INSPECTION

OF SILVERWARE USING

MACHINE VISION

By

SAI VENU GOPAL LOLLA

Bachelor of Technology

Jawaharlal Nehru Technological University

Hyderabad, India.

2002

Submitted to the Faculty of the Graduate College of the Oklahoma State University in partial fulfillment of the requirements for the Degree of
MASTER OF SCIENCE
May 2005

IDENTIFICATION AND INSPECTION

OF SILVERWARE USING

MACHINE VISION

Thesis Approved:

DR.LAWRENCE L HOBEROCK

Thesis Advisor

DR.GARY E YOUNG

DR.GARY G YEN

DR.A.GORDON EMSLIE

Dean of the Graduate College

ACKNOWLEDGMENTS

I am indebted to my Graduate Advisor, Dr. Lawrence L. Hoberock, and wish to express my sincere gratitude for his constant support, understanding, patient guidance, and encouragement throughout my graduate program. I wish to express my sincere thanks to Dr. Gary E. Young and Dr. Gary G. Yen for serving on my Graduate Committee.

I also express my sincere thanks to Mr. Jerry Dale, CEAT Lab Manager for his guidance. I extend my thanks to Mr. Matt Benson, Mr. Brett Riegel, and Mr.Vijay Serrao, MAE Electronics Shop for their assistance. I thank Mr.Sandeep Yeri, Ms.Shilpa Nagaraj, Mr.Vamshi Peddi, and Mr.Simon Jeyapalan for their constant support. I express special thanks to Ms.Anupama Balachandran for having allowed me to use her camera during the course of my experimentation.

I also wish to express my sincere thanks and appreciation to my family and friends for their unfailing support and constant encouragement in all my pursuits.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

1.1 – Silverware Identification/Inspection – A Base for New Technology

Identification and inspection of silverware pieces pose many challenges to the field of machine vision. The challenges include the need for a high processing rate to make the technology commercially feasible, the need for an efficient imaging setup to acquire high quality images of objects with specularly reflective complex geometric surfaces, the need for a fast identification technique to reliably identify objects, and the need for a fast inspection procedure to inspect the surfaces of the objects for anomalies. The problem presents scope for the development of new methods to solve various existing problems in the field of machine vision. New methods designed to successfully handle these challenges can be extended to numerous other applications with little effort.

1.2 - Automation of Silverware Sorting

Automation of large commercial dishwashing operations is desirable as it offers improved efficiency of operation and reduced labor costs. The operations are repetitive in nature and are to be performed in conditions such as inconvenient temperature, humidity levels and limited leeway, resulting in increased costs for manual labor (Yeri, 2002).

Automating this operation requires the integration of subsystems capable of singulation, identification, inspection and sorting, followed by wrapping of silverware. A singulation

system should be capable separating a mixed batch of silverware into non-touching, non-overlapping properly oriented positions (Hashimoto, 1995; Latvala, 1999). The identification system should differentiate silverware pieces and also acquire spatial orientation information (Yeri, 2002). The inspection system should discern between clean and dirty silverware pieces. The sorting system should segregate the silverware pieces into collection bins, and the wrapping system should wrap appropriate pieces of silverware into a paper or cloth napkin for next use. All these subsystems are then to be integrated into a single system that takes a mixed batch of silverware as input and gives clean wrapped silverware sets as output.

A search through United States Patents (Patent Class 209/926* - Silverware sorter subclass) reveals that there exist sorting mechanisms that utilize properties of silverware pieces, such as dimensions and location of center of gravity of silverware to sort and orient silverware pieces. Templates act as sieves to sort silverware pieces when the aforementioned properties are used as the criterion for sorting. There also exist mechanisms that utilize opto-electronics to identify silverware pieces. Phototransistors and lights are used to obtain an optical reading representative of the contour of the silverware. This reading is used to identify silverware pieces. However, none of the abovementioned mechanisms inspect the pieces for cleanliness, such that human intervention is required when cleanliness of piece is to be determined. Since such sorting mechanisms are dependent on the physical characteristics of the silverware piece, changes in templates or other hardware are necessary whenever silverware sets are changed. It thus appeals that a system having the capabilities of inspecting silverware

pieces and accommodating changes in silverware sets without having to change hardware is desirable. Inspection of silverware pieces requires information about surface features for decision-making. Ability to accommodate different sets of silverware requires the separation of information-acquisition, decision-making, and actuation sub-systems. The information-acquisition sub-system should also be capable of acquiring sufficient information about a silverware piece to enable identification. Visual images or photographs of silverware pieces provide shape and surface information, and hence make a suitable candidate for decision-making input. Hence a machine vision system was deemed suitable for implementing the information-acquisition and decision-making sub-systems.

Since machine vision systems are programmable, a large degree of flexibility can be provided in systems that utilize them. This is a huge advantage, since it eliminates the need for frequent hardware changes to cope with changing inputs or environments. Apart from programmability, vision systems also provide efficiency and repeatability in operations. These systems can be installed in work environments that are disagreeable to humans. Vision systems rarely incorporate moving parts and seldom require maintenance. Hence when appropriately designed, visions systems can compete with, and may surpass, human performance levels.

The current gamut of industrial applications that employ vision systems is vast. Examples of vision applications are found in the pharmaceutical industry, the labeling and packaging industry, the bottling industry, the food industry, and the agricultural industry

(RMA, 2004; IVP, 2004) to name a few. The pharmaceutical industry uses vision systems to detect cracks in tablets (Ukiva, 2004). The packaging and labeling industries use vision applications for rapid processing of compact labels (Ukiva, 2004). The food industry uses vision applications for quality control processes (JZW, 2004). Apart from these, vision applications can be found in various other industries in quality control processes, including manufacturing and assembly operations.

This study is concerned with the development of a vision system capable of identifying and inspecting singulated silverware. Assuming singulation has already occurred, the vision system should approximate the human decision-making capabilities in these operations and send appropriate control signals to the sorting system for further processing.

1.3 – Precedent work on Vision Systems for Silverware Identification

Sandeep Yeri, as a part of his thesis (Yeri, 2002), designed a vision system that included a frame grabber PCI card, camera, lenses, lighting equipment, lighting setup, camera trigger circuitry using optical sensors, and software applications.

The components used in the implementation of Yeri's prototype are as follows:

- Camera: Basler L104/1K (1024 Pixels, Programming capability via RS-232, High Sensitivity, Anti-blooming, High Signal to Noise Ratio, Compact Size).
- Lens: Cosmicar/Pentax Lens (C-Mount, 25mm Focal Length, 1:1.4 aperture ratio, manual focus).

- Frame Grabber: National Instruments PCI-1422 Image Acquisition Board (16 bit gray level Digital Image, 80 Mbytes/Sec Data Transfer Rate, 16MB On-Board Memory).
- Lighting Equipment: Regent Lighting Corp. Quartz Halogen Lamps Model MCL (30W, 120V, 60Hz, 0.29Amps).

A software application for the prototype system was developed in Microsoft Visual C++ 6.0. The software application used NI-IMAQ Vision Software Libraries (Device Drivers, Hardware & Software Interfacing software). This hardware and software was used in the study herein. The software application developed by Yeri was capable of extracting identification and orientation information about silverware from the acquired image. The algorithm used “Blob Analysis” to ascertain this information. “Blob” is a short term for Binary Large Object. A blob, also called “particle”, refers to a white region inside a binary image (black and white image). In blob analysis, properties such as area, perimeter, and moment of inertia are defined for the white regions, and examined in order to characterize the regions under examination. These binary images are obtained when grayscale images are subject to a thresholding process. A grayscale image is an image in which various shades of gray between white and black may be present. Typically, monochrome cameras provide 256 shades of gray. Thresholding is the process of segmenting an image into white and black regions. This is a lossy process, and some information about the image is irrecoverably lost during the process. Yeri’s system acquired grayscale images of silverware pieces, converted them to binary images, and processed the binary images, for identification purposes.

The software application developed by Yeri identifies silverware pieces based on area of white regions representing the silverware piece. Though this is a good metric that provides a fair amount of information about the silverware piece, it does not provide any information about shape of the object. Hence, an identification mechanism using area alone will fail in cases where areas of two different pieces of silverware are the same, even though their shapes are significantly different. Thus, including other features such as moments of inertia and perimeter, which convey some information about shape, will improve the reliability of an identification procedure. The application developed by Yeri also does not inspect pieces for cleanliness. Accordingly, we seek a vision system that can perform inspection, as well as identification with improved reliability.

1.4 –Thesis Objective

The objective of this thesis is to design and construct a vision system capable of identifying silverware pieces with high accuracy, inspecting them for cleanliness and sending appropriate signals to a sorting mechanism, to be designed by others (Peddi, 2005). The minimum throughput target for this system is 30 pieces of silverware per minute. The current manual-processing rate is around 2400 pieces of silverware in a two-hour shift, which is equivalent to 20 pieces of silverware a minute. This thesis is aimed at developing new techniques to handle the challenges associated with silverware identification and inspection mentioned earlier. This thesis also aims to develop these identification and inspection techniques in such a way that these techniques can be ported to other areas of application with minimal modifications.

Identification requires less effort, and is conceptually easier, than inspection. Efficiency of inspection varies according to color, size, and location of dirt on a silverware piece. It is difficult to inspect when the dirt color blends with the color of the silverware piece, when the dirt particle is small in size, and when the dirt particle is located close to the edges of the silverware piece. Identification on the other hand is a much easier task to perform since it is dependent only on the size and shape of the silverware. Hence the accuracy of identification will typically be higher than the accuracy of inspection, whether done manually or automatically with a machine vision system. For manual labor, because of the monotony of the task, efficiencies of both processes deteriorate with time. In manual inspection, accuracy of inspection declines faster than accuracy of identification.

This work intends to design and construct a vision system, which has improved identification accuracy compared to Yeri's prototype, and to develop an initial approach for inspecting pieces of silverware. It is anticipated that the inspection system will output certain false "clean" and false "dirty" results. A false "clean" result refers to a dirty silverware piece that is wrongly classified as clean, and a false "dirty" result refers to a clean silverware piece that is wrongly classified as dirty. It is assumed during the design of this system, that a false "clean" result is less preferable than a false "dirty" result, because a false clean is processed through to final clean storage, whereas a false dirty is merely recycled through the singulation, identification, and inspection stages of the overall system.

Though currently there exist many vision system applications capable of identification and inspection, the current problem poses certain unique challenges. The objects under consideration are metallic pieces, which have surfaces with complex geometry and a shiny finish. Such surface conditions give rise to a phenomenon known as specular reflection. Specular reflection is defined as a sharply defined beam resulting from reflection off a smooth surface (EWWOP, 2004). Specular reflections result in "glares" being formed in the image and yield a poor image of the object, which in turn results in incorrect image processing. Designing a lighting setup to avoid specular reflections usually results in non-uniform lighting, which again typically results in incorrect processing of the image. A lighting setup that minimizes specular reflections is dependent upon the geometry of the surface, and hence it is problematic to design a single setup that avoids specular reflections for various pieces of silverware. Most vision systems currently used for inspection purposes perform rudimentary analyses, since they only check for the presence or absence of a feature, without comparison with other images. On the other hand, our problem requires comparison of test images with images of clean pieces of silverware in order to make decisions about the cleanliness of a piece. An enlarged image yields greater detail for decision-making, but needs more time for processing. A reduced image can be processed quickly, but might not yield all the details necessary for decision-making.

Finally, because of the required throughput, the vision system has limited time for the identification and inspection processes, which limits the sophistication and rigor of the

system. Since all these conditions co-exist, the complexity of this problem is somewhat acute. To handle this, in Chapter 2, we address the design of the hardware, including lighting setup, optics, and selection of the camera and frame grabber PCI card. Chapter 3 presents the design of image processing algorithms for identification and inspection. Experimental results and analysis of them are given in Chapter 4, and conclusions and recommendations are presented in Chapter 5.

Chapter 2: Vision Setup

2.1 – Problems identified with existing setup

Inspection of silverware pieces requires the image acquisition system to deliver images of high quality. The vision setup used for Yeri's experiments (Yeri, 2002) was found to be delivering images that were sometimes unusable for inspection purposes.

The images acquired by Yeri's system revealed the following undesirable characteristics:

- Distortion in shape of silverware piece (Figure 1).
- Noise, in the form of lines, parallel to image scanning direction (Figure 2).
- Shadows and ill-lighted regions on the silverware piece (Figure 3).

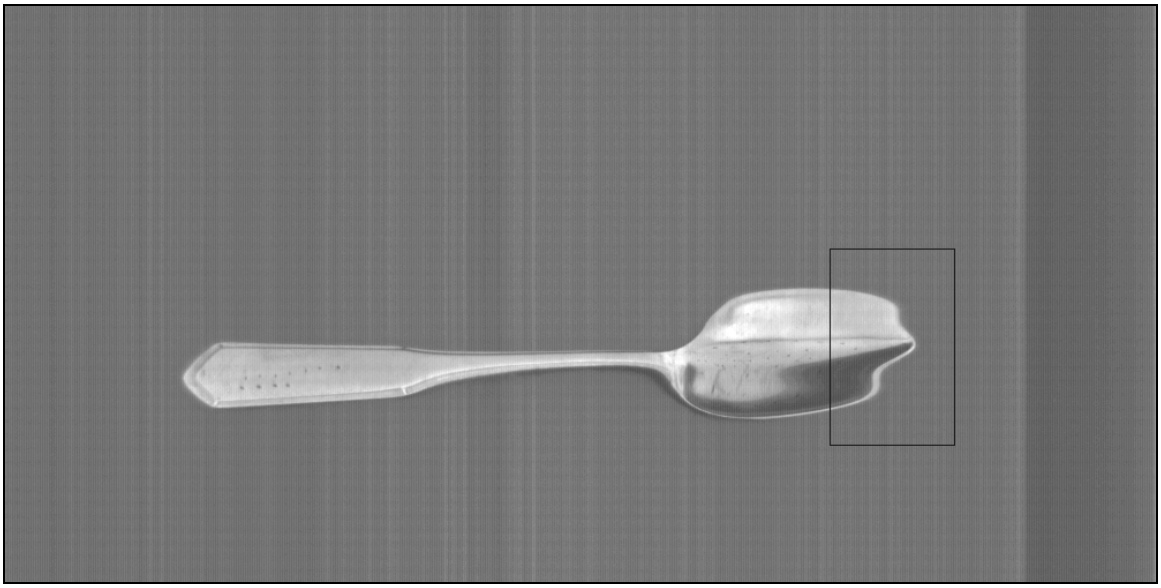


Figure 1: Distortion of Shape of Silverware Piece in Acquired Images.

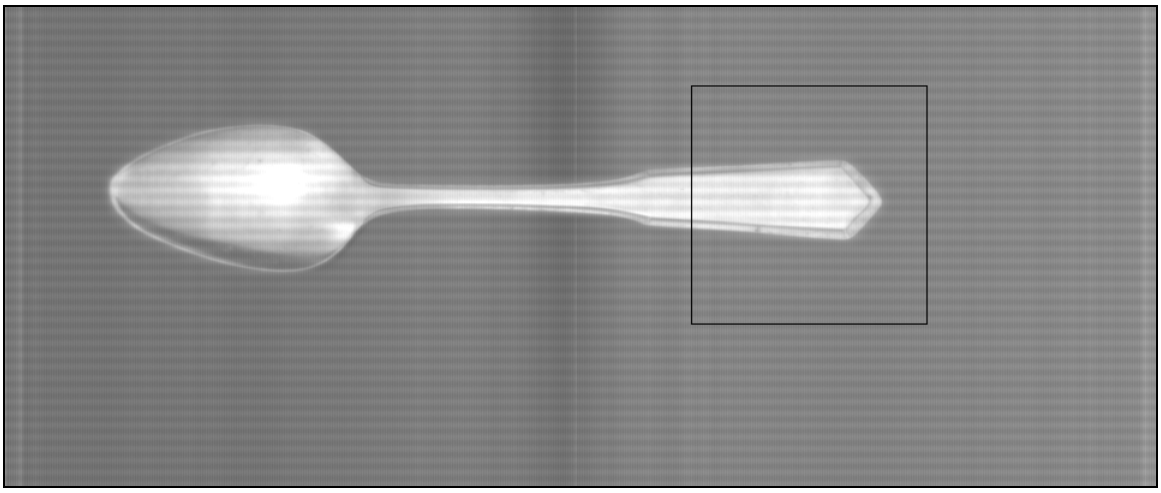


Figure 2: Noise in the Form of Lines Parallel to Image Scanning Direction.

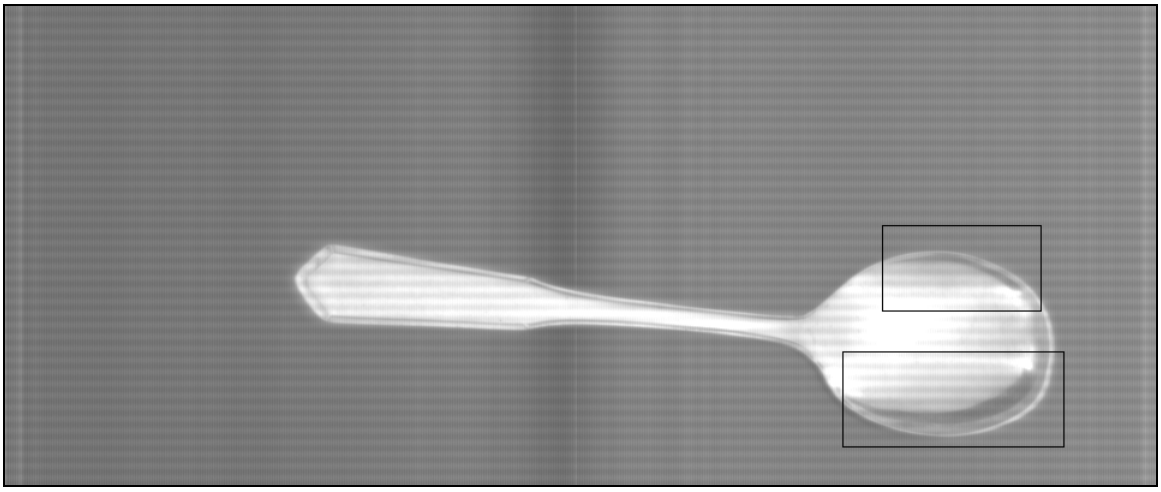


Figure 3: Ill-lighted Regions on Silverware.

The distortion of shape of silverware pieces in images was found to occur because of vibration of the silverware pieces as they were conveyed in a sliding mode underneath the camera. For low conveying speeds, the effects were small, but at higher speeds the images exhibited distortion levels that were unacceptable. The remedial measure for this was to modify the conveying mechanism by removing the sliding mode and fixing silverware pieces to a moving magnet. This eliminated the vibration of the silverware pieces as they were conveyed underneath the camera. Ravi V Peddi, as a part of his thesis (Peddi, 2005), implemented this modification to the conveying mechanism.

In an attempt to locate the source of the noise lines in the images, various light sources were used to study their influences on the lines. The noise lines were found to have frequencies, at various times, of approximately 120 Hz and 150 Hz. Halogen lamps, incandescent lamps, and fluorescent tube lights, all powered by 110V AC building power, together with “torch lights” powered by DC batteries were used as light sources in various experiments in an attempt to identify the source of the noise lines. The noise lines, when present, had a frequency independent of the light source. The noise lines persisted in the first three cases but were found to be absent when a battery light was used for illumination. We suspected that noise in the ATRC Lab Building AC power supply was the problem. Lamps, rectifier circuits and AC isolator circuits were employed, along with halogen lamps, incandescent lamps and tube lights, but the noise lines were found to persist. An oscilloscope connected to the lab power supply verified the presence of 120Hz and 150Hz noise in the 110V 60Hz building power supply. DC Lamps were then chosen for illumination purposes, powered by filtered DC obtained from a Switch Mode Power Supply (SMPS) fed by 110V 60Hz building power supply, commonly used to power PCs. Images acquired using this combination of lights and power supply did not contain any noise lines.

The ill-lighted regions on silverware pieces are sometimes caused because of the complex surface geometry of the pieces. The lighting equipment should be capable of providing diffused light onto the silverware from multiple directions to minimize ill-lighted regions. But this requirement would force a lighting arrangement to be specific to the physical shape of the silverware being imaged. The lighting setup requirements for elimination of

shadows and the requirements for elimination of specular reflections are mutually conflicting for silverware pieces with curved surfaces. Thus, it is difficult to prescribe a lighting setup that provides ideal lighting conditions for different silverware pieces having different surface geometries. However, among the well-established lighting techniques, the "Direct Front Illumination" (Figure 4) and "Light Tent" (Figure 5) lighting setups (MVA, 2004) appear to offer the best compromise for the system under consideration. The lighting setup used for Yeri's experiments (Yeri, 2002) is an adapted version of the "Light Tent" setup. Making slight modifications to Yeri's setup reduced the ill-lighted regions. The exact modifications introduced into the lighting system will be described in Section 2.3.

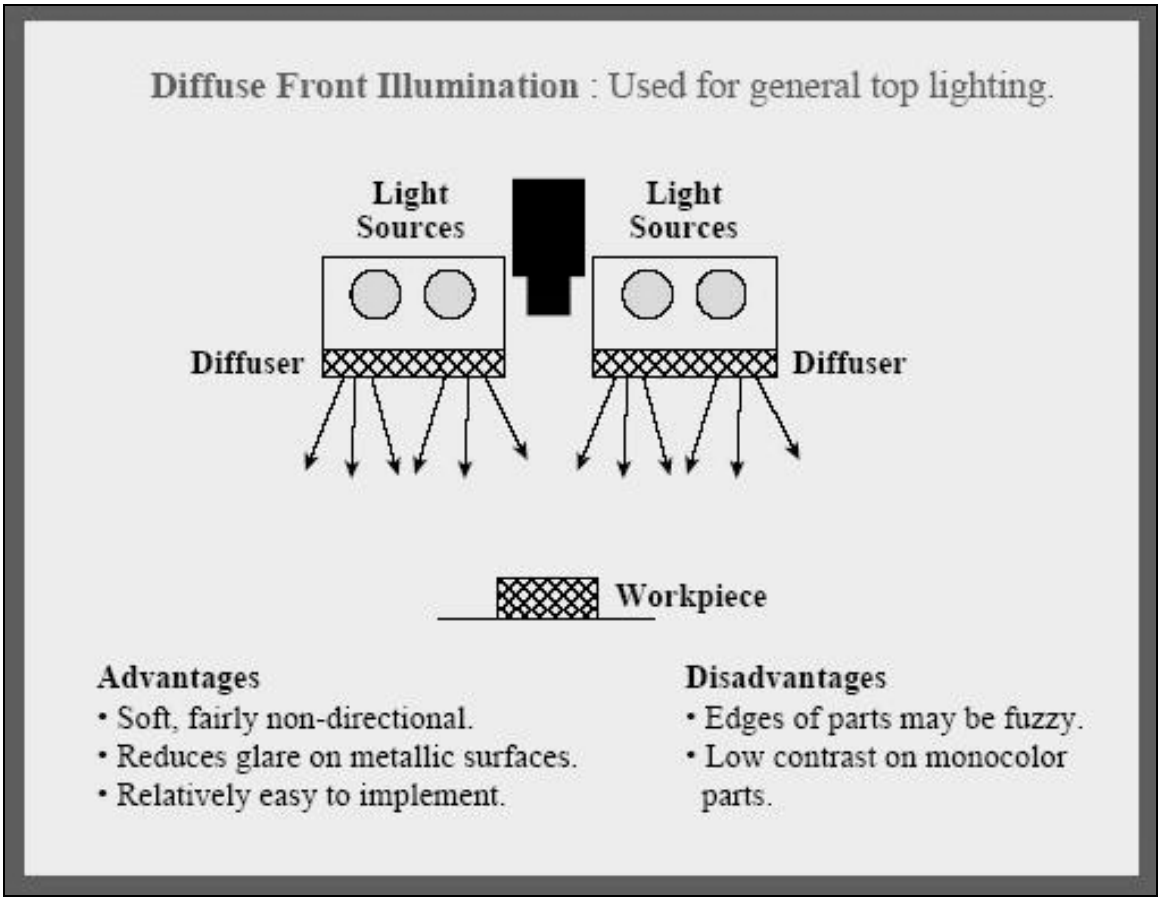


Figure 4: Diffuse Front Illumination
(Machine Vision Association: http://www.sme.org/downloads/mva/mvaposter.pdf)

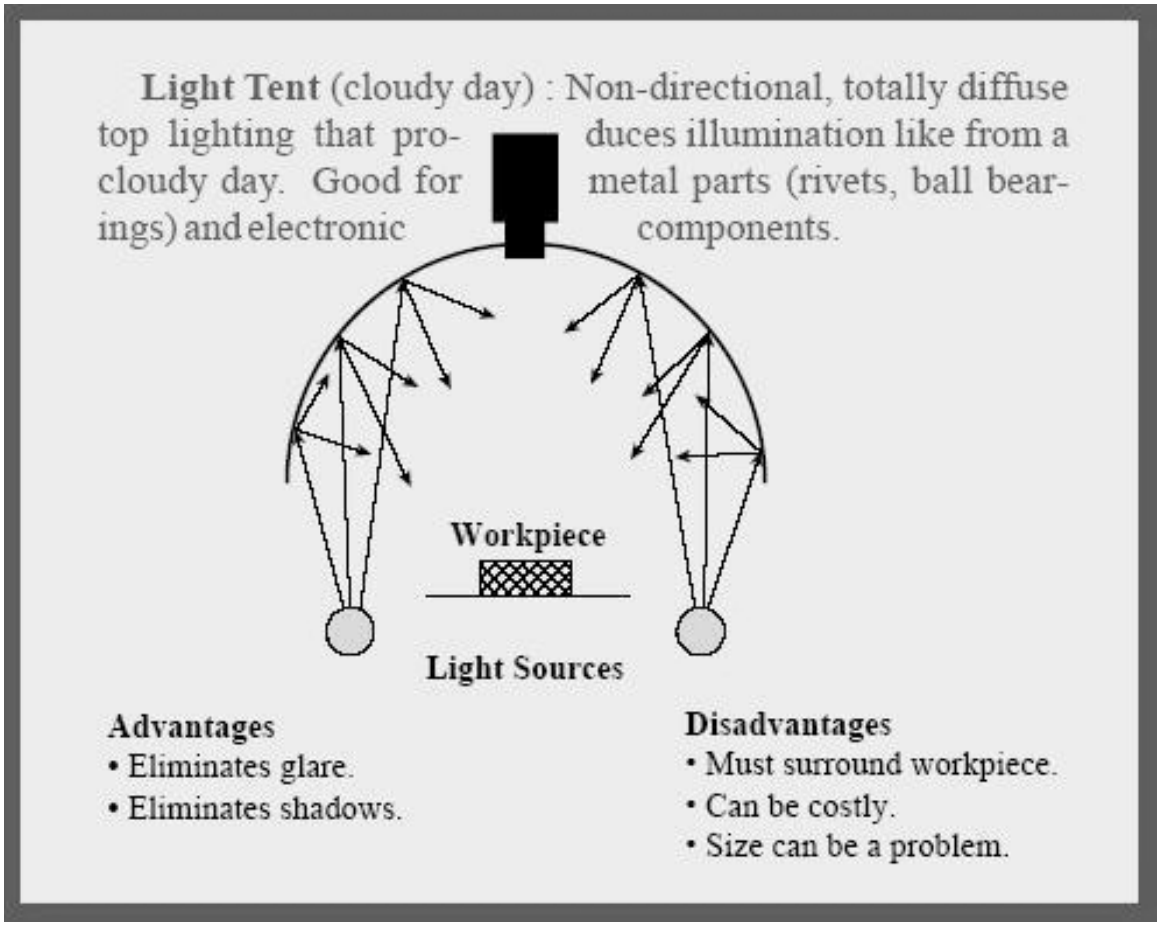


Figure 5: Light Tent Illumination
(Machine Vision Association: http://www.sme.org/downloads/mva/mvaposter.pdf)

2.2 – Camera & Board

The image acquisition system used for experimentation during the course of this thesis uses a PCI-1422 Image Acquisition Board and a Basler L104/1K camera for imaging purposes. These were originally acquired and implemented by Yeri (Yeri, 2002). The Basler L104/1K camera is a monochrome line scan camera capable of providing images of 256 gray levels with a maximum width of 1024 pixels. It offers features including RS-232 programming capability, high signal to noise ratio, electronically controllable sensor exposure time, electronic trigger capability, and a maximum scan-line rate of 57.45KHz. The camera's sensor has a principal spectral response from 300nm to 1000nm, peaking at

approximately 700nm, as shown in Figure 6. The DC Lamps chosen for illumination emit light of wavelength ranging between 550nm to 650nm, generating good spectral response from the camera.

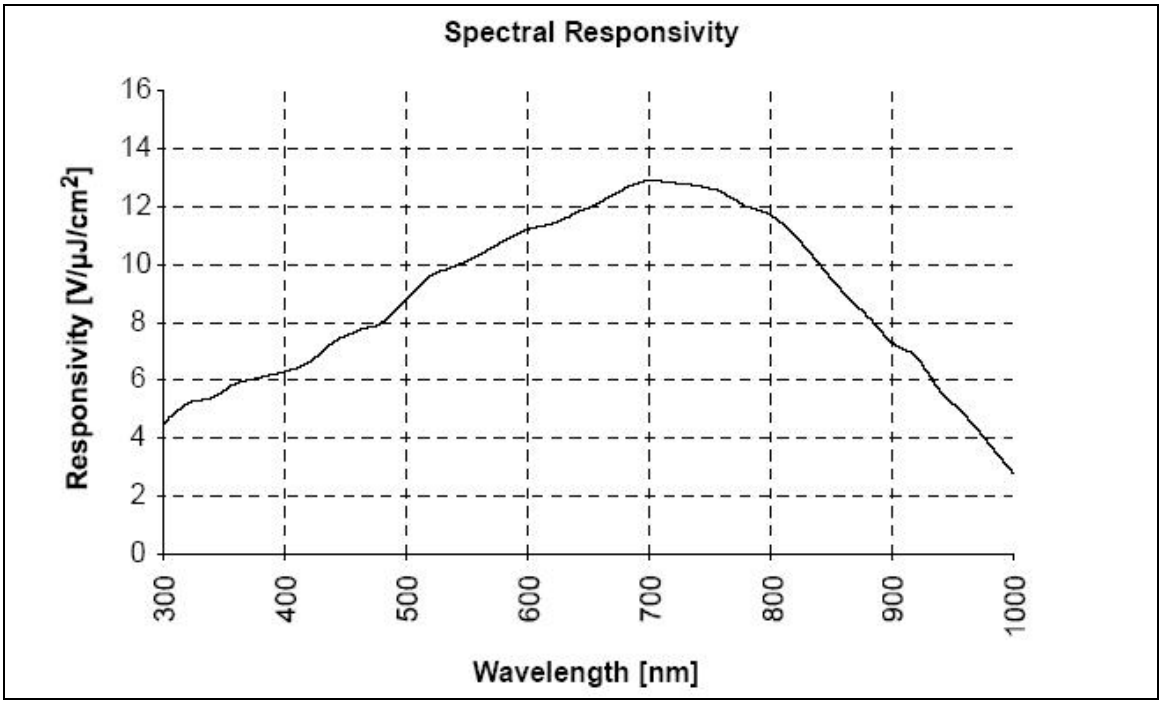


Figure 6: Spectral Response of BaslerL104/1K Camera.
(Basler Corporation: http://www.baslerweb.com/popups/403/L100_Users_Manual.pdf)

The PCI-1422 Image Acquisition Board, provided by National Instruments, was chosen because apart from being compatible with the camera, National Instruments provides good software support in the form of pre-compiled libraries that can be used in the development of custom applications. National Instruments provides extensive software routines that facilitate programmable image acquisition and image processing. The camera and frame grabber combination mentioned above, when employed along with an appropriate lighting setup, is capable of delivering images meeting the quality requirements for identification and inspection of silverware pieces.

2.3 – Lighting Setup & Optics

During the imaging process, the lighting setup illuminates the object to be imaged, and the optical equipment focuses light from the object onto the camera sensor, which registers an impression of the object. The criteria for selection of optical equipment include resolution, sensor size, field of view, working distance and depth of field, illustrated in Figure 7.

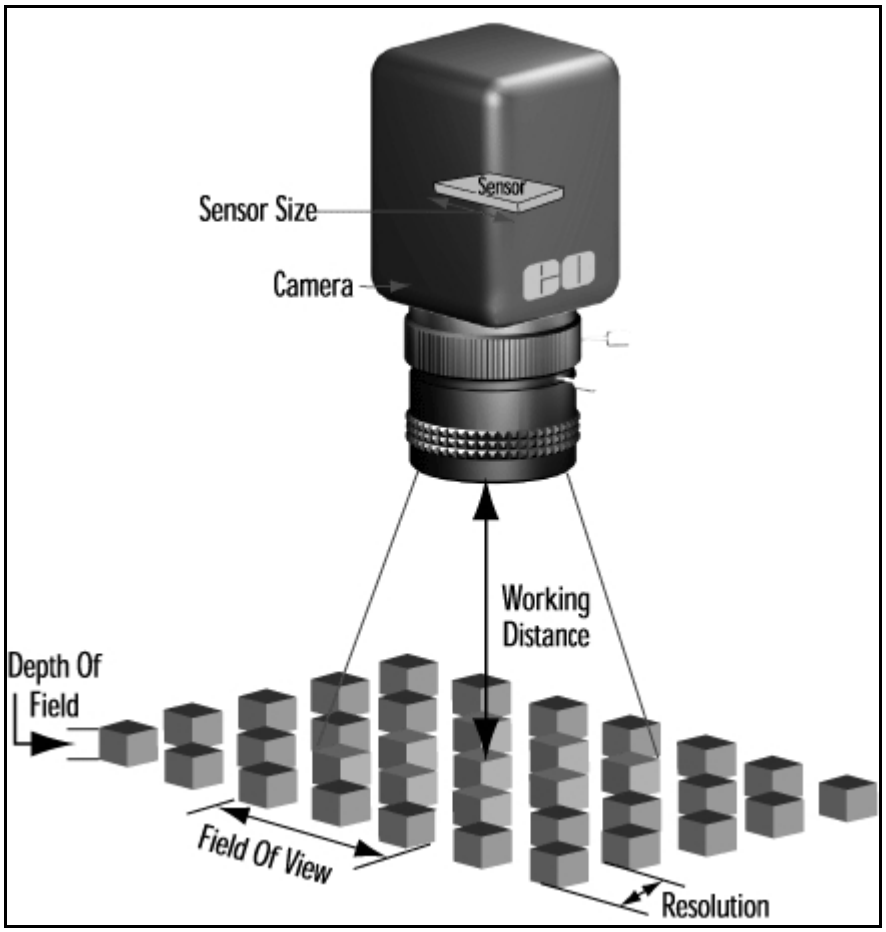


Figure 7: Fundamental Parameters of an Imaging System (EIO, 2004).
(Edmund Industrial Optics: http://www.edmundoptics.com/techSupport/DisplayArticle.cfm?articleid=287)

A C-mount lens, of 25mm focal length and an aperture ratio of 1:1.4, was selected since could accommodate the 1024 pixels on the camera sensor of 10.24mm width (Yeri,

2002). Selecting a maximum field of view of 12 inches, we obtain the working distance of 24.5 inches using (1):

$$W = d_o \frac{w}{f} - w \qquad (1)$$

where W is the width of object being imaged, f is focal length of lens used, d_o is working distance, and w is the width of sensor.

The lighting setup for illuminating the object in this thesis is a modified version of the lighting setup used by Yeri for his experiments (Yeri, 2002), shown in Figure 8, which is a modified "light tent" setup.

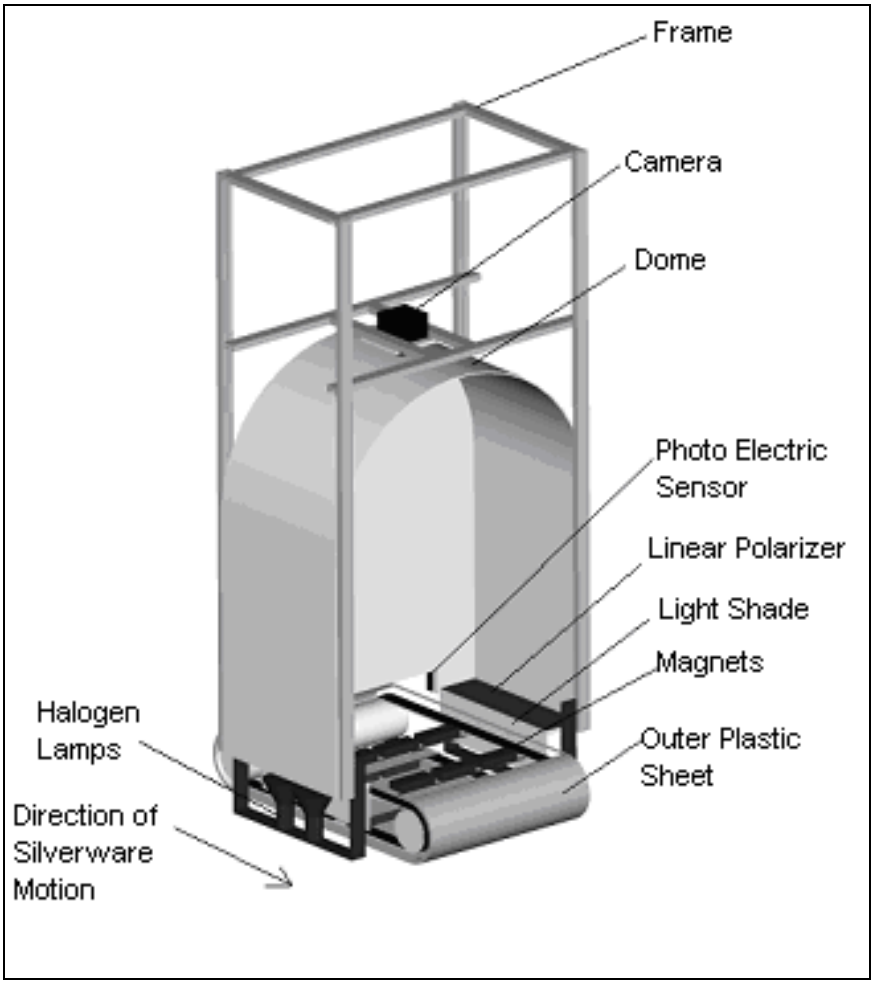


Figure 8: Illumination Setup used by Yeri (Yeri, 2002).

Our modifications to this setup consist of adding “curtains” that serve as extended reflecting surfaces in order to reduce the ill-lighted regions. Figure 9 shows the lighting setups with and without curtains. Yeri used polarizers to control the intensity of illumination. The DC Lamps mentioned previously are of a lower intensity than the halogen lamps used by Yeri, and hence the polarizers have been removed to allow more light into the lighting dome.

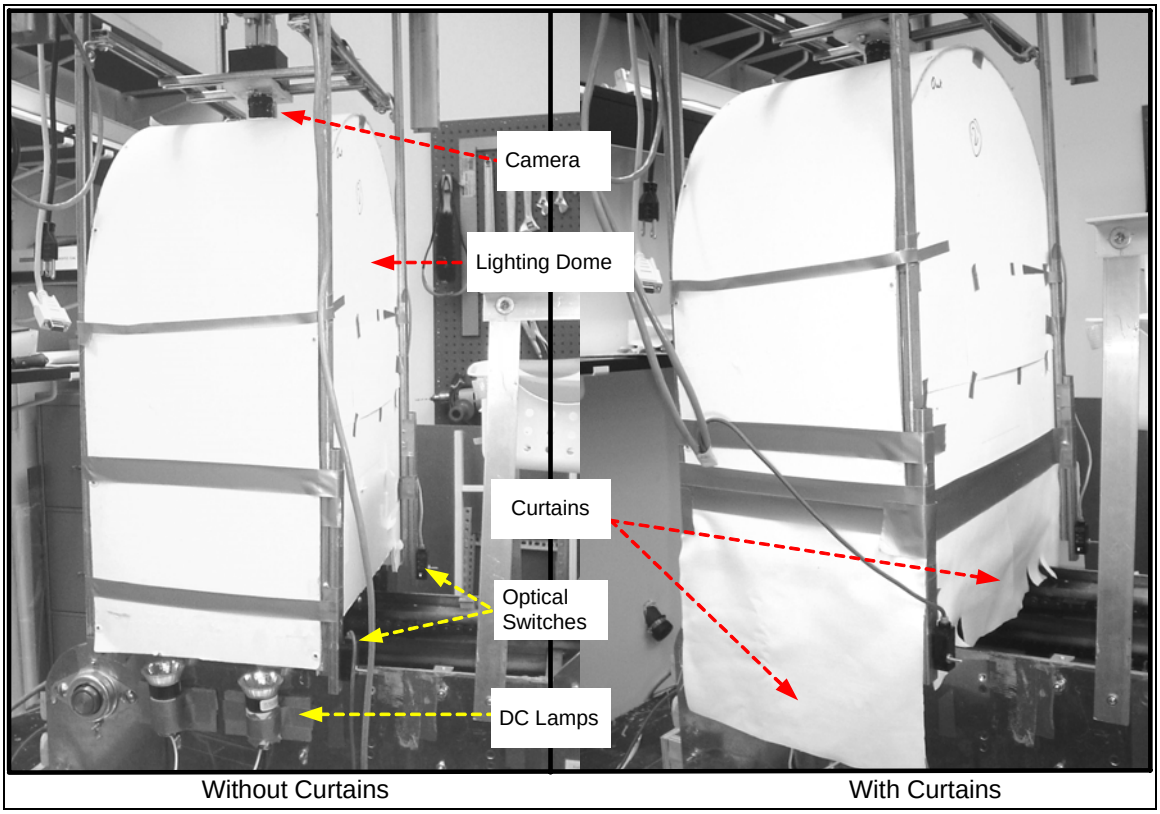


Figure 9: Lighting Setup With and Without Curtains.

Images acquired using this lighting setup showed significant reduction in the ill-lighted regions, and also do not contain distorted shapes or noise lines. This indicates more suitability for inspection of silverware pieces. Figure 10 shows a sample image acquired using this lighting setup. It can be seen that significant improvement has been achieved over the previous images acquired by Yeri’s setup shown, in Figures 1,2, and 3.

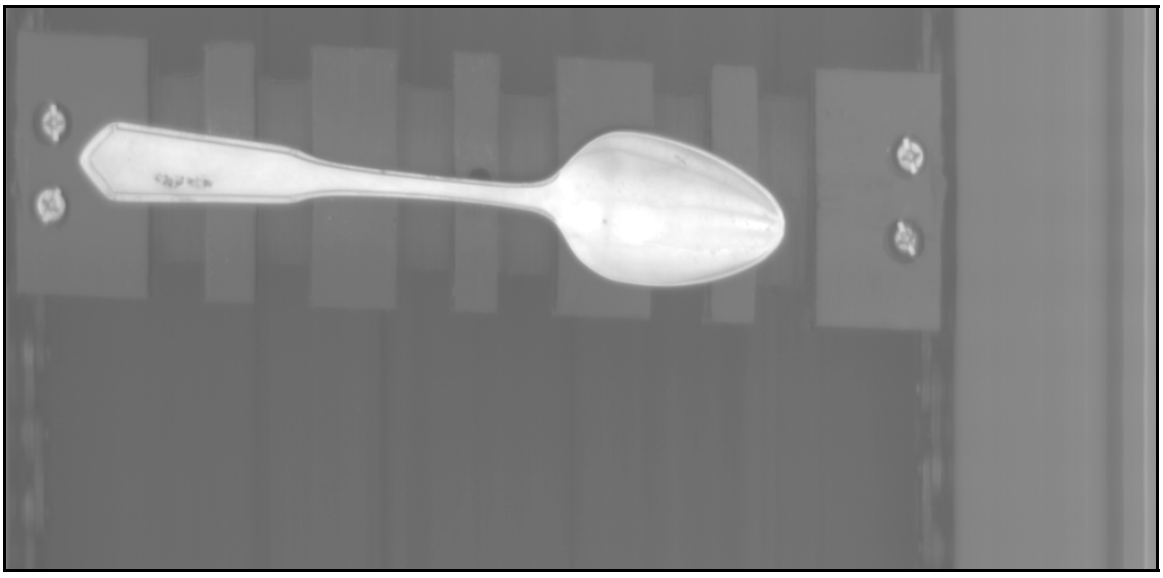


Figure 10: Sample Image Acquired Using Lighting Setup with Curtains.

Chapter 3 will present the image processing algorithms for identification and inspection of silverware pieces acquired using the lighting setup described above.

Chapter 3: Algorithm Design & Description

3.1 - Digital Images & Fundamental Image Processing Operations

A digital image is a two-dimensional array of values representing light intensity of a given picture element (pixel). They are collections of values f(x,y), where 'f' is a function of brightness at the pixel at spatial co-ordinates (x,y) (NIVCM, 2004). The images used for this thesis are 8-bit grayscale images, meaning that f(x,y) is an integer in the range [0,255] representing black(0), white(255), and various gray levels in between, as shown in Figure 11.

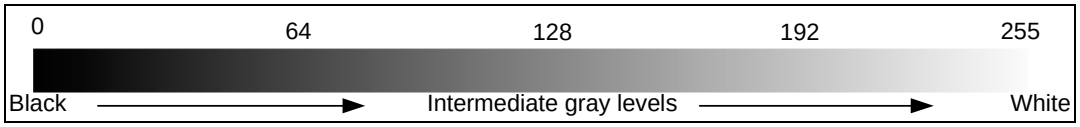


Figure 11: Gray Scale Reference Chart

By convention, the location of the image origin is at the left-top corner of a 2-D scale. The X co-ordinate increases from left to right and the Y co-ordinate increases from top to bottom, as shown in Figure 12.

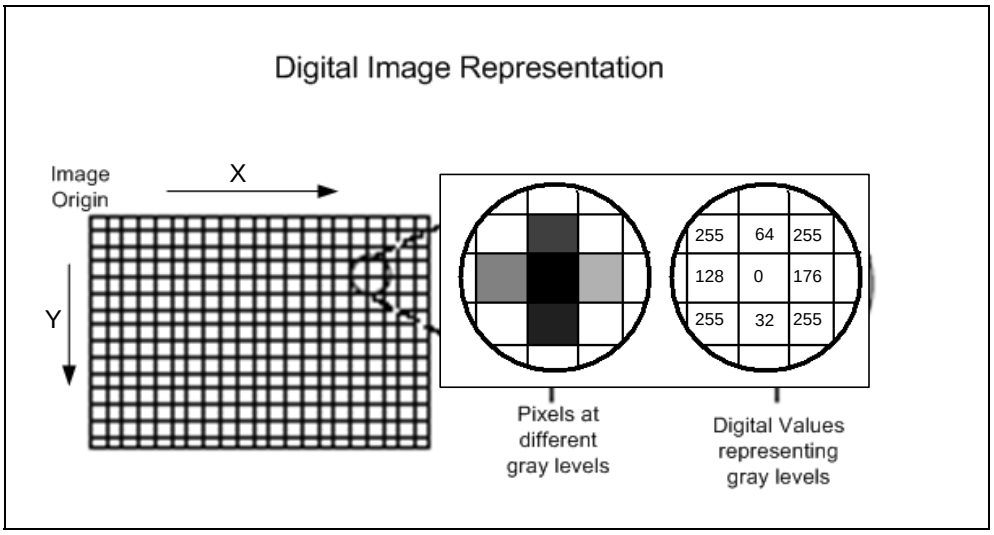


Figure 12: Digital Image Representation.
(Machine Vision Association: http://www.sme.org/downloads/mva/mvaposter.pdf)

The actual internal representation of a digital image also includes image borders. Image borders are value cells attached to all sides of the actual digital image. Most image processing algorithms require neighboring pixel values to evaluate any given pixel. Since pixels on the edges of the images do not have neighbors on all four sides, image borders are added to images to act as surrogate neighbors, thus facilitating the processing of edge pixels of the images. The border size for all images acquired and processed during the course of this thesis is 5 pixels wide. Figure 13 illustrates the internal image representation used by the libraries provided by National Instruments (NI).

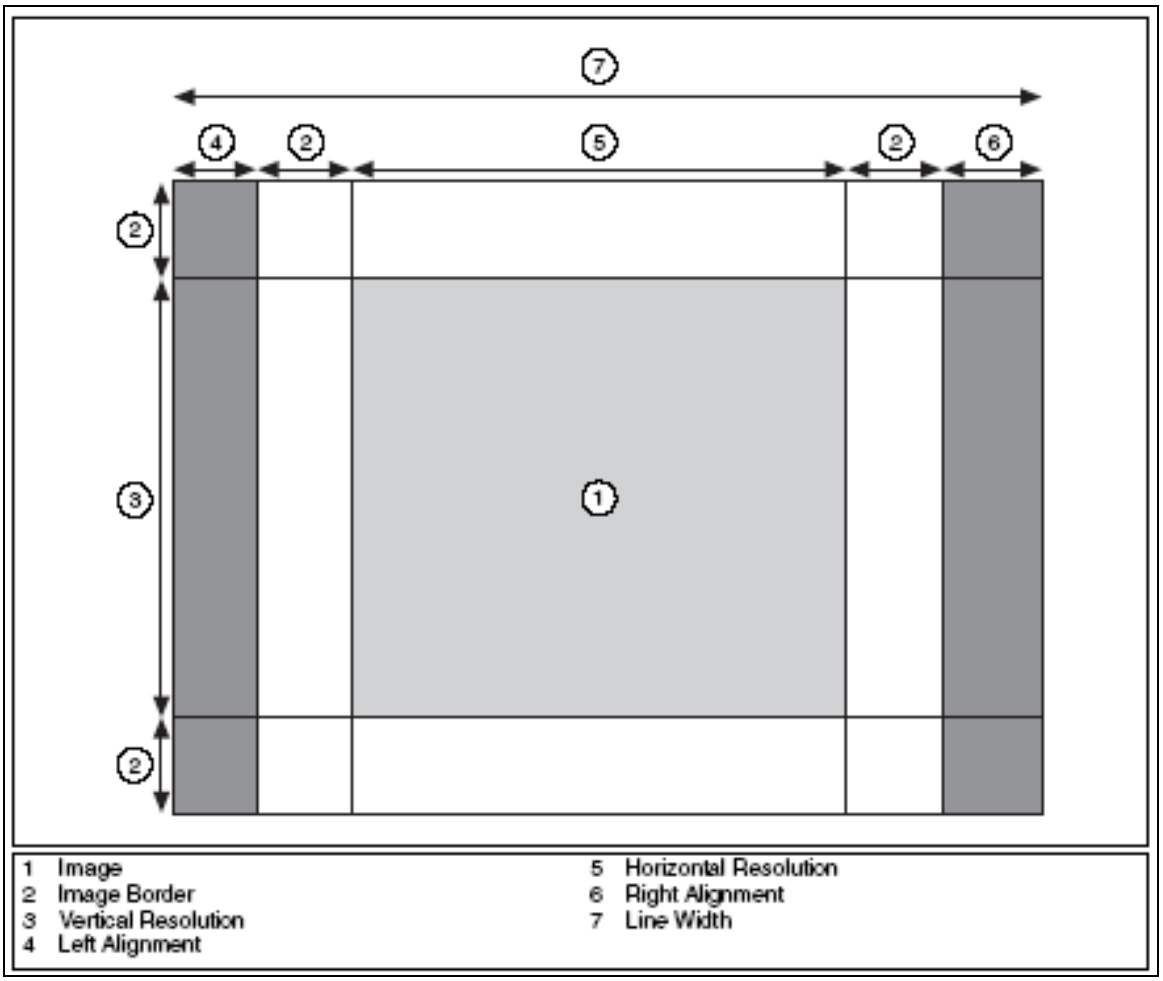


Figure 13: Internal Image Representation in National Instruments Libraries.
(National Instruments: http://www.ni.com/pdf/manuals/372916c.pdf)

NI provides a variety of image processing functions to analyze and manipulate images. It is outside the scope of this thesis to provide a detailed description all the functions, but we provide a brief description of the frequently employed functions to assist understanding.

Thresholding is the process of separating the “foreground” and the “background” of an image. This operation is performed by setting the values of all pixels within a range to one of the two values, typically 1 or 255, and the value of all other pixels to another

typically 0. NI provides algorithms for thresholding, auto-thresholding and multi-thresolding. Thresholding effects are illustrated in Figure 14.

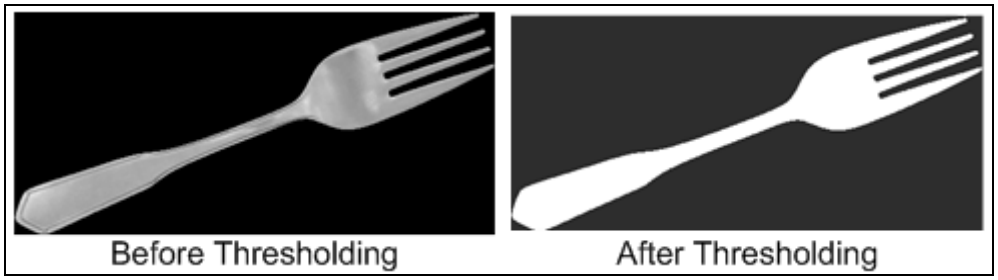


Figure 14: Thresholding of Images.

Edge-detection is the process of locating the edges in an image. An edge is usually defined as a region of sharp changes in gray levels. Edges are detected by employing a convolution operation followed by a thresholding operation. Edge-detection results are illustrated in Figure 15.

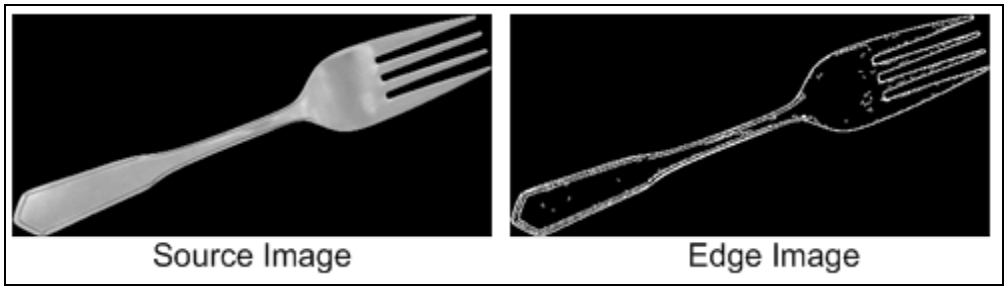


Figure 15: Edge-detection in Images.

Image rotation algorithms are also provided by the NI libraries, which rotate images about their center using zero-order or bilinear interpolation. Image rotation functions are used for alignment of images during inspection in this thesis. Rotation effects are illustrated in Figure 16.

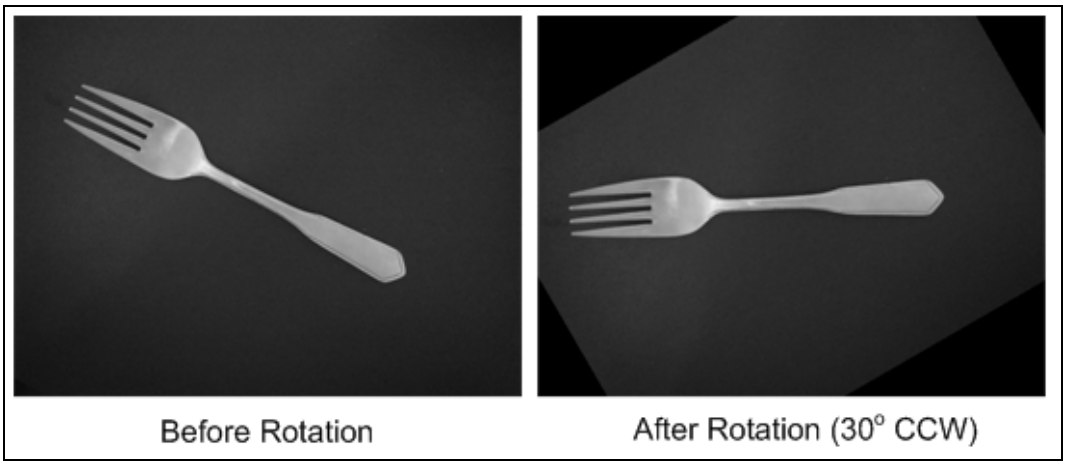


Figure 16: Rotation of Images.

Image re-sampling refers to the process of resizing images employing certain methods of interpolation. Image re-sampling functions are used in this work for accommodating slight changes in sizes of silverware pieces during alignment for inspection. Re-sampling results are illustrated in Figure 17.

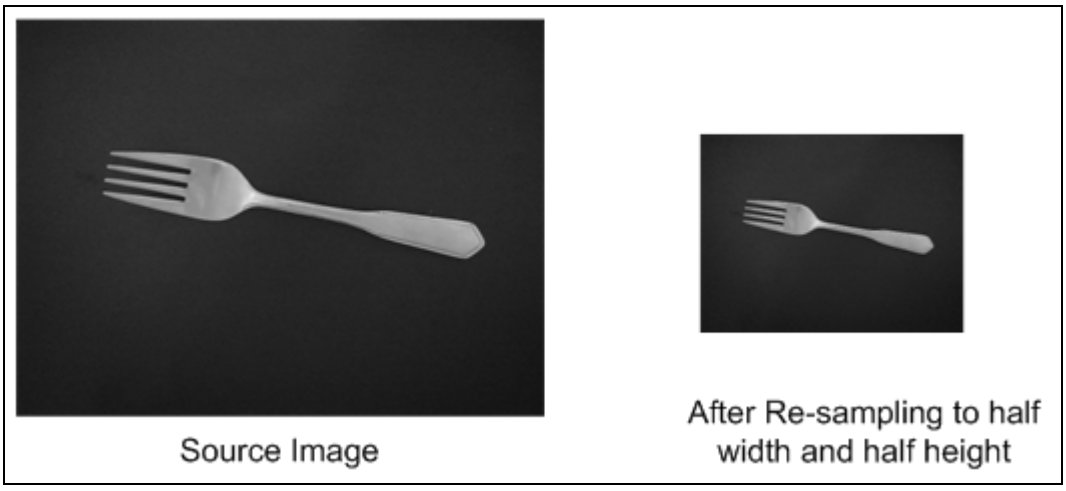


Figure 17: Re-sampling in Images.

NI libraries also provide extensive functions for binary and grayscale morphology, and particle measurements that are used extensively in this work. These functions provide methods to perform various analyses and to manipulate images, if so desired.

3.2 – Image Pre-processing

The image acquired from the camera includes features other than the silverware piece. Hence certain pre-processing has to be done before the image can be passed on for identification and inspection processes. Pre-processing expunges unnecessary features from the image and improves the quality of the data left in the image passed on for inspection and identification. Figure 18 demonstrates images before and after pre-processing.

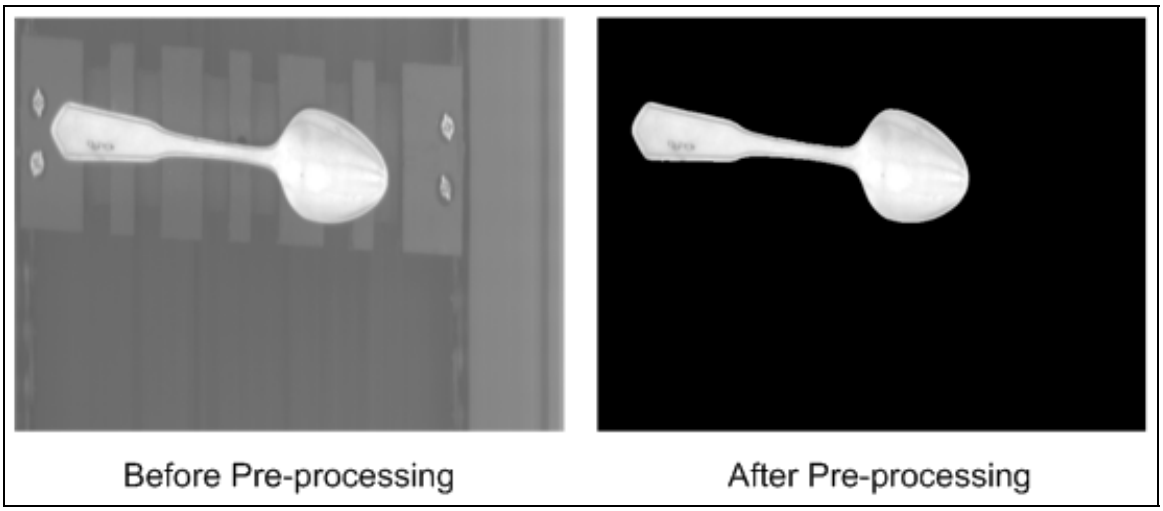


Figure 18: Images Before and After Pre-processing.

In this work, the image pre-processing algorithm includes thresholding, selection of the largest particle, rejection of all other particles, construction of a mask using the selected particles, followed by masking of the original image to suppress all other unnecessary information. A mask refers to a binary image that is the same size or smaller than the image being masked. Masking isolates parts of an image for processing and is used when processing of the image is to be applied to particular parts of the image. Only those pixels that correspond to non-zero values in the binary image are processed in the image being

masked. Figure 19 shows image before and after masking operation and the binary mask used in the masking operation.

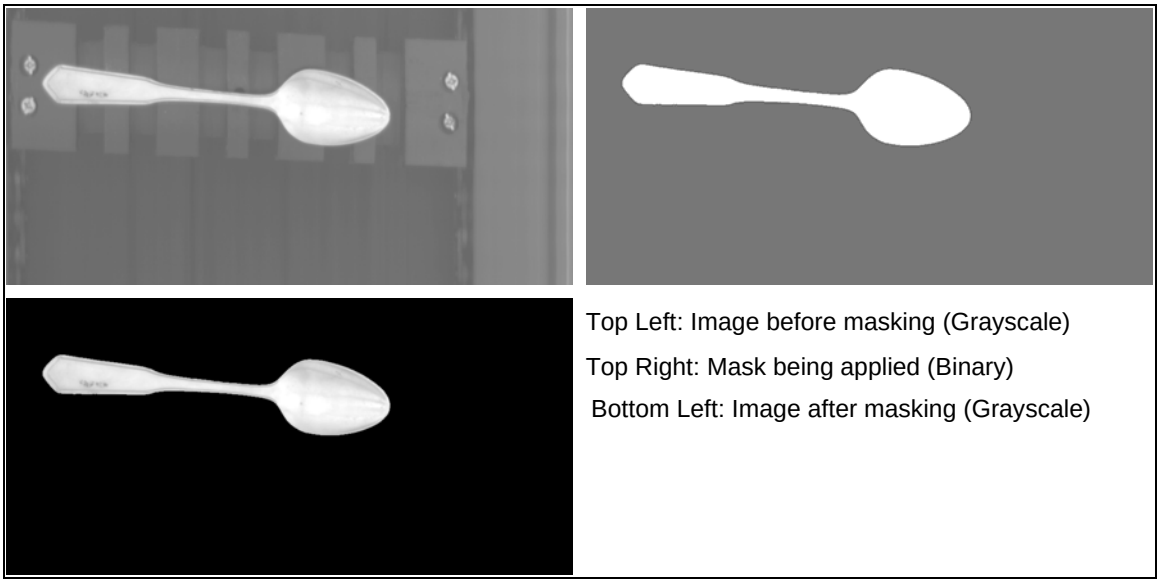


Figure 19: Images Before and After Masking and Image of Mask.

The image pre-processing algorithm utilizes the NI Library functions as image processing primitives (i.e. basic operations). A detailed flowchart representing the pre-processing algorithm and code developed for the algorithm can be seen in Appendix A.

3.3 – Identification

For identifying silverware pieces, we use three features, namely, the area of the largest particle, the area moment of inertia of the largest particle about an axis perpendicular to the image plane (z-axis), and the perimeter of the largest particle. The moment of inertia about the z-axis is evaluated from moments of inertia about the x and y-axes using the Perpendicular Axis Theorem. The area, moments of inertia about the x- and y-axes, and the perimeter are computed using NI library functions. These values are evaluated using binary images obtained by thresholding pre-processed images. Use of the moment of

inertia and perimeter introduces shape-dependence into the identification algorithm, which improves the reliability over identification algorithms based only on area. The feature set, once evaluated for an image to be processed, is compared to the feature sets of pre-learned and pre-loaded binary prototype images. When feature set differences are within a user-defined tolerance, a match is confirmed. A heuristic is added to the identification algorithm that compares the sum of feature set differences to a preset value, and tries to “soften” the “hard limit” nature of comparison of feature set differences the user-defined tolerance. A detailed flowchart representing the identification algorithm and code developed for the algorithm can be seen in Appendix A.

Failure to confirm matches to any of the pre-learned prototypes results in abortion of all further processing for inspection, and appropriate signals are delivered to the sorting mechanism so that the silverware piece is reprocessed.

If a silverware piece is successfully identified, then further analysis is performed to evaluate the angle at which the silverware piece is oriented with respect to the x-axis, and to determine symmetry of silverware piece along its longest axis. This information is needed to align the image to be inspected with the appropriate prototype image for further analysis. It is assumed here that the largest dimension of the silverware piece lies along the handle of the silverware piece. It is also assumed that symmetry, if present, will have an axis of symmetry along the longest dimension (LD).

NI Libraries provide functions to evaluate the orientation of the LD, but further information about the direction in which the head of the silverware piece is pointing is also required to properly align the test image with the prototype image.

A non-symmetrical distribution of area prevails in some silverware images, and this causes the centroid of the area to be closer to one end of the silverware piece than the other. Due to this property, when a line equal in length to the length of the silverware piece is centered about the centroid, one end of the line lies outside the rectangular bounding box, which encloses the silverware piece.

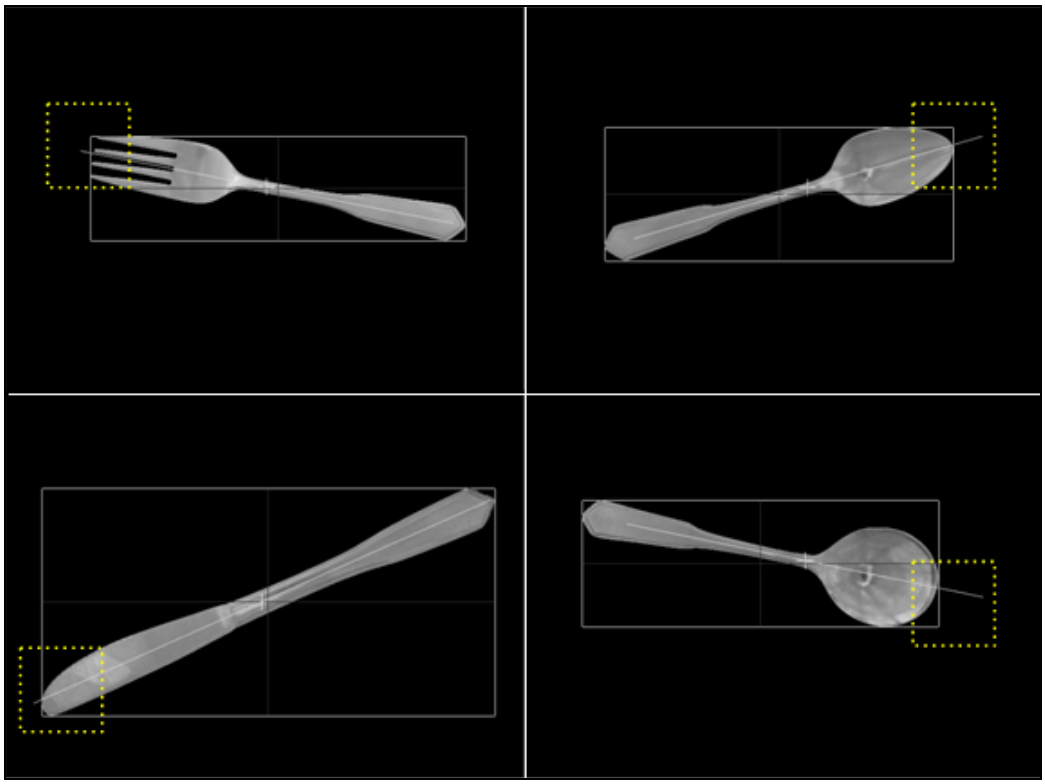


Figure 20: Determination of Orientation of Silverware Pieces.

This property is utilized to evaluate information about the direction in which the head of the silverware piece points. Figure 20 illustrates the aforementioned property, with the dotted box indicating the line segment lying outside the enclosing rectangle.

In order to verify symmetry in a silverware piece, the image is rotated so that the LD is oriented along the x-axis. The locations of the centroid of the test image, and the centroid of the test image reflected about a horizontal line passing through the center of the image, are evaluated and compared to verify symmetry in the image. If the distance between the centroids is within a pre-defined tolerance, the image is assumed to be symmetric about its LD; otherwise the silverware piece is assumed to be non-symmetric about its LD. Figure 21 illustrates verification of symmetry in a soupspoon and a knife.

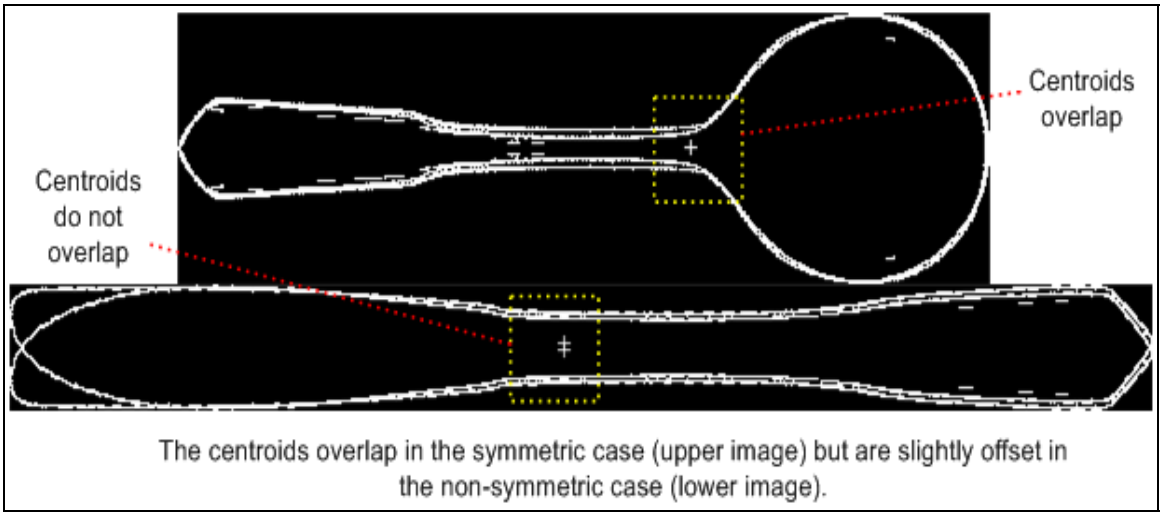


Figure 21: Verification of Symmetry in Silverware Pieces – Edge Image.

For non-symmetric silverware pieces, proximity between centroids is used as the selection criterion to select the appropriate prototype image for inspection process. Any discrepancy between symmetry of test image and prototype will result in abortion of all further processing for inspection, and appropriate signals are sent to the sorting mechanism so that the silverware piece is reprocessed.

3.4 – Inspection

Once a silverware piece is successfully identified, its orientation evaluated, and symmetry successfully verified, further analysis is performed in order to determine the cleanliness of the silverware piece. Dirt particles are located by human vision as discrepancies in surface texture. Sharp differences in colors and patterns are observed at the boundaries of the dirt particles on the silverware piece. The inspection algorithm implemented in this thesis attempts to emulate this. It is observed that due to the sharp differences in color at the boundaries of dirt particles, distinct edges will be formed when subjecting the image to an edge-detection algorithm. However, any actual physical features present on the silverware piece will also form edges. Hence a comparison is required in order to determine whether an edge is being formed due to dirt particles or otherwise. Figure 22 shows how actual physical features on a silverware piece form edges.

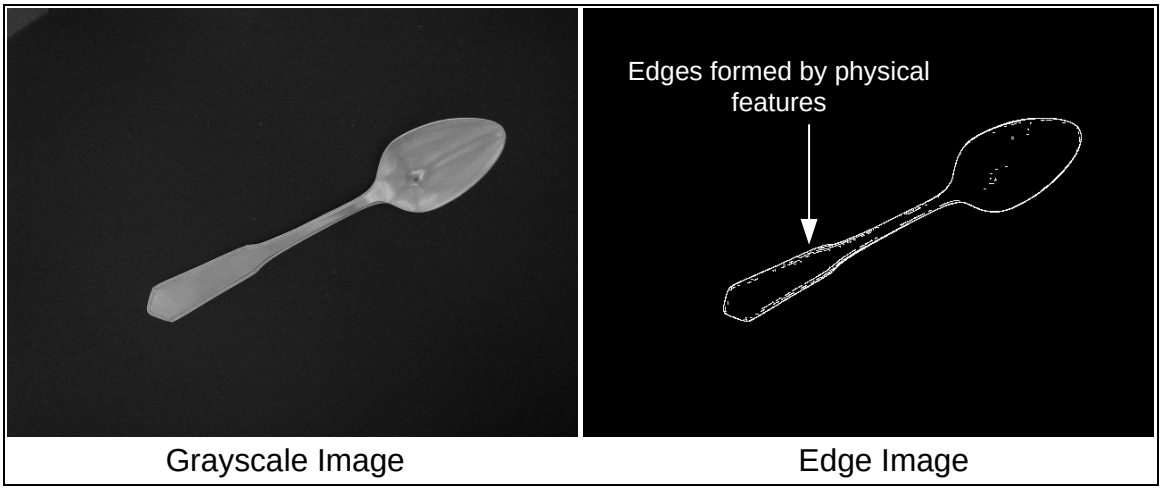


Figure 22: Clean Silverware Piece and Edges Formed by Physical Features.

Figure 23 shows how dirt particles on a silverware piece form edges.

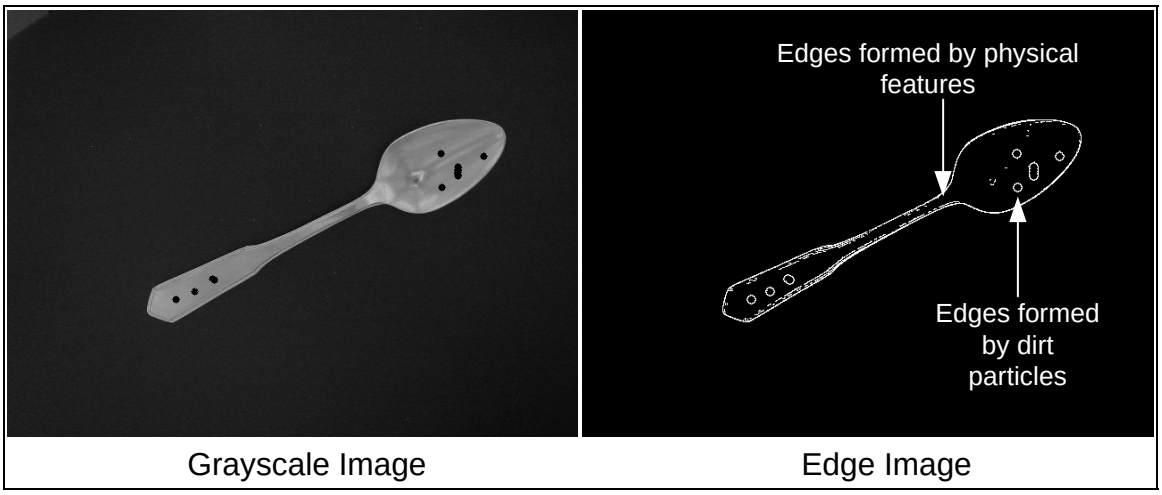


Figure 23: Dirty Silverware Piece and Edges Formed by Dirt Particles.

The orientation and symmetry information previously discussed is used to select the appropriate prototype edge-image for comparison purposes. During comparison, each edge in an edge image of the silverware piece being inspected is compared against edges in the prototype edge image. If the edge in the test image has a corresponding edge in the prototype image, then it means that some physical feature actually present on the silverware piece is forming the edge. However if no corresponding edge is located in the prototype image, then it implies that a feature not present in the prototype is forming the edge. This feature is most likely a dirt particle, and hence the system classifies the silverware piece as dirty.

Repeating (1) given in Chapter 2, we have

$$W = d_o \frac{w}{f} - w \qquad (1)$$

Then using numerical values from our experimental setup, namely f = 25mm, w = 0.01 mm (1024 pixels on 10.24mm) and d_o = 24.5 inches, we obtain W= 0.23992 mm. This is the smallest dimension on the silverware piece that will map onto one pixel of the camera

sensor. Accordingly, the inspection algorithm should be able to locate any dirt particle whose dimensions are greater than or equal to 0.24mm x 0.24mm.

To facilitate comparison of an acquired (test) image with a prototype image, the test image is rotated and resized, using the functions provided in NI libraries, so that both images have the same size and the LDs are oriented in the same direction. It is observed, however, that even after rotation and resizing, a pixel-by-pixel correspondence is very improbable. This is due to the very low probability of two silverware pieces being imaged at identical locations under the camera, in identical positions, and precisely the same timing. Imaging of silverware pieces even under slightly different positions, or different locations under the camera or at different timings, will result in a significant number of changes in the values of pixels in the image, even though the human eye might not detect it. Hence the inspection algorithm searches in a neighborhood of a pixel instead of looking for a single pixel. It also incorporates a mechanism that keeps track of the direction of propagation of the edge by means of an error vector, ensuring that the search for corresponding pixels occurs in the most probable regions. During each search for a pixel, the error vector records the deviation between expected location of pixel and the actual location at which it is found. This deviation is then used to offset the search for the next pixel so in an attempt to minimize the deviation for the next search. As the process of storing deviations and offsetting searches is repeated over the whole process, it is tantamount to tracking the direction of propagation of an edge. The tolerance levels for the error vector are to pre-defined and depend upon the size of the image being

processed. A detailed flowchart representing the inspection algorithm and code developed for the algorithm can be seen in Appendix A.

Figure 24 shows dirt located on a silverware piece after the inspection process. Figure 25 shows the processed image of a clean silverware piece. Figure 26 shows a context level flowchart for the entire processing algorithm. The experiments conducted and results obtained are discussed in Chapter 4.

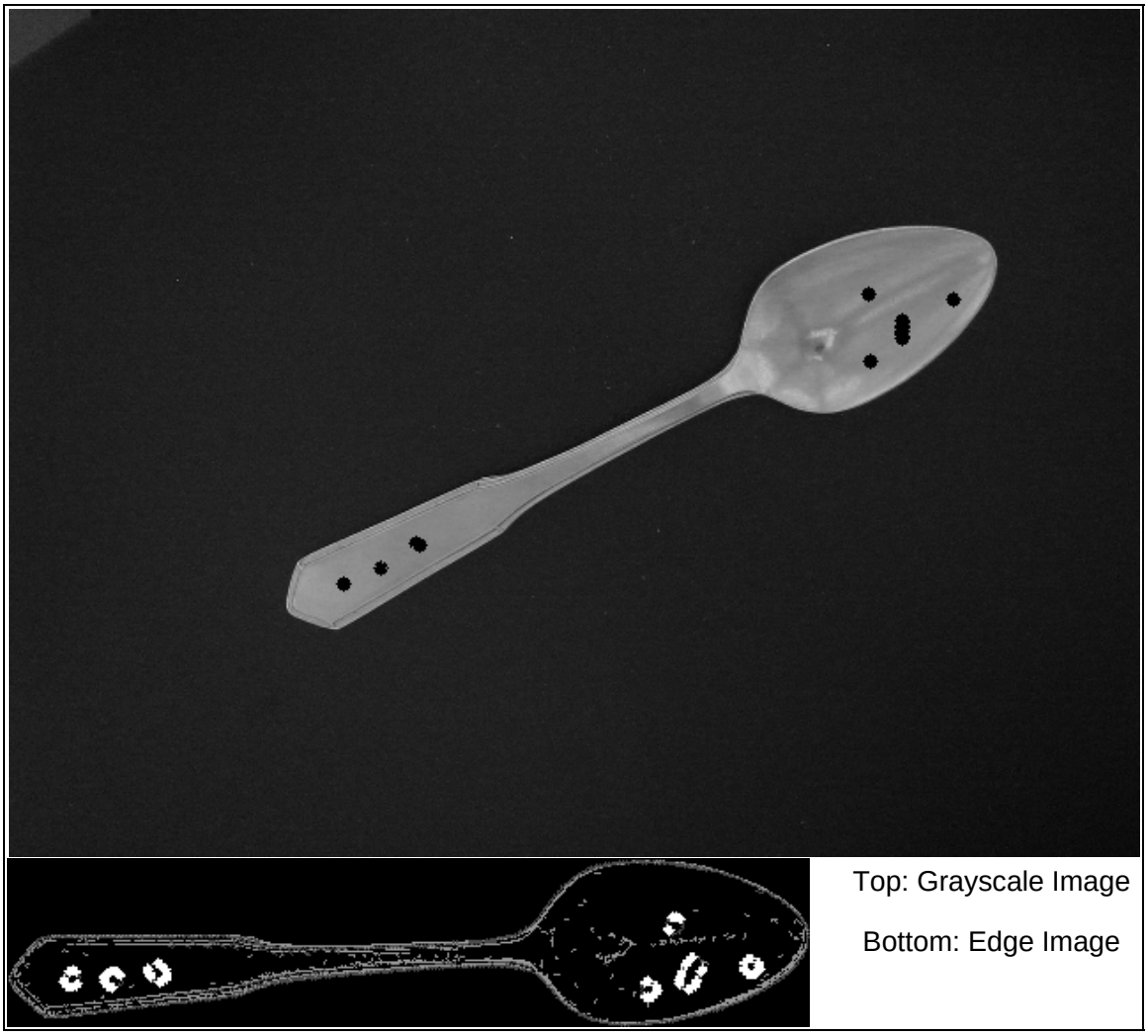


Figure 24: Bold White Spots Mark Dirt Located on Silverware Piece After Inspection.

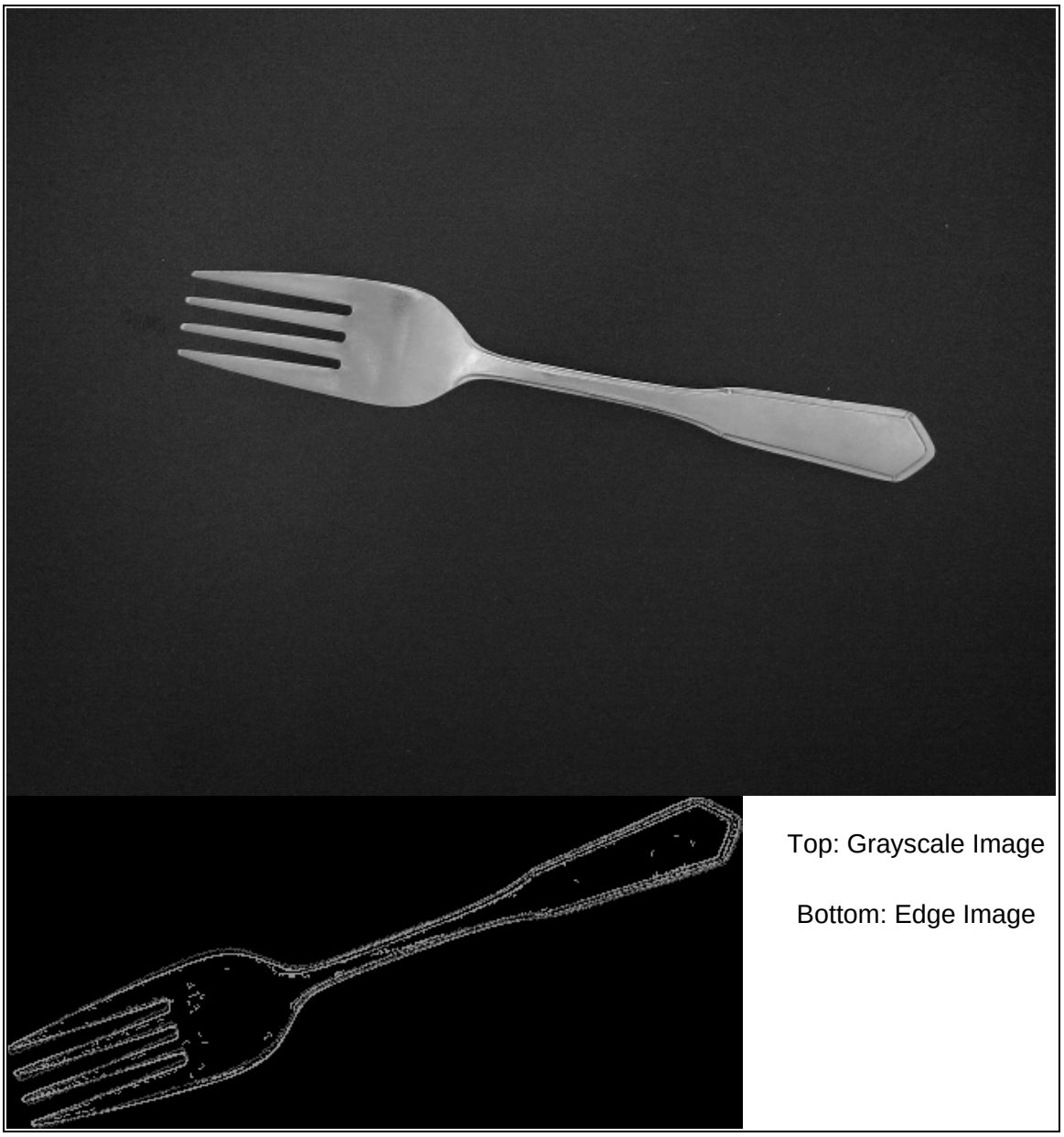


Figure 25: No Dirt Located on a Clean Silverware Piece.

3.5 – New Techniques Developed

The vision algorithm developed herein can be employed not only for the identification and inspection of silverware pieces, but also to almost any other problem requiring the same for objects (of any size and shape), with minimal modifications. The algorithm can

be ported for use with other image types with slight modifications. The vision system is capable of automatically generating all prototype images it requires (from pre-acquired grayscale images) to process test images. The vision system developed presents modifications to existing lighting setups in order to refine image quality. The vision system presents a fast identification procedure that works by comparison of feature sets and identifies shapes with a high reliability. The system also presents a technique to identify orientations of objects with accuracy comparable to that of template matching. The system also presents methods to evaluate the degree of symmetry/non-symmetry present in an object using blob analysis results. Finally the vision algorithm presents a new technique to detect the presence of surface anomalies (independent of their size and shape) on the surface of the object. The inspection procedure uses algorithms that were developed to track edges in images. The code and flowcharts for the above algorithms can be found in Appendix-A. This technique needs a special mention since it takes the solution a step closer to the solution domain from the problem domain. It simulates the human system more than previous systems since it deals with entities such as edges and shapes, and properties such as symmetry and orientation rather than mere pixels, making the algorithm easily portable to other applications. The abovementioned techniques can be applied together or individually for other problems depending on their respective needs.

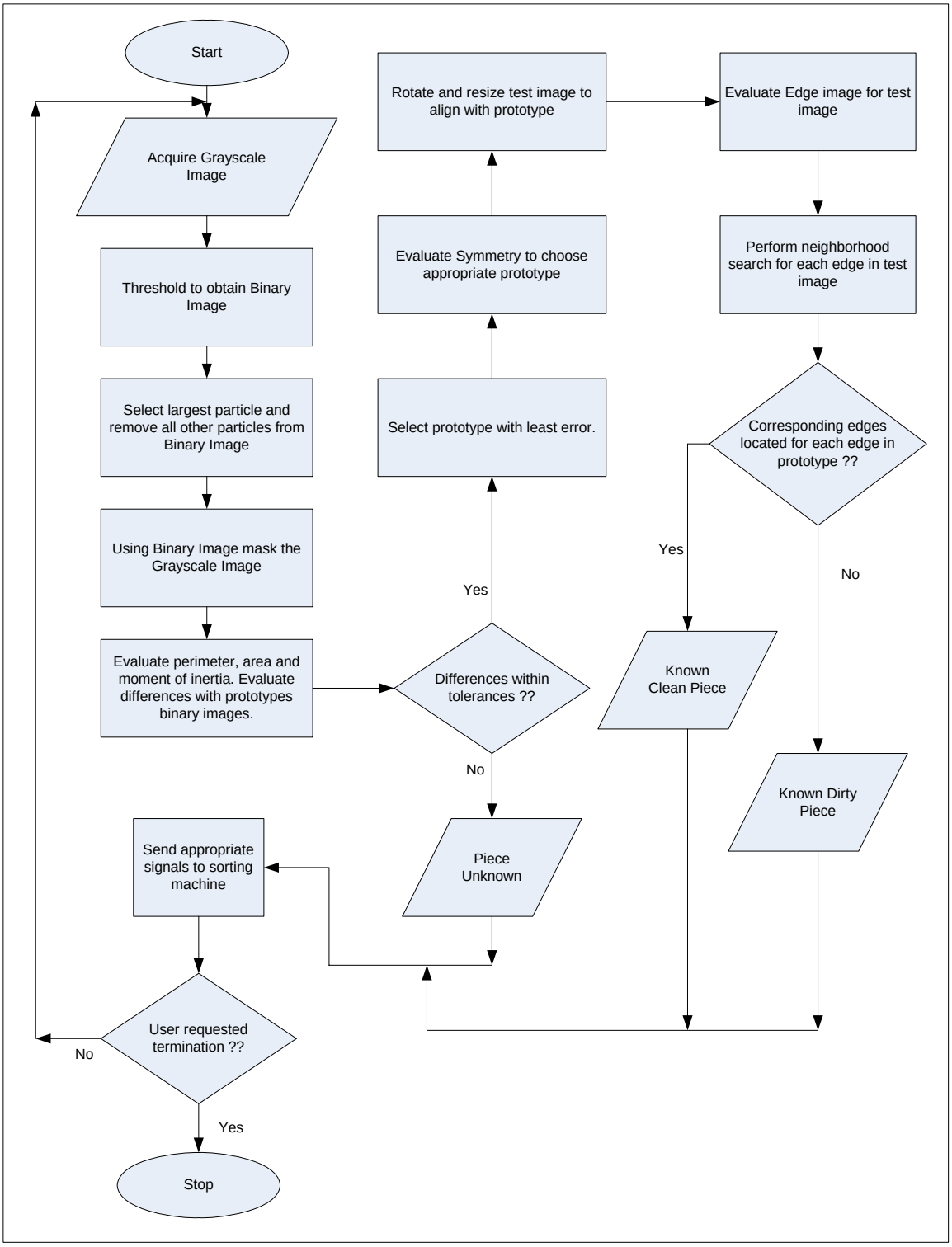


Figure 26: Context Level Flowchart Representing Image Acquisition, Pre-processing, Identification and Inspection.

Chapter 4: Experimental Results and Discussion

The vision system described in Chapter 3 was tested to evaluate its performance in identifying and inspecting silverware pieces. Mixed batches of randomly oriented silverware pieces were fed in random order as input to the vision system, and the outputs were analyzed to evaluate the efficiencies of identification and inspection. Silverware pieces were manually placed on the conveyer belt upstream of the lighting box to isolate the efficiency of the vision system from the efficiency of the hopper mechanism used for singulation of silverware pieces. Manually placing silverware on the moving magnets effectively simulates a 100% efficient singulation system.

The vision system can operate only below a certain maximum processing rate due to the constraint on time available for processing. The vision system must complete processing of one silverware piece before the next silverware piece arrives under the camera for imaging. The maximum processing rate is inversely proportional to the time required for vision system processing. A fraction of the time between the triggers caused by two consecutive silverware pieces is consumed for imaging the silverware piece and another fraction is consumed by waiting for the silverware piece to arrive under the camera. For various processing rates, Figure 27 presents the total time between two consecutive triggers, image acquisition time, time available for processing and actual time required.

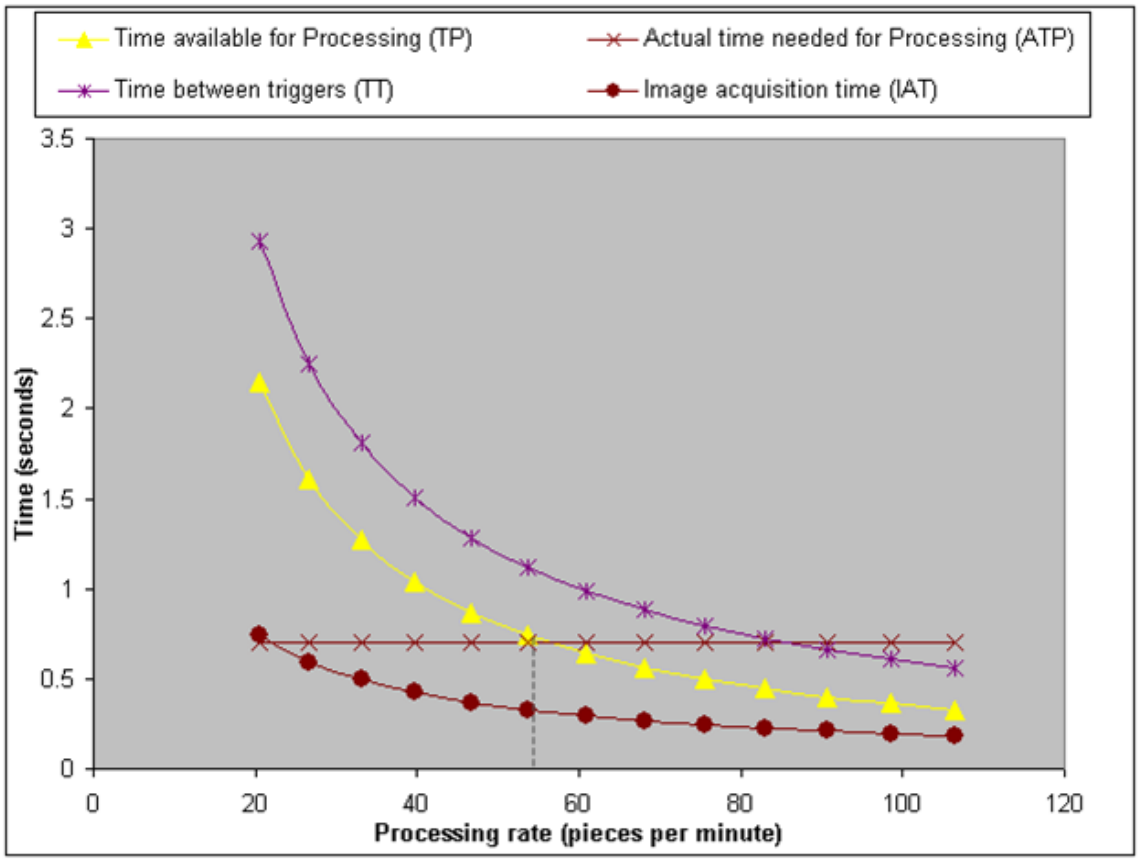


Figure 27: Time between triggers, Time Available for Processing, Actual Time Needed for Processing, Image Acquisition Time.

The relationships among these quantities are given by:

$$TT = 103.54 * BS^{(-1.1896)}$$

$$AR = 33.75 * BS$$

$$IAT = 500/AR$$

$$TP = TT - IAT - IWT$$

$$PPM = 60/TT$$

Where *TT* is Time between Triggers, *AR* is Camera Line Scan Rate, *IAT* is Image Acquisition Time, *TP* is Time available for Processing and *PPM* is Silverware Piece Processing Rate, BS is Belt Speed and *IWT* is Wait Time before start of Image Acquisition.

The actual time needed for processing depends upon the speed and processing power of the computer employed by the vision system. The current setup uses a PC with a 1.1GHz processor and 392 MB of RAM. This configuration results in a maximum processing time of about 0.7 seconds for an image size of 960 x 500 pixels. It can be seen from Figure 27 that an intersection occurs between “actual time needed for processing” graph and the “time available for processing” graph, indicated by the dashed vertical line. The processing rate at which this intersection occurs is the maximum processing rate, namely 55 pieces per minute, which the vision system can handle. At processing rates higher than this, one or more pieces will not be processed, since the vision system would still be busy processing a previous image. Using a state of the art computer for processing (e.g. 3.8 GHz with 1GB of RAM) will reduce the actual time needed for processing, allowing higher processing rates.

Testing was done at different speeds of conveying of silverware pieces under the camera, equivalent to setting the processing rate. Images of the four different “clean” silverware pieces are shown in Figures 28 and 29. “Dirty” silverware pieces, examples of which are shown in Figures 30 and 31, were made “artificially dirty” by means of a black felt tip pen. The conveying speeds arbitrarily chosen for testing purposes correspond to processing rates of 33 pieces per minute, 39 pieces per minute and 46 pieces per minute. Silverware pieces were placed on the magnets manually, and whenever (by mistake) improper placement of silverware onto the magnets occurred, the entire batch of silverware was re-fed to the vision system. A silverware piece was placed on each passing magnet. All images acquired during testing were of size 960 x 500 pixels. Test

results for processing rate of 33, 39, and 46 pieces per minute are shown in Table 1 through 6.

Line Scan Rate for Image Acquisition	1.013 KHz
Average time taken to acquire image	0.505 seconds
Average time taken to process image	0.512 seconds
Total number of clean silverware pieces fed	100
Total number of dirty silverware pieces fed	100
Total number of silverware pieces fed	200
Total number of silverware pieces successfully identified	200/200 (100%)
Total number of “clean” pieces classified as “clean”	87/100 (87%)
Total number of “dirty” pieces classified as “dirty”	92/100 (92%)

Table 1:Test results for processing rate of 33 pieces/minute (Set-1: “Artificial Dirt”).

Line Scan Rate for Image Acquisition	1.182 KHz
Average time taken to acquire image	0.436 seconds
Average time taken to process image	0.527 seconds
Total number of clean silverware pieces fed	100
Total number of dirty silverware pieces fed	100
Total number of silverware pieces fed	200
Total number of silverware pieces successfully identified	200/200 (100%)
Total number of “clean” pieces classified as “clean”	88/100 (88%)
Total number of “dirty” pieces classified as “dirty”	93/100 (93%)

Table 2:Test results for processing rate of 39 pieces/minute (Set-1: “Artificial Dirt”).

Line Scan Rate for Image Acquisition	1.350 KHz
Average time taken to acquire image	0.382 seconds
Average time taken to process image	0.519 seconds
Total number of clean silverware pieces fed	100
Total number of dirty silverware pieces fed	100
Total number of silverware pieces fed	200
Total number of silverware pieces successfully identified	200/200 (100%)
Total number of “clean” pieces classified as “clean”	86/100 (86%)
Total number of “dirty” pieces classified as “dirty”	92/100 (92%)

Table 3:Test results for processing rate of 46 pieces/minute (Set-1: “Artificial Dirt”).

Line Scan Rate for Image Acquisition	1.013 KHz
Average time taken to acquire image	0.501 seconds
Average time taken to process image	0.509 seconds
Total number of clean silverware pieces fed	100
Total number of dirty silverware pieces fed	100
Total number of silverware pieces fed	200
Total number of silverware pieces successfully identified	200/200 (100%)
Total number of “clean” pieces classified as “clean”	86/100 (86%)
Total number of “dirty” pieces classified as “dirty”	90/100 (90%)

Table 4:Test results for processing rate of 33 pieces/minute (Set-2: “Artificial Dirt”).

Line Scan Rate for Image Acquisition	1.182 KHz
Average time taken to acquire image	0.433 seconds
Average time taken to process image	0.513 seconds
Total number of clean silverware pieces fed	100
Total number of dirty silverware pieces fed	100
Total number of silverware pieces fed	200
Total number of silverware pieces successfully identified	200/200 (100%)
Total number of “clean” pieces classified as “clean”	87/100 (87%)
Total number of “dirty” pieces classified as “dirty”	91/100 (91%)

Table 5:Test results for processing rate of 33 pieces/minute (Set-2: “Artificial Dirt”).

Line Scan Rate for Image Acquisition	1.350 KHz
Average time taken to acquire image	0.381 seconds
Average time taken to process image	0.521 seconds
Total number of clean silverware pieces fed	100
Total number of dirty silverware pieces fed	100
Total number of silverware pieces fed	200
Total number of silverware pieces successfully identified	200/200 (100%)
Total number of “clean” pieces classified as “clean”	86/100 (86%)
Total number of “dirty” pieces classified as “dirty”	91/100 (91%)

Table 6:Test results for processing rate of 33 pieces/minute (Set-2: “Artificial Dirt”).

Tests were conducted for some silverware pieces in Set-1 that contained actual dirt, including dried egg yolk, dried coffee stains and dried sauce stains. Test results for Set-1 for processing rates of 33, 39, and 46 pieces per minute are shown in Table 7 through 9.

Line Scan Rate for Image Acquisition	1.013 KHz
Average time taken to acquire image	0.503 seconds
Average time taken to process image	0.512 seconds
Total number of clean silverware pieces fed	25
Total number of dirty silverware pieces fed	25
Total number of silverware pieces fed	50
Total number of silverware pieces successfully identified	50/50 (100%)
Total number of “clean” pieces classified as “clean”	43/50 (86%)
Total number of “dirty” pieces classified as “dirty”	46/50 (92%)

Table 7:Test results for processing rate of 33 pieces/minute (Set-1: “Real Dirt”).

Line Scan Rate for Image Acquisition	1.182 KHz
Average time taken to acquire image	0.432 seconds
Average time taken to process image	0.527 seconds
Total number of clean silverware pieces fed	25
Total number of dirty silverware pieces fed	25
Total number of silverware pieces fed	50
Total number of silverware pieces successfully identified	50/50 (100%)
Total number of “clean” pieces classified as “clean”	44/50 (88%)
Total number of “dirty” pieces classified as “dirty”	45/50 (90%)

Table 8:Test results for processing rate of 39 pieces/minute (Set-1: “Real Dirt”).

Line Scan Rate for Image Acquisition	1.350 KHz
Average time taken to acquire image	0.396 seconds
Average time taken to process image	0.523 seconds
Total number of clean silverware pieces fed	25
Total number of dirty silverware pieces fed	25
Total number of silverware pieces fed	50
Total number of silverware pieces successfully identified	50/50 (100%)
Total number of “clean” pieces classified as “clean”	43/50 (86%)
Total number of “dirty” pieces classified as “dirty”	46/50 (92%)

Table 9:Test results for processing rate of 46 pieces/minute (Set-1: “Real Dirt”).

The camera line-scan rate is directly proportional to belt speed. For higher processing rates the belt speeds are higher and hence the line-scan rates are higher. Processing

efficiency is the percentage of correctly identified and correctly classified silverware pieces. It can be concluded from the results given above that the processing efficiency and processing time are independent of the processing rate. The false “dirty” classifications of clean silverware pieces were caused by variations in positions of silverware pieces and by shapes of reflections of surroundings onto the silverware pieces. The “false” clean classifications of dirty silverware were caused by dirt particles located too close to physical features forming “edges” in images, since the neighborhood search feature in the inspection algorithm assumes the dirt to be part of the physical feature itself. It can be also seen that results using “real dirt” (Tables 7-9) are comparable with results using “artificial dirt” (Tables 1-6). Our vision system algorithm allows the setting of various tolerances and thresholds, whose values influence the efficiency of the system. In general, these values should be tuned initially for optimal performance of the vision system.

It is concluded that the vision system developed has a high reliability for identification and a reasonable reliability for classifying clean and dirty silverware pieces. The accuracy of classification of dirty pieces is more crucial than the accuracy of classification of clean pieces. Our vision system algorithm also establishes an initial approach to classify specularly reflective objects with small anomalies (i.e. “dirt”). The vision system surpasses its minimum throughput target of identifying and inspecting silverware pieces at a processing rate of 30 pieces per minute. However there remains room for improvement in classification.

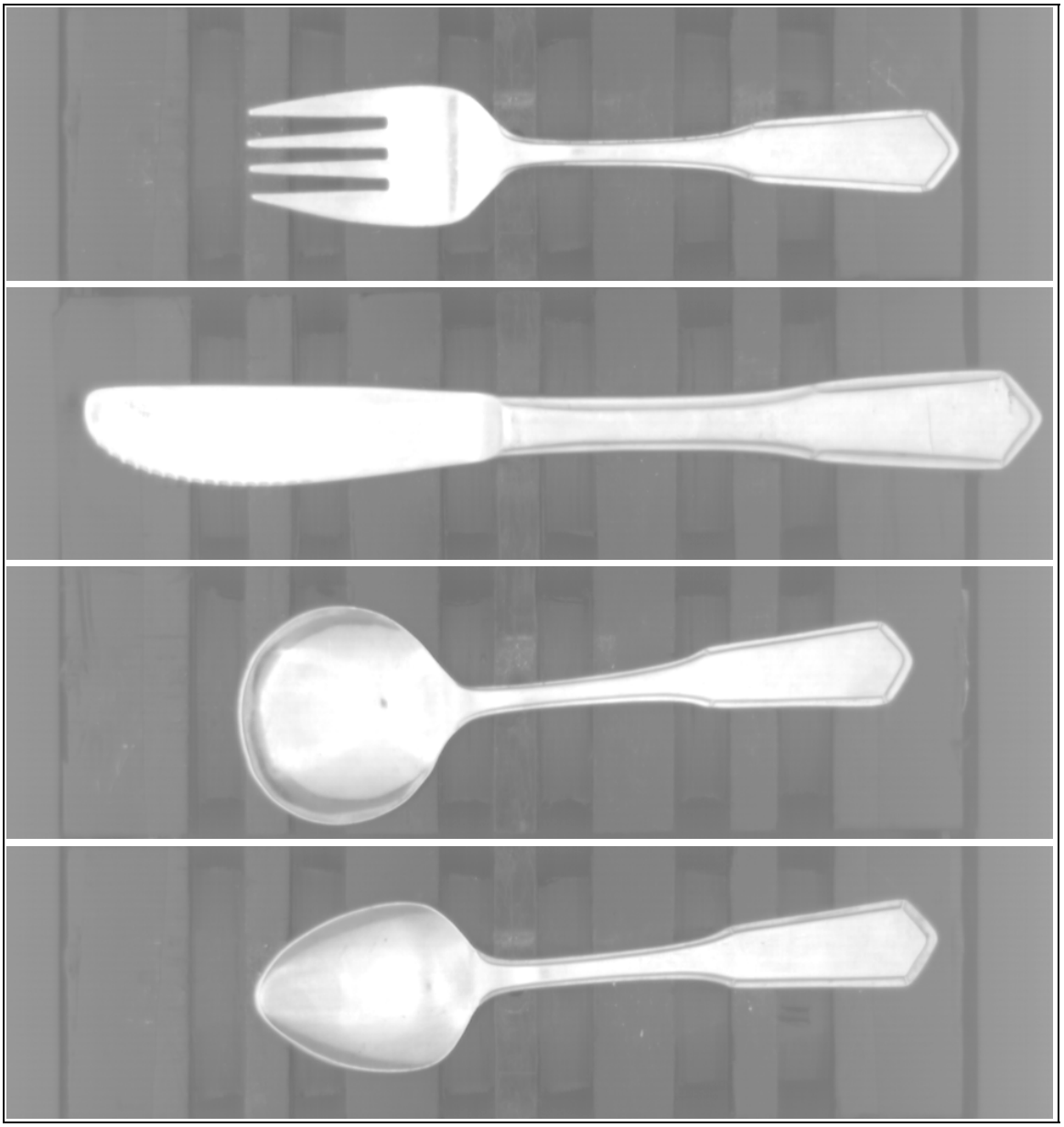


Figure 28: Images of “Clean” Spoon, Soup Spoon, Knife and Fork (Set-1).

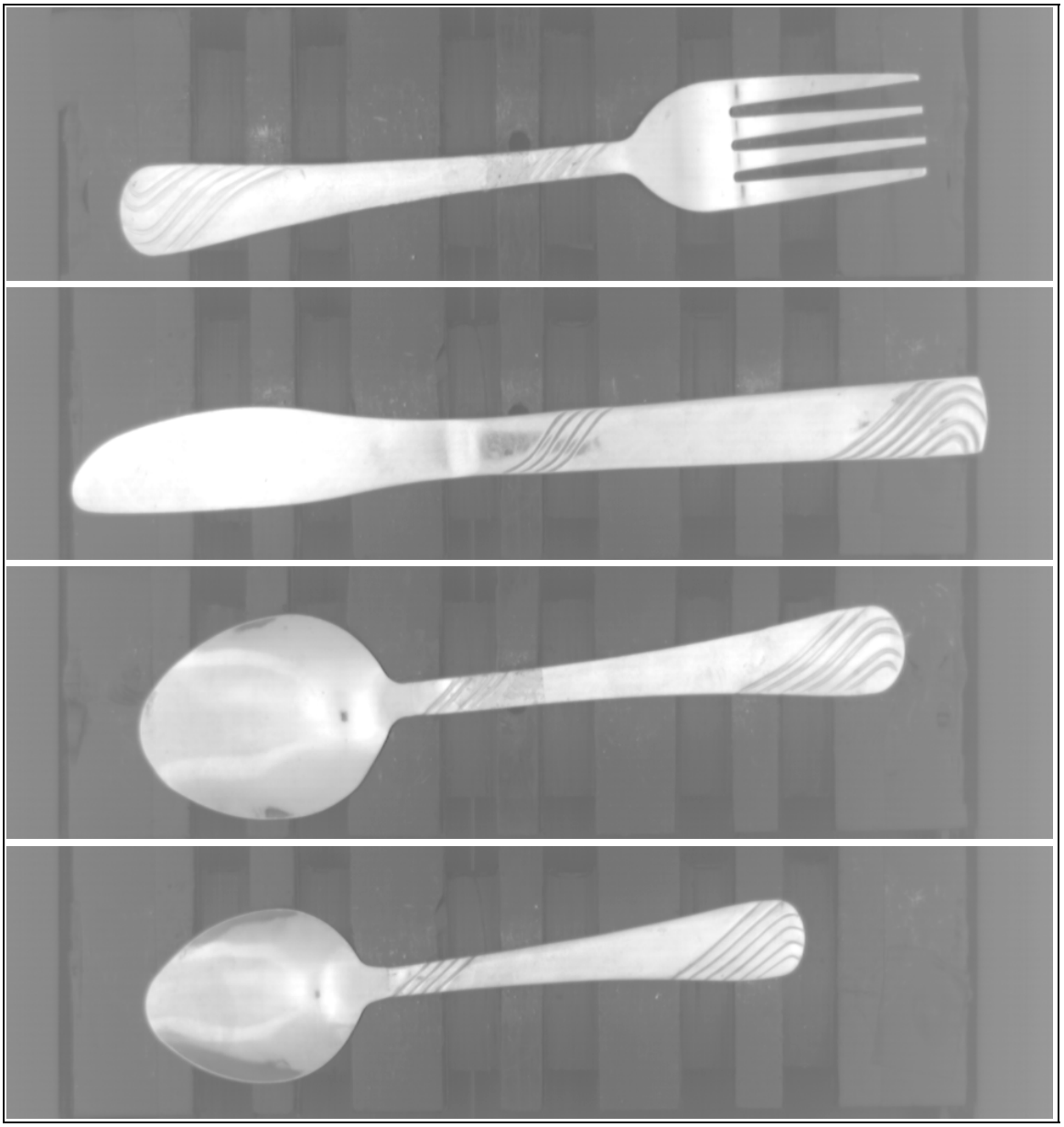


Figure 29: Images of “Clean” Spoon, Soup Spoon, Knife and Fork (Set-2).

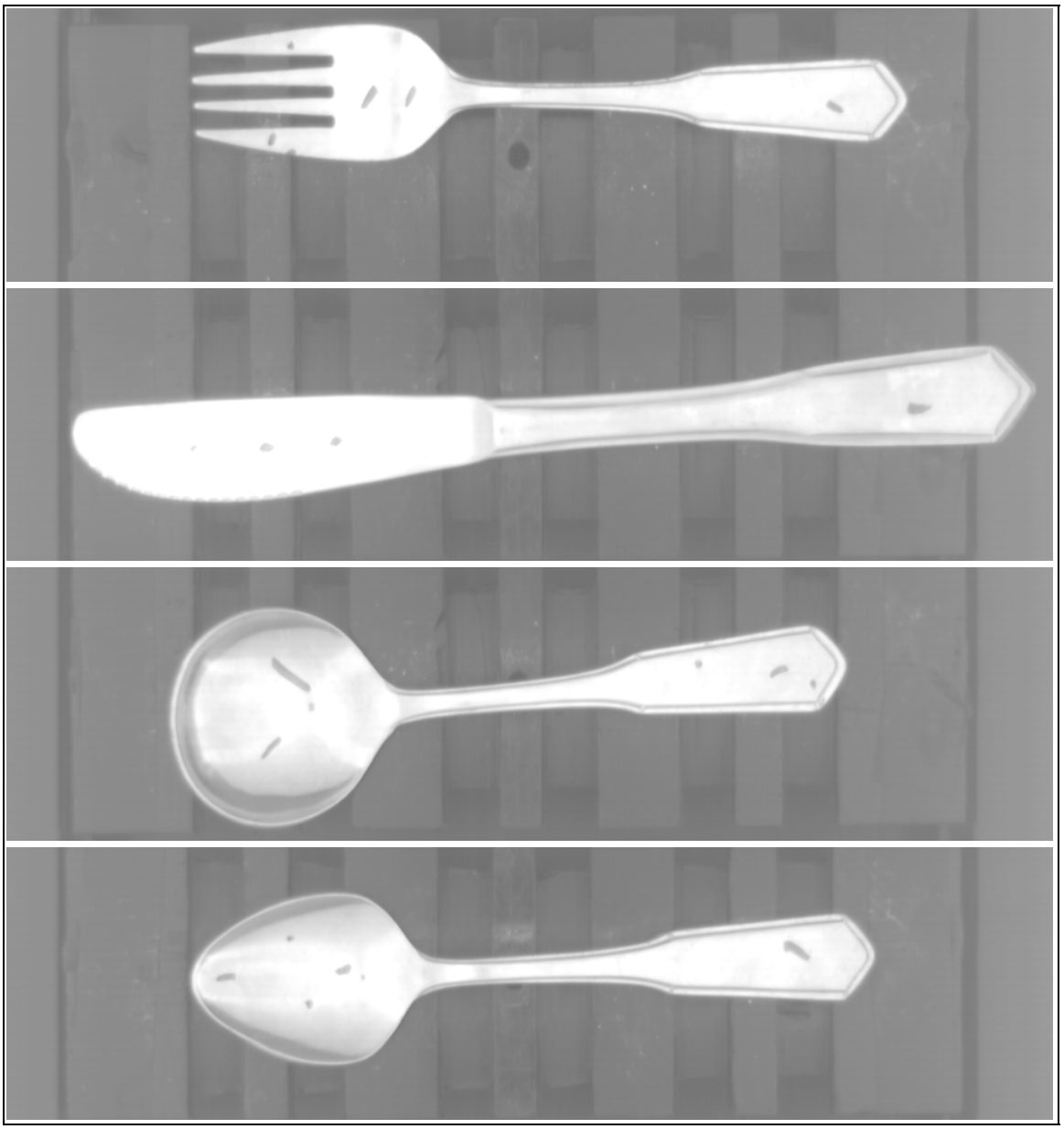


Figure 30: Images of “Dirty” Spoon, Soup Spoon, Knife and Fork (Set-1).

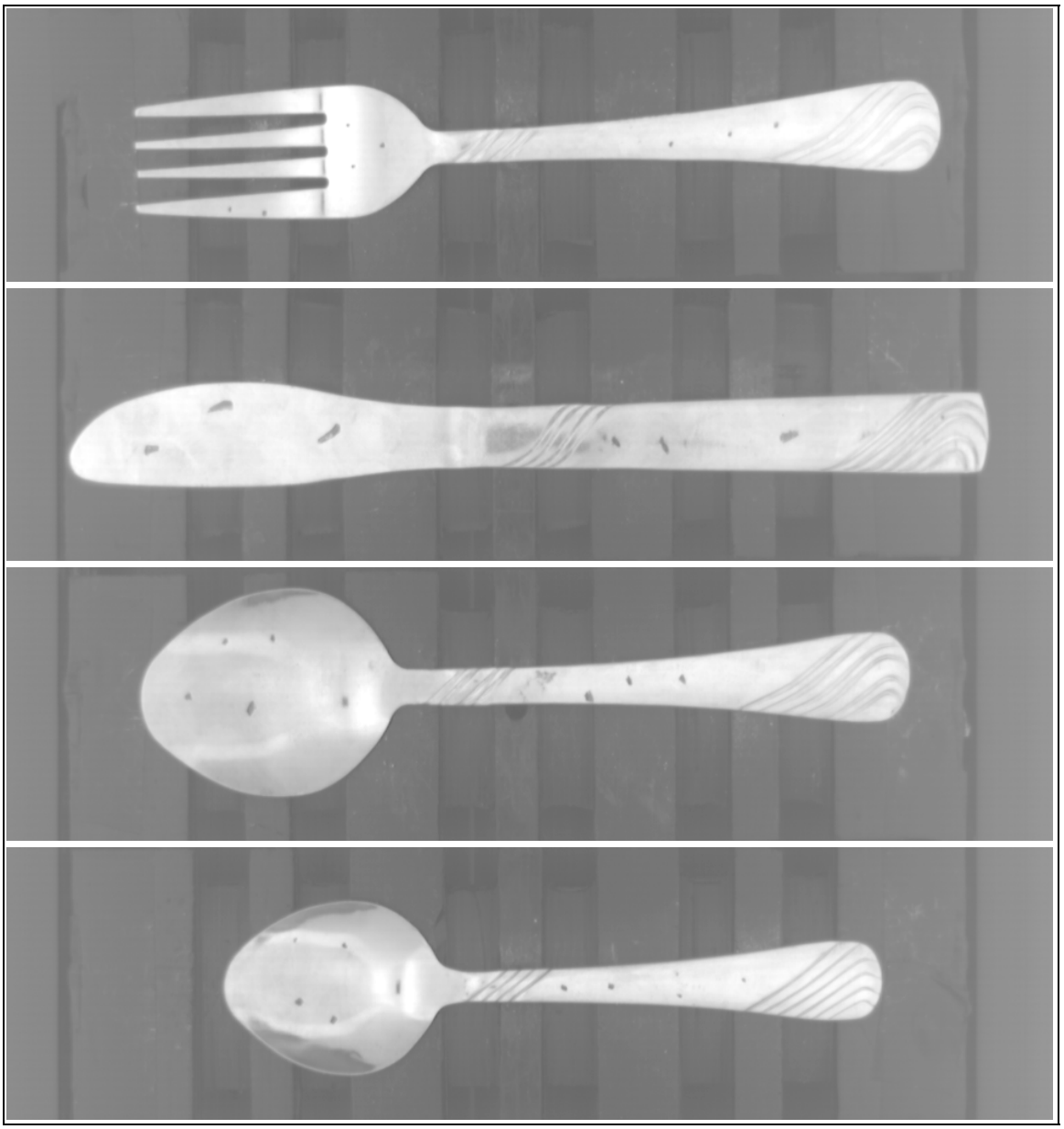


Figure 31: Images of “Dirty” Spoon, Soup Spoon, Knife and Fork (Set-2).

Chapter 5: Conclusions and Recommendations

5.1 – Conclusions

This thesis focused on developing a vision system for identification and inspection of silverware. A vision system capable of identifying and inspecting silverware pieces was developed. The lighting setup was modified in order to reduce shadows and ill-lighted regions. Noise lines were eliminated from the image by selecting an appropriate power source for the lighting equipment. The current system identifies silverware pieces with an accuracy of 100%, classifies clean silverware pieces with an average accuracy of 87%, and classifies dirty silverware pieces with an average accuracy of 91%, with the average processing time of approximately 0.52 seconds. The maximum processing rate for the current setup is approximately 55 pieces per minute.

5.2 – Contributions

In this thesis we have contributed the following (Refer Appendix-A for code and flowcharts):

- A fast identification algorithm using feature sets to identify objects of any size and shape.
- A technique to evaluate orientation of an object.
- A technique to evaluate degree of symmetry present (about the longest axis) in an object.

- A technique to compare and track edges between two images.
- A technique to detect the presence of surface anomalies.
- A modification to the existing lighting technique to enhance image quality by reducing ill-lighted regions.

The algorithms developed here lay an initial approach to inspecting surfaces of specularly reflective objects.

The algorithm can be easily ported to other applications with minor modifications. The algorithm can be used for vegetable sorting and classification in the food industry, for metal surface finish testing in the production sector, and in the web-handling industry to check shape integrity and surface quality of the web, to name a few examples of other applications where it can be employed.

5.3 – Recommendations

Use of a color camera (using Red-Blue-Green (RGB)) for imaging is expected to produce greater efficiency of the inspection system. However, using RGB images instead of grayscale images will increase the amount of processing required by roughly three times and will also increase the cost by a minimum factor of 2. Use of UV lighting will enhance the contrast between the silverware piece and dirt particles, but a UV camera is needed to acquire images under UV lighting which will increase camera costs by a factor of 3. Use of thermal/Infra-red imaging techniques is expected to eliminate the inefficiencies caused by specular reflections and improve the accuracy of the inspection algorithm, but such imaging equipment is costly (increases the cost by a factor of 5) to acquire and maintain.

Employing texture analysis can enhance the performance of the inspection algorithm, but such an analysis requires more processing significantly, which could reduce processing rate. Use of a higher resolution camera will yield in better images, and hence better inspection results, but higher resolution images will require more processing time. The initial approach to inspection of silverware pieces suggested in this thesis can be enhanced using one or more of the recommendations above, provided the associated drawbacks can be eliminated or tolerated. It is essential to keep in mind that the overall cost of the system must be minimized in order to have a viable commercial system. Hence we expect that an appropriate trade-off among cost, efficiency, and processing rate must be made for a commercially viable system. Cost is likely to be the dominant factor.

REFERENCES

(Basler, 2004) User's Manual, *http://www.baslerweb.com/popups/403/ L100_Users_ Manual.pdf,* Date Accessed: 10/10/2004

(EIO, 2004) Edmund Industrial Optics, *http://www.edmundoptics.com/us/*, Date Accessed: 12/03/2004

(EWWOP, 2004) Specular Reflection, *http://scienceworld.wolfram.com/physics/ SpecularReflection.html* , Date Accessed: 11/11/2004.

(Hashimoto,1995) Hashimoto, Sachiko, "Separation of Silverware For Machine Vision Sorting and Inspection. M.S. thesis, School of Mechanical and Aerospace Engineering, Stillwater, OK: Oklahoma State University, May 1995.

(IVP, 2004) Sorting Potatoes with an artificial eye, *http://www.machinevisiononline.org/ public/articles/ivp_potato_case1.pdf* , Date Accessed: 11/11/2004.

(JZW, 2004) Vision Inspection, *http://www.jzw.com.au/page42.html*, Date Accessed: 21/10/2004.

(Latvala,1999) Latvala, Jessica.Student, Oklahoma State University, BSME 1999, Research Log Book.

(MVA, 2004) Lighting & Optics Reference Guide, *http://www.sme.org/downloads/mva/ mvaposter.pdf*, Date Accessed: 10/10/2004.

(NIVCM, 2004) Imaq Vision Concepts Manual, *http://www.ni.com/pdf/manuals/ 372916c.pdf*, Date Accessed: 09/10/2004.

(Peddi, 2005) Peddi, Ravi Vamsidhar, "Silverware Sorting and Orienting System", M.S. thesis, School of Mechanical and Aerospace Engineering, Stillwater, OK: Oklahoma State University, 2005

(RMA, 2004) Way 2C, *http://www.way2c.com/w2home.htm*, Date Accessed: 21/10/2004.

(Ukiva, 2004) Applications of Vision Systems, *http://www.ukiva.org/pages/applica tions.html* , Date Accessed: 21/10/2004.

(Yeri, 2002) Yeri, Sandeep, "Classification of silverware pieces using machine vision", M.S. thesis, School of Mechanical and Aerospace Engineering, Stillwater, OK: Oklahoma State University, 2003

APPENDIX – A

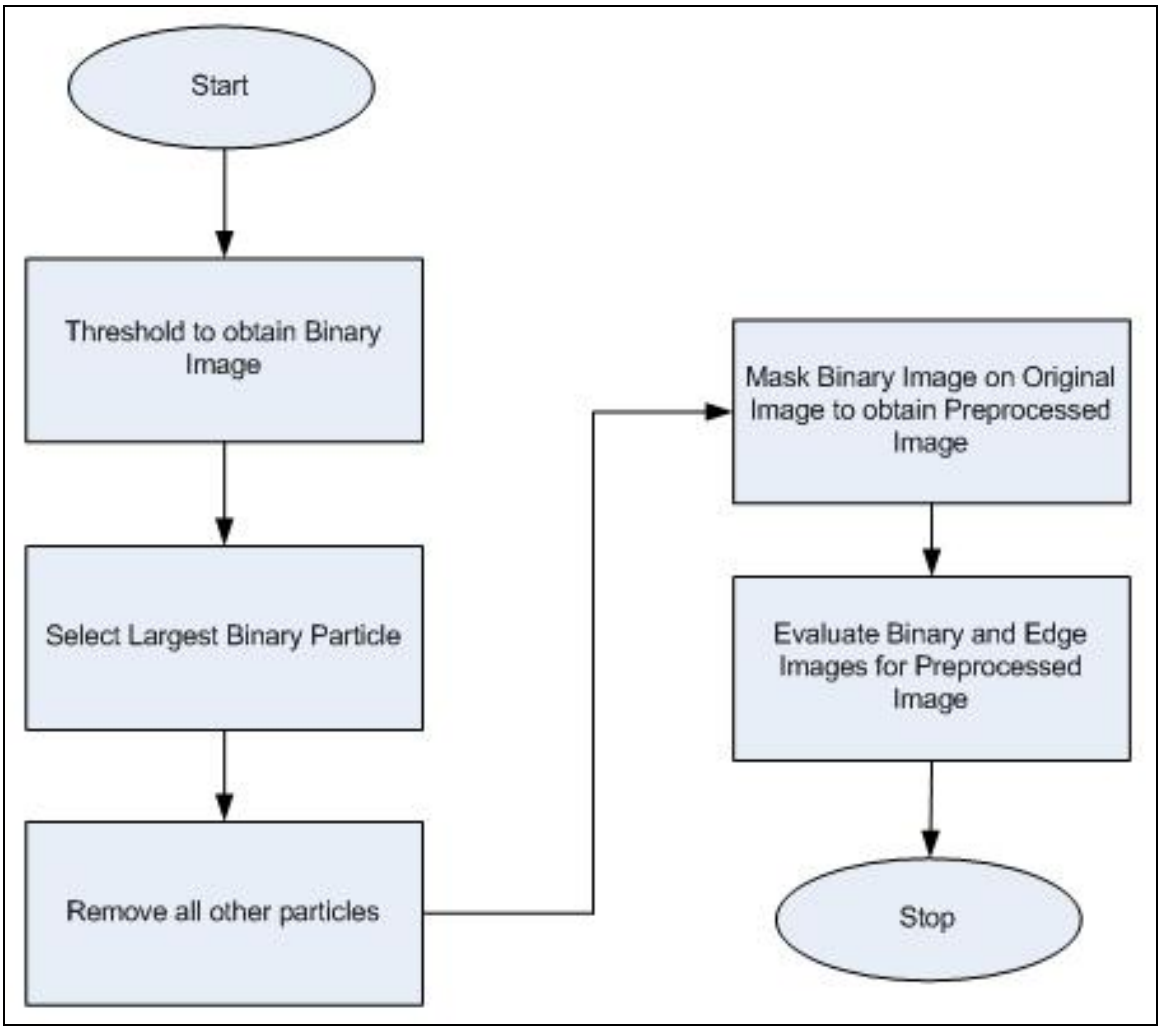
Software Code & Flowcharts

NI Vision Libraries are extensively used in the development of the algorithms in this thesis. The library has extensive routines to perform various image processing functions. The functions frequently employed in this thesis are

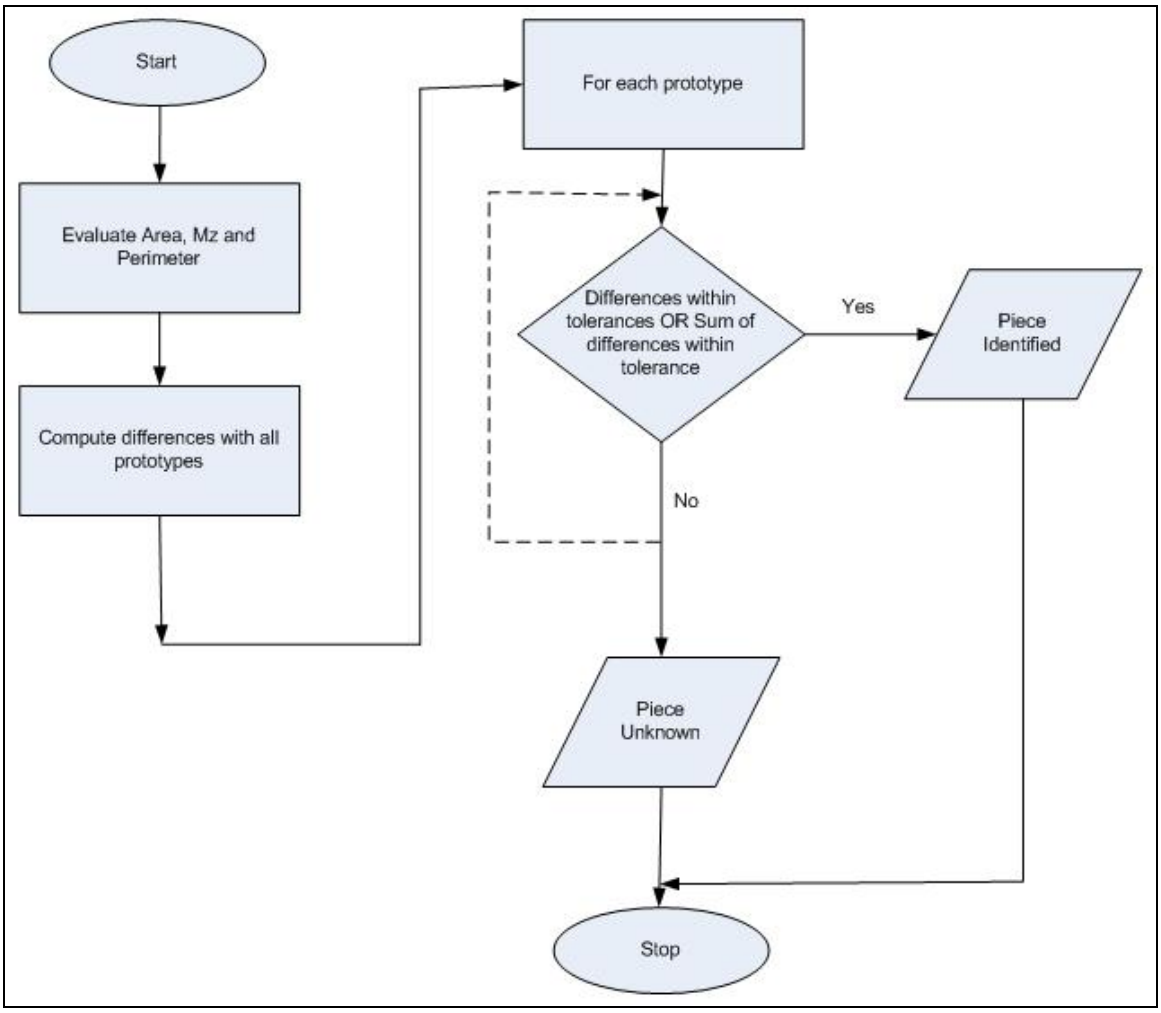
- Thresholding
- Edge detection
- Rotation
- Re-sampling

NI algorithms perform thresholding using Histogram based techniques. These include clustering, entropy, moments and interclass variance. Edge detection is based on Prewitt, Sobel, Laplacian and Gaussian filters. Rotation and re-smapling employ standard zero-order, bilinear and bi-cubic interpolation techniques. Detailed information about the aforementioned can be obtained at the NI website http://www.ni.com in a document titled "IMAQ Vision Concepts Manual".

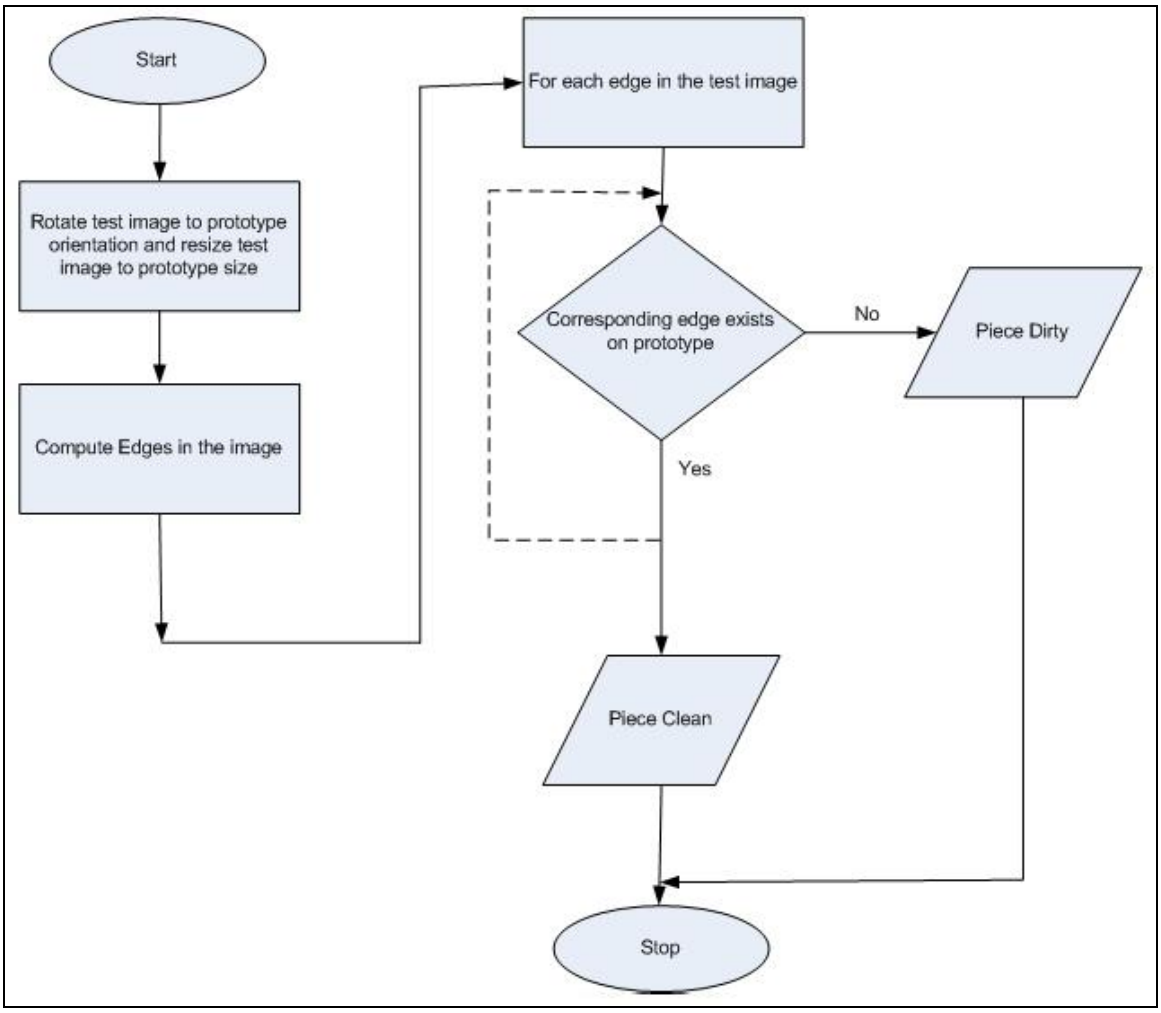
In what follows, we present 3 flow charts describing various processes in the complete pre-processing, identification and inspection procedure along with the code.s



Flowchart for Preprocessing Algorithm (See “Prepare_for_Matching” for related code).



Flowchart for Identification Algorithm (See “Process_Image” for related code).



Flowchart for Inspection Algorithm (See “Inspect_Image” for related code).

Code

File : ImgAn.h

```
#ifndef C_MY_IMAGE
#include <atlstr.h>

#include "niimaq.h"
#include "nivision.h"

class CMyImage
{
public :

	Image * main_image;
	Image * display_image;
	Image * edge_image;
	Image * binary_image;
	Image * temp_image;
	Image * temp_image1;
	Image * match_image;

	int num_proto;
	int inspect;
	int write_images;
	int im_count;

	CString im_name;
	CString im_path;
	CString errmsg;

	float *a,*cx,*cy,*ar,*mz,*ml,*sw,*sh,*pm,*pbx,*pby;
	int * im_sym;
	CString * pnames;
	Image ** bi, ** ei, ** pi, ** oei;
	float icx,icy;
	float icx2,icy2;
	float ibx,iby;
	float area,moment,perimeter;

	float area_tol,moment_tol,peri_tol,soft_tol_scale;
	float sym_x,sym_y;
	float sym_fact;

	int piece_found;
	int is_dirty;

	float bin_th;
	float edge_th;

public :

	CMyImage();
	~CMyImage();

	int Load_Image(CString fn);
	int Load_Proto(CString pt);
```

```
	int Make_Prototype(CString pt, CString fn);
	int Rotate_to_Horiz(void);
	int Collect_Info(void);

	int Process_Image();
	int find_proto_info();

	float Prepare_Angle(float deg,float mx,float my,Rect * bb,float
ll);
	int In_Rectangle(PointFloat pp,Rect * bb);
	int PrepareImage(Image * I,Image * BI);
	int Prepare_for_Matching(Image * I,Image * mI,Image * eI,Image *
bI);
	int Inspect_Image(Image * pe);

	//int Inspect_Image(Image * I, Image * eI,float iml,float
pml,float rot_ang,float icx,float icy,float cx,float cy);

};


struct belt_data
{
double belt_speed;
double del_time;
double belt_max_time;
double belt_tol;
double belt_acq_delay;
};

#define C_MY_IMAGE
#endif
```

File : ImgAn.cpp

```
#include "ImgAn.h"
#include "niimaq.h"
#include "nivision.h"
#include "math.h"
#include <stdio.h>
#include <conio.h>
#include <atlstr.h>
#include <list>
using namespace std;

#define PI 3.1415926538
float sf=1.0f;
float src_wd=4;
int ngs=2;
int ng=2;

void draw_plus(Image * I,float dcx,float dcy,int dx,int dy,float col);

CMyImage::CMyImage()
{
main_image=imaqCreateImage(IMAQ_IMAGE_U8,5);
display_image=imaqCreateImage(IMAQ_IMAGE_U8,5);
edge_image=imaqCreateImage(IMAQ_IMAGE_U8,5);
binary_image=imaqCreateImage(IMAQ_IMAGE_U8,5);
temp_image=imaqCreateImage(IMAQ_IMAGE_U8,5);
temp_image1=imaqCreateImage(IMAQ_IMAGE_U8,5);
match_image=imaqCreateImage(IMAQ_IMAGE_U8,5);

this->bi=NULL;
this->ei=NULL;
this->pi=NULL;
this->oei=NULL;
this->pnames=0;

this->num_proto=0;
this->write_images=1;
this->inspect=1;

this->area_tol = 12.0f;
this->moment_tol = 12.0f;
this->peri_tol = 12.0f;
this->sym_fact = 2.0f;
this->soft_tol_scale = 1.5f;

this->a=NULL;
this->cx=NULL;
this->cy=NULL;
this->ar=NULL;
this->mz=NULL;
this->ml=NULL;
this->sw=NULL;
this->sh=NULL;
```

```
this->pm=NULL;
this->pbx=NULL;
this->pby=NULL;
this->im_sym=NULL;
this->pnames=NULL;

this->im_path = "c:\\temp_im\\";
this->im_count=0;

//this->bin_th = 0.75f; Set - 1
this->bin_th = 0.625f;
//this->bin_th = 0.5f;
this->edge_th = 0.8f;
}

CMyImage::~CMyImage()
{
int i;
imaqDispose(main_image);
imaqDispose(edge_image);
imaqDispose(display_image);
imaqDispose(binary_image);
imaqDispose(temp_image);
imaqDispose(temp_image1);
imaqDispose(match_image);

if (bi!=NULL)
      {
      for(i=0;i<this->num_proto;i++) imaqDispose(bi[i]);
      delete[] bi;
      }

if (ei!=NULL)
      {
      for(i=0;i<this->num_proto;i++) imaqDispose(ei[i]);
      delete[] ei;
      }

if (pi!=NULL)
      {
      for(i=0;i<this->num_proto;i++) imaqDispose(pi[i]);
      delete[] pi;
      }

if (oei!=NULL)
      {
      for(i=0;i<this->num_proto;i++)
            {
            if (im_sym[i]==1) imaqDispose(oei[i]);
            }
      delete[] oei;
      }

if (this->a!=NULL) delete[] a;
if (this->cx!=NULL) delete[] cx;
if (this->cy!=NULL) delete[] cy;
if (this->ar!=NULL) delete[] ar;
```

```
if (this->mz!=NULL) delete[] mz;
if (this->ml!=NULL) delete[] ml;
if (this->sw!=NULL) delete[] sw;
if (this->sh!=NULL) delete[] sh;
if (this->pm!=NULL) delete[] pm;
if (this->pbx!=NULL) delete[] pbx;
if (this->pby!=NULL) delete[] pby;
if (this->pnames!=NULL) delete[] pnames;
if (this->im_sym!=NULL) delete[] im_sym;
}

int CMyImage::Load_Image(CString fn)
{
int n;
n=imaqReadFile(this->main_image,fn.GetString(),NULL, NULL);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
return(0);
}

int CMyImage::Load_Proto(CString pt)
{
CString str,s1,s2,s3,s4;
char fs[300];
char ptf[300];
int i,n;
FILE * fp;

strcpy(ptf,pt.GetString());
fp=fopen(ptf,"r");
if (fp==NULL) {this->errmsg="Unable to open Prototype File !!";return(-
1);}

fgets(fs,299,fp);
this->num_proto = atoi(fs);
if (this->num_proto<=0) {this->errmsg="Corrupt Prototype File
!!";return(-1);}

this->bi = new Image * [this->num_proto];
this->ei = new Image * [this->num_proto];
this->oei = new Image * [this->num_proto];
this->pi = new Image * [this->num_proto];

this->a=new float[this->num_proto];
this->cx=new float[this->num_proto];
this->cy=new float[this->num_proto];
this->ar=new float[this->num_proto];
this->mz=new float[this->num_proto];
this->ml=new float[this->num_proto];
this->sw=new float[this->num_proto];
this->sh=new float[this->num_proto];
this->pm=new float[this->num_proto];
this->pbx=new float[this->num_proto];
this->pby=new float[this->num_proto];
this->im_sym=new int[this->num_proto];
this->pnames=new CString[this->num_proto];
```

```
for(i=0;i<this->num_proto;i++)
      {
      this->bi[i]=imaqCreateImage(IMAQ_IMAGE_U8,5);
      this->ei[i]=imaqCreateImage(IMAQ_IMAGE_U8,5);
      this->pi[i]=imaqCreateImage(IMAQ_IMAGE_U8,5);
      fgets(fs,299,fp);fs[strlen(fs)-1]='\0';str=fs;
      fgets(fs,299,fp);fs[strlen(fs)-1]='\0';s1=fs;
      s2="e_";s2=s2 + s1;s2=s2 + ".bmp";
      s3="b_";s3=s3 + s1;s3=s3 + ".bmp";
      s4="p_";s4=s4 + s1;s4=s4 + ".bmp";
      s2 = str + s2;
      s3 = str + s3;
      s4 = str + s4;

      this->pnames[i]=s1;
      n=imaqReadFile(this->ei[i],s2.GetString(),NULL, NULL);
      if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
      n=imaqReadFile(this->bi[i],s3.GetString(),NULL, NULL);
      if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
      n=imaqReadFile(this->pi[i],s4.GetString(),NULL, NULL);
      if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

      fgets(fs,299,fp);
      this->im_sym[i] = atoi(fs);
      if (this->im_sym[i]==1)
            {
            this->oei[i]=imaqCreateImage(IMAQ_IMAGE_U8,5);
            fgets(fs,299,fp);fs[strlen(fs)-1]='\0';s1=fs;
            s2="e_";s2=s2+s1;s2=s2+".bmp";
            s2=str+s2;
            n=imaqReadFile(this->oei[i],s2.GetString(),NULL, NULL);
            if (n==0) {this->errmsg =
imaqGetErrorText(imaqGetLastError()); return(-1);}
            }
      }

n = this->find_proto_info();
return(n);
}


int CMyImage::find_proto_info()
{
int i,j;
int ww,hh;
ParticleReport* pr;
float x,y,area,mx,my,pmr;
int n,num;

j=0;
for(i=0;i<this->num_proto;i++)
      {
      pr = imaqGetParticleInfo(this->bi[i],TRUE,IMAQ_ALL_INFO,&num);
```

```
      if (pr==NULL) {this->errmsg =
imaqGetErrorText(imaqGetLastError()); return(-1);}

      n=imaqCalcCoeff(this->bi[i],&pr[j],IMAQ_AREA,&area);
      if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
      this->ar[i]=area;

      n=imaqCalcCoeff(this->bi[i],&pr[j],IMAQ_CENTER_MASS_X,&x);
      if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
      this->cx[i]=x;

      n=imaqCalcCoeff(this->bi[i],&pr[j],IMAQ_CENTER_MASS_Y,&y);
      if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
      this->cy[i]=y;

      n=imaqCalcCoeff(this->bi[i],&pr[j],IMAQ_PERIMETER,&pmr);
      if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
      this->pm[i]=pmr;

      n=imaqCalcCoeff(this->bi[i],&pr[j],IMAQ_INERTIA_XX,&mx);
      if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
      n=imaqCalcCoeff(this->bi[i],&pr[j],IMAQ_INERTIA_YY,&my);
      if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
      this->mz[i] = mx + my;

      imaqGetImageSize(this->pi[i],&ww,&hh);
      this->sw[i]=(float)ww;
      this->sh[i]=(float)hh;

      imaqDispose(pr);
      }

return(0);
}

//// --------------- LOGIC BELOW ----------------------
//



int CMyImage::Rotate_to_Horiz(void)
{
// align image to horizontal ....

PixelValue pp;
ParticleReport* pr;
float deg,ps,x,y,max_len;
int j,n,num;
Rect bbox;

pr = imaqGetParticleInfo(this->binary_image,TRUE,IMAQ_ALL_INFO,&num);
```

```
if (pr==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

if (num==0) return(-2);

ps=0.0f;j=0;
for(n=0;n<num;n++)
      if (pr[n].area > ps)
            {
            j=n;
            ps=(float)pr[n].area;
            }

// find out orientation and centre of mass ...

n=imaqCalcCoeff(this->binary_image,&pr[j],IMAQ_ORIENTATION,&deg);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
n=imaqCalcCoeff(this->binary_image,&pr[j],IMAQ_CENTER_MASS_X,&x);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
this->icx=x;
n=imaqCalcCoeff(this->binary_image,&pr[j],IMAQ_CENTER_MASS_Y,&y);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
this->icy=y;
n=imaqCalcCoeff(this->binary_image,&pr[j],IMAQ_MAX_INTERCEPT,&max_len);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

this->ibx = pr[j].boundingBox.left + pr[j].boundingBox.width / 2.0f;
this->iby = pr[j].boundingBox.top + pr[j].boundingBox.height / 2.0f;
bbox =
imaqMakeRect(pr[j].boundingBox.top,pr[j].boundingBox.left,pr[j].boundin
gBox.height,pr[j].boundingBox.width);

deg = this->Prepare_Angle(deg,this->icx,this->icy,&bbox,max_len);

pp.grayscale=0;
n=imaqRotate(this->main_image,this->main_image,-deg,pp,IMAQ_BILINEAR);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

return(0);
}

int CMyImage::Collect_Info(void)
{
ParticleReport* pr;
float area,x,y,pmr,mx,my;
int n,num,j;

pr = imaqGetParticleInfo(this->binary_image,TRUE,IMAQ_ALL_INFO,&num);
if (pr==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

if (num==0) return(-2);
```

```
j=0;
n=imaqCalcCoeff(this->binary_image,&pr[j],IMAQ_AREA,&area);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
this->area=area;

n=imaqCalcCoeff(this->binary_image,&pr[j],IMAQ_CENTER_MASS_X,&x);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
this->icx=x;

n=imaqCalcCoeff(this->binary_image,&pr[j],IMAQ_CENTER_MASS_Y,&y);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
this->icy=y;

n=imaqCalcCoeff(this->binary_image,&pr[j],IMAQ_PERIMETER,&pmr);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
this->perimeter=pmr;

n=imaqCalcCoeff(this->binary_image,&pr[j],IMAQ_INERTIA_XX,&mx);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
n=imaqCalcCoeff(this->binary_image,&pr[j],IMAQ_INERTIA_YY,&my);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
this->moment = mx + my;

imaqDispose(pr);
imaqFlip(this->temp_image,this->binary_image,IMAQ_HORIZONTAL_AXIS);

pr = imaqGetParticleInfo(this->temp_image,TRUE,IMAQ_ALL_INFO,&num);
if (pr==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

j=0;
n=imaqCalcCoeff(this->temp_image,&pr[j],IMAQ_CENTER_MASS_X,&x);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
this->icx2=x;

n=imaqCalcCoeff(this->temp_image,&pr[j],IMAQ_CENTER_MASS_Y,&y);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
this->icy2=y;

this->sym_x = (float)fabs(this->icx - this->icx2);
this->sym_y = (float)fabs(this->icy - this->icy2);

imaqDispose(pr);
return(0);
}

int CMyImage::Make_Prototype(CString pt,CString fn)
{
```

```
Image * II;
ThresholdData* tinfo;
PixelValue pp;
ParticleReport* pr;
HistogramReport* hr;
//SelectParticleCriteria pcrit;
int num;
int j,n;
float ps,nf=0.25;


Rect roi;
Point des;

II=imaqCreateImage(IMAQ_IMAGE_U8,5);

des.x=0;
des.y=0;

// prepare main image ...
this->PrepareImage(this->main_image,this->binary_image);
this->Rotate_to_Horiz();
this->PrepareImage(this->main_image,this->binary_image);

imaqDuplicate(II,this->main_image);

hr = imaqHistogram(II,3,0,255,this->binary_image);
if (hr==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

pp.grayscale=(hr[0].mean + hr[0].max)/2.0f;
n = imaqMinConstant(II,II,pp);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

//if (hr[0].mean!=0) pp.grayscale = 255.0f/hr[0].mean; else
pp.grayscale=1.0f;
//n = imaqMultiplyConstant(II,II,pp);
//imaqWriteBMPFile(II,"zdum.bmp",FALSE,NULL);

// calculate edge Image ...
float kern[9]={-1,-1,-1,-1,8,-1,-1,-1,-1};
n = imaqConvolve(this->edge_image,II, kern,3,3,nf,NULL);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

num=3;
tinfo= imaqAutoThreshold(this->temp_image,this-
>edge_image,num,IMAQ_THRESH_CLUSTERING);
if (tinfo==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

float xx,yy;

//xx=tinfo[num-1].rangeMin * 0.5f + tinfo[num-2].rangeMin * 0.5f;
xx=tinfo[num-2].rangeMin;
yy=tinfo[num-1].rangeMax;
```

```
n = imaqThreshold(this->temp_image,this->edge_image,xx,yy,TRUE,1.0);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

//imaqWriteBMPFile(this->temp_image,"zdum.bmp",FALSE,NULL);

ParticleFilterCriteria cr[1];

cr[0].parameter = IMAQ_AREA;
cr[0].lower = 1;
cr[0].upper = 5*sf;
cr[0].exclude = FALSE;

//cr[0].parameter = IMAQ_PARTICLE_TO_IMAGE;
//cr[0].lower = 0;
//cr[0].upper = 0.005f;
//cr[0].exclude = FALSE;

n = imaqParticleFilter(this->edge_image,this-
>temp_image,&cr[0],1,TRUE,TRUE);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

pp.grayscale = 255;
imaqMultiplyConstant(this->edge_image,this->edge_image,pp);

// prepare slightly bigger mask ...

imaqDuplicate(this->temp_image1,this->binary_image);
n = imaqMorphology(this->temp_image1,this-
>temp_image1,IMAQ_DILATE,NULL);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

pr = imaqGetParticleInfo(this->temp_image1,TRUE,IMAQ_ALL_INFO,&num);
if (pr==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

// select biggest particle ...
// should be only one particle in the image because of prepare image
...

ps=0.0f;j=0;
for(n=0;n<num;n++)
      if (pr[n].area > ps)
            {
            j=n;
            ps=(float)pr[n].area;
            }

roi.top  = pr[j].boundingBox.top;
roi.left = pr[j].boundingBox.left;
roi.height = pr[j].boundingBox.height;
roi.width = pr[j].boundingBox.width;

// create the logical prototype ...
```

```
imaqDuplicate(this->temp_image,this->binary_image);

// teach prototype ...

n = imaqSetImageSize(II,roi.width,roi.height);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

n = imaqCopyRect(II,this->temp_image,roi,des);

pp.grayscale = 200;
imaqMultiplyConstant(II,II,pp);
pp.grayscale = 50;
n = imaqAddConstant(II,II,pp);

//if (imaqLearnPattern(II,IMAQ_LEARN_ALL)==0) {this->errmsg =
imaqGetErrorText(imaqGetLastError()); return(-1);}

// write files now ....

CString ff1,ff2,ff;

// logical prototype ...
ff=fn;
ff=ff+".bmp";
ff.MakeUpper();

// visual ...
n = imaqCopyRect(II,this->main_image,roi,des);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
ff1="p_";ff2=pt;ff1=ff1+ff;ff2=ff2+ff1;
n = imaqWriteBMPFile(II,ff2.GetString(),FALSE,NULL);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

// edge ...
n = imaqCopyRect(II,this->edge_image,roi,des);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
ff1="e_";ff2=pt;ff1=ff1+ff;ff2=ff2+ff1;
n = imaqWriteBMPFile(II,ff2.GetString(),FALSE,NULL);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

// binary ...
n = imaqCopyRect(II,this->binary_image,roi,des);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
ff1="b_";ff2=pt;ff1=ff1+ff;ff2=ff2+ff1;
n = imaqWriteBMPFile(II,ff2.GetString(),FALSE,NULL);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

imaqDispose(pr);
imaqDispose(hr);
```

```
imaqDispose(II);
return(0);
}

//
////-----------------------------------------------------------------
----------
//
int CMyImage::Prepare_for_Matching(Image * I,Image * mI,Image *
eI,Image * bI)
{
//HistogramReport * hr;
ThresholdData * tinfo;
ParticleReport* pr;
int n,num,n1;
PixelValue pp;
float nf=0.25f;
Image * II;
Rect roi;
Point des;
float ps;


II=imaqCreateImage(IMAQ_IMAGE_U8,5);
imaqDuplicate(II,bI);

n = imaqMorphology(this->temp_image1,II,IMAQ_DILATE,NULL);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

// select the largest particle ...
pr = imaqGetParticleInfo(this->temp_image1,TRUE, IMAQ_BASIC_INFO,&n);
if (pr==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

if (n==0) return(-2);

num=0;
ps = (float)pr[num].area;
for(n1=0;n1<n;n1++)
      {
      if (pr[n1].area > ps)
            {
            ps=(float)pr[n1].area;
            num = n1;
            }
      }

roi.top  = pr[num].boundingBox.top;
roi.left = pr[num].boundingBox.left;
roi.height = pr[num].boundingBox.height;
roi.width = pr[num].boundingBox.width;

imaqSetImageSize(bI,roi.width,roi.height);
imaqSetImageSize(mI,roi.width,roi.height);

des.x=0;
```

```
des.y=0;

n = imaqCopyRect(bI,II,roi,des);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

n = imaqCopyRect(mI,I,roi,des);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

//hr = imaqHistogram(mI,3,0,255,bI);
//if (hr==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

//pp.grayscale = hr[0].mean;
//n = imaqMinConstant(mI,mI,pp);
//if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

//imaqDispose(pr);

// calculate edge Image ...
float kern[9]={-1,-1,-1,-1,8,-1,-1,-1,-1};
n = imaqConvolve(eI,mI, kern,3,3,nf,NULL);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

num=3;
tinfo = imaqAutoThreshold(this-
>temp_image,eI,num,IMAQ_THRESH_CLUSTERING);
if (tinfo==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

float xx,yy,ff=0.8f;

//xx=(tinfo[num-1].rangeMin + tinfo[num-2].rangeMin)/2.0f;
//xx=tinfo[num-2].rangeMin;
xx=tinfo[num-2].rangeMin * ff + tinfo[num-1].rangeMin * ( 1.0f - ff);
yy=tinfo[num-1].rangeMax;

n = imaqThreshold(this->temp_image,eI,xx,yy,TRUE,1.0);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

imaqDuplicate(eI,this->temp_image);

//pr = imaqGetParticleInfo(this->temp_image,TRUE,IMAQ_ALL_INFO,&iii);
//if (pr==NULL ){this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

ParticleFilterCriteria cr[2];
int n_cr=2;

//cr[0].parameter = IMAQ_AREA;
//cr[0].lower = 1;
//cr[0].upper = 5*sf;
//cr[0].exclude = FALSE;
```

```
//cr[0].parameter = IMAQ_NUM_HOLES;
//cr[0].exclude = FALSE;
//cr[0].lower=0;
//cr[0].upper=1;
//
//cr[1].parameter = IMAQ_AREA_OF_HOLES;
//cr[1].exclude = FALSE;
//cr[1].lower=0;
//cr[1].upper=1;
//
//iii = imaqParticleFilter(this->edge_image,this-
>edge_image,cr,2,TRUE,TRUE);
//if (pr==NULL ){this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

cr[0].parameter = IMAQ_PARTICLE_TO_IMAGE;
cr[0].lower = 0;
cr[0].upper = 0.01f;
cr[0].exclude = FALSE;

cr[1].parameter = IMAQ_ELONGATION;
cr[1].lower = 0;
cr[1].upper = 2;
cr[1].exclude = FALSE;

n = imaqParticleFilter(eI,this->temp_image,cr,n_cr,TRUE,FALSE);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

//imaqSizeFilter(eI,eI,TRUE,2,IMAQ_KEEP_LARGE,NULL);

pp.grayscale = 255;
imaqMultiplyConstant(eI,eI,pp);

//imaqWriteBMPFile(eI,"z_edge.bmp",FALSE,NULL);
//imaqWriteBMPFile(II,"z_auto.bmp",FALSE,NULL);

if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

imaqDispose(II);
imaqDispose(pr);
return(0);
}

//-------------------------------------------------------------------
--------

int CMyImage::PrepareImage(Image * I,Image * BI)
{
int n,n1,num;
float ps,xx,yy;
ThresholdData* tinfo;
ParticleReport* pr;
HistogramReport* hr;
PixelValue pp;
```

```
float m1,m2,m3;

//
//n = imaqEqualize(I,I,0.0f,255.0f,NULL);
//if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
hr = imaqHistogram(I,3,0.0f,255.0f,NULL);
if (hr==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

m1 = hr->max;
m2 = hr->min;
m3 = 255 / (m1 - m2);

pp.grayscale = m2;
n = imaqSubtractConstant(I,I,pp);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

pp.grayscale = m3;
n = imaqMultiplyConstant(I,I,pp);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}


// calculate Binary Image ...
tinfo= imaqAutoThreshold(BI,I,2,IMAQ_THRESH_CLUSTERING);
if (tinfo==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

xx=tinfo->rangeMin+(tinfo->rangeMax-tinfo->rangeMin)*this->bin_th;
yy=255;

n = imaqThreshold(BI,I,xx,yy,TRUE,1.0);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}


// fill holes ...
n = imaqFillHoles(BI,BI,TRUE);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

// remove all particles touching border ...
n = imaqRejectBorder(BI,BI,TRUE);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

// select the largest particle ...
pr = imaqGetParticleInfo(BI,TRUE, IMAQ_ALL_INFO,&n);
if (pr==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

if (n>0)
      {
      num=0;
      ps = (float)pr[num].area;
```

```
	for(n1=0;n1<n;n1++)
		{
		if (pr[n1].area > ps)
			{
			ps=(float)pr[n1].area;
			num = n1;
			}
		}


	n = imaqCalcCoeff(BI,&pr[num],IMAQ_CENTER_MASS_X,&xx);
	if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

	n = imaqCalcCoeff(BI,&pr[num],IMAQ_CENTER_MASS_Y,&yy);
	if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

	// take out all other particles ...
	Point pt;

	pt.x = (int)xx;
	pt.y = (int)yy;

	n = imaqMagicWand(BI,BI,pt,0.0f,TRUE,1.0f);
	if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

	// mask the main image ...
	n = imaqMultiply(I,I,BI);
	if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}
	}
else
	{
	return(-2);
	}

imaqDispose(pr);
imaqDispose(hr);
return(0);
}


float CMyImage::Prepare_Angle(float deg,float mx,float my,Rect *
bb,float ll)
{
float fin_deg;
float degr;
PointFloat pf1,pf2;
int pr1,pr2;

ll=ll/2.0f;
fin_deg=deg;

if (fin_deg < 90.0)
	{
```

```
      degr=fin_deg*(float)PI/180.0f;

      pf1.x = mx + ll*cosf(degr);
      pf1.y = my - ll*sinf(degr);

      pf2.x = mx - ll*cosf(degr);
      pf2.y = my + ll*sinf(degr);

      pr1=In_Rectangle(pf1,bb);
      pr2=In_Rectangle(pf2,bb);

      while((pr1==1)&&(pr2==1))           // both inside ... increase
length ...
            {
            ll++;
            pf1.x = mx + ll*cosf(degr);
            pf1.y = my - ll*sinf(degr);

            pf2.x = mx - ll*cosf(degr);
            pf2.y = my + ll*sinf(degr);

            pr1=In_Rectangle(pf1,bb);
            pr2=In_Rectangle(pf2,bb);
            }

      while((pr1==0)&&(pr2==0))           // both outside ... decrease
length ...
            {
            ll--;
            pf1.x = mx + ll*cosf(degr);
            pf1.y = my - ll*sinf(degr);

            pf2.x = mx - ll*cosf(degr);
            pf2.y = my + ll*sinf(degr);

            pr1=In_Rectangle(pf1,bb);
            pr2=In_Rectangle(pf2,bb);
            }


      if ((pr1==0)&&(pr2==1)) fin_deg=fin_deg;              // proper
orientation ...
      if ((pr1==1)&&(pr2==0)) fin_deg=fin_deg+180.0f;       // head on
other side ...

      }
else
      {
      degr=(180.0f-fin_deg)*(float)PI/180.0f;

      pf1.x = mx + ll*cosf(degr);
      pf1.y = my + ll*sinf(degr);

      pf2.x = mx - ll*cosf(degr);
      pf2.y = my - ll*sinf(degr);

      pr1=In_Rectangle(pf1,bb);
```

```
      pr2=In_Rectangle(pf2,bb);

      while((pr1==1)&&(pr2==1))           // both inside ... increase
length ...
            {
            ll++;
            pf1.x = mx + ll*cosf(degr);
            pf1.y = my + ll*sinf(degr);

            pf2.x = mx - ll*cosf(degr);
            pf2.y = my - ll*sinf(degr);

            pr1=In_Rectangle(pf1,bb);
            pr2=In_Rectangle(pf2,bb);
            }

      while((pr1==0)&&(pr2==0))           // both outside ... decrease
length ...
            {
            ll--;
            pf1.x = mx + ll*cosf(degr);
            pf1.y = my + ll*sinf(degr);

            pf2.x = mx - ll*cosf(degr);
            pf2.y = my - ll*sinf(degr);

            pr1=In_Rectangle(pf1,bb);
            pr2=In_Rectangle(pf2,bb);
            }

      if ((pr1==0)&&(pr2==1)) fin_deg=fin_deg+180.0f;       // head on
other side ...
      if ((pr1==1)&&(pr2==0)) fin_deg=fin_deg;              // proper
orientation ...
      }

return(fin_deg);
}

int CMyImage::In_Rectangle(PointFloat pp,Rect * bb)
{
int res=0;
if ((pp.x >= bb->left)&&(pp.x <= (bb->left+bb->width))&&(pp.y >= bb-
>top) && (pp.y <= (bb->top+bb->height))) res=1;
return(res);
}


// ---------------------------------------------------------------
int CMyImage::Process_Image()
{
float * ers;
float e1,e2,e3,e4;
int i,j,ec;
int c1,c2;
float tot_tol;
int iii;
```

```
// PRI

// prepare image ...
//if (this->write_images==1)
//    {
//    imaqWriteBMPFile(this->main_image,this-
>im_name.GetString(),FALSE,NULL);
//    }
this->piece_found=-1;
this->im_count++;
this->is_dirty = -2;

this->area = -1;
this->moment = -1;
this->perimeter = -1;

//printf("\n [%d : %04d : %d]",this->piece_found,this->im_count,this-
>is_dirty);

iii=this->PrepareImage(this->main_image,this->binary_image);
if (iii==-1) return(-1);
if (iii==-2)
      {
      this->piece_found=-1;
      return(0);
      }
if (this->Rotate_to_Horiz()==-1) return(-1);
if (this->PrepareImage(this->main_image,this->binary_image)==-1)
return(-1);
if (this->Prepare_for_Matching(this->main_image,this->match_image,this-
>edge_image,this->binary_image)==-1) return(-1);
//
//// -- we are not interested in edges which do not have holes ....

//ParticleFilterCriteria cr[2];
//cr[0].parameter = IMAQ_NUM_HOLES;
//cr[0].exclude = FALSE;
//cr[0].lower=0;
//cr[0].upper=1;
//
//cr[1].parameter = IMAQ_AREA_OF_HOLES;
//cr[1].exclude = FALSE;
//cr[1].lower=0;
//cr[1].upper=1;
//
//iii = imaqParticleFilter(this->edge_image,this-
>edge_image,cr,2,TRUE,TRUE);
//if (iii==0){this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

CString ss,s1;
s1.Format("%04d.bmp",im_count);

//ss=this->im_path + "main";
//ss=ss+s1;
//imaqWriteBMPFile(this->main_image,ss.GetString(),FALSE,NULL);
```

```
ss=this->im_path + "edge";
ss=ss+s1;
imaqWriteBMPFile(this->edge_image,ss.GetString(),FALSE,NULL);


iii=this->Collect_Info();
if (iii==-1) return(-1);

// evaluate feature set ....
ers = new float[this->num_proto*4];
ec=0;
tot_tol = (this->area_tol + this->peri_tol + this->moment_tol)/3.0f *
this->soft_tol_scale;

//printf("\nImage : %s",this->im_name.GetString());
if (iii==0)
      {
      for(i=0;i<this->num_proto;i++)
            {
            CString ss;

            e1=(float)fabs(this->ar[i]-this->area)/this->ar[i]*100.0f;
            e2=(float)fabs(this->mz[i]-this->moment)/this-
>mz[i]*100.0f;
            e3=(float)fabs(this->pm[i]-this->perimeter)/this-
>pm[i]*100.0f;
            e4=e1+e2+e3;
            //printf("\n\t## %.3f  %.3f  %.3f  %.3f",e1,e2,e3,e4*1.5);
            //
            //ss.Format("%s : %.2f , %.2f , %.2f",this-
>pnames[i],e1,e2,e3);
            //AfxGetMainWnd()->MessageBox(ss.GetString());
            c1=0;
            c2=0;
            if ((e1<=this->area_tol)&&(e2<=this-
>moment_tol)&&(e3<=this->peri_tol)) c1=1;
            if (e4 <= tot_tol) c2=1;
            if ((c1==1)||(c2==1))
                  {
                  ers[ec*4]=e1;
                  ers[ec*4+1]=e2;
                  ers[ec*4+2]=e3;
                  ers[ec*4+3]=(float)i;
                  ec++;
                  }
            }

      //this->piece_found = -1;
      if (ec>0)
            {
            j=(int)ers[3];
            int mf=0;
            for(i=0;i<ec;i++)
                  {
```

```
                  if
((ers[i*4]<ers[mf*4])&&(ers[i*4+1]<ers[mf*4+1])&&(ers[i*4+2]<ers[mf*4+2
]))
                        {
                        mf = i;
                        j=(int)ers[i*4+3];
                        }
                  }
            this->piece_found=j;
            }
      }

if ((this->piece_found!=-1)&&(this->inspect==1))
      {
      // inspect call ...
      if (this->im_sym[this->piece_found]==1)
            {
            float d1;
            float d2;
            float mdx1,mdy1,mdx2,mdy2;
            mdx1 = (this->cx[this->piece_found] - this->icx);
            mdy1 = (this->cy[this->piece_found] - this->icy);
            d1=mdx1*mdx1 + mdy1*mdy1;

            mdx2 = (this->cx[this->piece_found] - this->icx2);
            mdy2 = (this->cy[this->piece_found] - this->icy2);
            d2=mdx2*mdx2 + mdy2*mdy2;

            //CString ss;
            //ss.Format("%f,%f && %f,%f",mdx1,mdy1,mdx2,mdy2);
            //AfxGetMainWnd()->MessageBox(ss.GetString());

            if (d1<d2)
                  {
                  this->sym_x = (float)fabs(mdx1)/2;
                  this->sym_y = (float)fabs(mdy1)/2;
                  if (this->Inspect_Image(this->ei[this-
>piece_found])==-1) return(-1);
                  }
            else
                  {
                  this->sym_x = (float)fabs(mdx2)/2;
                  this->sym_y = (float)fabs(mdy2)/2;
                  if (this->Inspect_Image(this->oei[this-
>piece_found])==-1) return(-1);
                  }

            }
      else
            {
            if (this->Inspect_Image(this->ei[this->piece_found])==-1)
return(-1);
            }
      }

return(0);
}
```

```
int CMyImage::Inspect_Image(Image * pe)
{
int isdirty=0;
unsigned char *ea,*ia,*ba;
PixelValue pp;
int i,j,a,b;
int ww,hh;
int w,h;
int pid;
int n1,ngsl,ngl;
int stx,stpx,sty,stpy;
int n;

Rect rr;
Image *tI,*zI;

tI  = imaqCreateImage(IMAQ_IMAGE_U8,5);
zI  = imaqCreateImage(IMAQ_IMAGE_U8,5);

// resize image ...

pid = this->piece_found;

imaqGetImageSize(pe,&w,&h);

rr = imaqMakeRect(0,0,h,w);
n1 = imaqResample(zI,this-
>edge_image,w,h,IMAQ_ZERO_ORDER,IMAQ_NO_RECT);

//imaqWriteBMPFile(zI,"ztest_edge.bmp",FALSE,NULL);
//imaqWriteBMPFile(pe,"zprot_edge.bmp",FALSE,NULL);
//
//pp.grayscale = 255;
//imaqDivideConstant(pe,pe,pp);
//pp.grayscale = 255;
//imaqDivideConstant(zI,zI,pp);
//
//pp.grayscale = 75;
//imaqMultiplyConstant(pe,pe,pp);
//pp.grayscale = 150;
//imaqMultiplyConstant(zI,zI,pp);
//imaqAdd(tI,zI,pe);
//imaqWriteBMPFile(tI,"zover.bmp",FALSE,NULL);


ia=(unsigned char *)imaqImageToArray(zI,IMAQ_NO_RECT,&ww,&hh);
if (ia==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

ea=(unsigned char *)imaqImageToArray(pe,IMAQ_NO_RECT,&ww,&hh);
if (ea==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

ba=(unsigned char *)imaqImageToArray(pe,IMAQ_NO_RECT,&ww,&hh);
```

```
if (ba==NULL) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}


list <Point> elist,dlist;
list <Point>::iterator tpt;
Point p1,p2;
int done=0;//,oi,oj;
long count=0;

for(i=0;i<ww;i++)
for(j=0;j<hh;j++)
      {
      if (ia[i+j*ww]!=0) count++;
      ba[i+j*ww]=0;
      }

for(i=0;i<ww;i++)
for(j=0;j<hh;j++)
      if ((ea[i+j*ww]!=0)&&(ia[i+j*ww]==0))
            ia[i+j*ww]=100;

i=0;j=0;

if (this->sym_x > this->sym_y)
      {
      ngsl = (int)ceil(this->sym_x * this->sym_fact);
      }
else
      {
      ngsl = (int)ceil(this->sym_y * this->sym_fact);
      }

if (ngsl==0) ngsl=1;

int nn_1 = (int)ceil(src_wd * this->sym_fact);

if (nn_1 > ngsl) ngsl = nn_1;

//CString ss;
//ss.Format("%f , %f - %d",this->sym_x,this->sym_y,ngsl);
//AfxGetMainWnd()->MessageBox(ss.GetString());

ngl = ngsl;

// find out all points to be examined ...
while(done==0)
      {
      if (ia[i+j*ww]==255)
            {
            p1.x=i;
            p1.y=j;
            elist.push_back(p1);
            ia[i+j*ww]=200;   //already taken ...
            count--;
            stx=i-ngsl;if (stx<0) stx=0;
            stpx=i+ngsl;if (stpx>=ww) stpx=ww-1;
```

```
            sty=j-ngsl;if (sty<0) sty=0;
            stpy=j+ngsl;if (stpy>=hh) stpy=hh-1;
            for(a=stx;a<=stpx;a++)
            for(b=sty;b<=stpy;b++)
                  if (ia[a+b*ww]==255)
                        {
                        i=a;
                        j=b;
                        a=stpx+1;
                        b=stpy+1;
                        }
            }
      else
            {
            for(a=0;a<ww;a++)
            for(b=0;b<hh;b++)
                  if (ia[a+b*ww]==255)
                        {
                        i=a;
                        j=b;
                        a=ww;
                        b=hh;
                        }
            }
      if (count==0) done=1;
      }

// edges will be in 200 ....


n=imaqArrayToImage(zI,ia,ww,hh);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

int dcx,dcy,dx,dy;
int nf;

pp.grayscale=1;
dcx=0;
dcy=0;
isdirty=0;

tpt=elist.begin();
p2.x=((Point)(*tpt)).x;
p2.y=((Point)(*tpt)).y;

for(tpt=elist.begin();tpt!=elist.end();tpt++)   // process whole list
...
      {
      p1.x=((Point)(*tpt)).x;
      p1.y=((Point)(*tpt)).y;

      dx=abs(p1.x-p2.x);
      dy=abs(p1.y-p2.y);

      nf=0;
```

```
	if ((dx>ngsl)||(dy>ngsl))
		{
		stx = p1.x - ngl + dcx;
		stpx = p1.x + ngl + dcx;

		sty = p1.y - ngl + dcy;
		stpy = p1.y + ngl + dcy;

		if (stx<0) stx=0;
		if (stpx>=ww) stpx=ww-1;

		if (sty<0) sty=0;
		if (stpy>=hh) stpy=hh-1;

		for(a=stx;a<=stpx;a++)
		for(b=sty;b<=stpy;b++)
			if (ea[a+b*ww]!=0)
				{
				dcx=a-p1.x;
				dcy=b-p1.y; // update dcx , dcy ...

				p2.x=p1.x;
				p2.y=p1.y;

				nf=1;

				a = stpx+1;
				b = stpy+1;
				}

		if (nf==0) // might not be following patterns ... so try
again ...
			{dcx=0;dcy=0;}

		}	// discontinuous edge ....

	if (nf==0)
		{
		stx = p1.x - ngl + dcx;
		stpx = p1.x + ngl + dcx;

		sty = p1.y - ngl + dcy;
		stpy = p1.y + ngl + dcy;

		if (stx<0) stx=0;
		if (stpx>=ww) stpx=ww-1;

		if (sty<0) sty=0;
		if (stpy>=hh) stpy=hh-1;

		for(a=stx;a<=stpx;a++)
		for(b=sty;b<=stpy;b++)
			if (ea[a+b*ww]!=0)
				{
				dcx=a-p1.x;
				dcy=b-p1.y; // update dcx , dcy ...
```

```
                        p2.x=p1.x;
                        p2.y=p1.y;

                        nf=1;

                        a = stpx+1;
                        b = stpy+1;
                        }
                }

        if (nf!=1)  // dirty ...
                {
                i = (int)p1.x;
                j = (int)p1.y;
                ba[i+j*ww]=255;
                isdirty=1;
                }

        }

n=imaqArrayToImage(tI,ba,ww,hh);
if (n==0) {this->errmsg = imaqGetErrorText(imaqGetLastError());
return(-1);}

imaqDispose(ia);
imaqDispose(ea);
imaqDispose(ba);


if (isdirty==1)
        {
        ParticleReport * pr;int num;
        pr = imaqGetParticleInfo(tI,TRUE,IMAQ_ALL_INFO,&num);
        ParticleFilterCriteria cr[1];
        //cr[0].parameter = IMAQ_MAX_INTERCEPT;
        //cr[0].lower = 1;
        //cr[0].upper = 2*sf;
        //cr[0].exclude = FALSE;

        //cr[0].parameter = IMAQ_PERIMETER;
        //cr[0].lower = 1;
        //cr[0].upper = 4*sf;
        //cr[0].exclude = FALSE;

        cr[0].parameter = IMAQ_AREA;
        cr[0].lower = 0;
        cr[0].upper = 4*sf;
        cr[0].exclude = FALSE;

        imaqParticleFilter(tI,tI,&cr[0],1, TRUE,TRUE);
        pr = imaqGetParticleInfo(tI,TRUE,IMAQ_BASIC_INFO,&num);

        if (num==0) isdirty=0;
                else
                {
                imaqAdd(zI,zI,tI);
                }
```

```
      }


if (!elist.empty()) elist.clear();
if (!dlist.empty()) dlist.clear();

if ((this->write_images == 1)&&(isdirty==1))
      {
      CString ss;
      ss.Format("z_%s.bmp",this->im_name.GetString());
      ss = this->im_path + ss;
      imaqWriteBMPFile(zI,ss.GetString(),FALSE,NULL);
      }

this->is_dirty = isdirty;

imaqDispose(tI);
imaqDispose(zI);
return(0);
}



void draw_plus(Image * I,float dcx,float dcy,int dx,int dy,float col)
{
Point p1,p2,p3,p4;

p1.x=(int)dcx-dx;
p1.y=(int)dcy;

p2.x=(int)dcx+dx;
p2.y=(int)dcy;

imaqDrawLineOnImage(I,I,IMAQ_DRAW_VALUE,p1,p2,col);

p3.x=(int)dcx;
p3.y=(int)dcy-dy;

p4.x=(int)dcx;
p4.y=(int)dcy+dy;

imaqDrawLineOnImage(I,I,IMAQ_DRAW_VALUE,p3,p4,col);
}
```

File : main.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <process.h>
#include <conio.h>
#include <time.h>

#include "ImgAn.h"

//int prog_stat;
char con_file[]="c:\\temp_im\\config\\config.txt";
char con_path[]="c:\\temp_im\\config\\";
char tt_file[]="c:\\temp_im\\forke.bmp";
char proto_file[200];

struct belt_data belt;
CMyImage * img1;

INTERFACE_ID int_id;
SESSION_ID ssn_id;
long sig_count=0;
long im_count;
int sys_busy=0;
int waiting;
int need_pr;
int err_oc;
int service_ret;
int kp;
int needs_del=-1;
int prog_stat=0;
int belt_control;
int trig_count;

clock_t last_trig,curr_trig;
double min_trig,trig_dur,max_trig;
double belt_acq;
double belt_tol;
double min_trig_th;

clock_t tstart,tstop;
int first_trig;

Image ** im;
Rect acqRect;
FILE * mfp;

int belt_calibrate(void);
int load_prototypes(void);
int make_prototype(void);
```

```
int process_pieces(void);
int stop_process(void);
int read_config_file(void);
int write_config_file(void);
int process_pieces(void);


void errChk(IMG_ERR err);
void my_sleep(clock_t wait );
uInt32 process_routine(SESSION_ID sid, IMG_ERR err, uInt32 signal,
void* userdata);
uInt32 calibrate_routine(SESSION_ID sid, IMG_ERR err, uInt32 signal,
void* userdata);
void closing_routines();
int starting_routines();

void main()
{
int done=0;
char ch,ch1;
int retval;

img1= new CMyImage();
img1->inspect=1;
img1->write_images=1;
belt_control=0;
min_trig_th = 0.8;
belt.belt_acq_delay = 0.05;

while(!done)
      {
      if ((kbhit())||(prog_stat<0))
            {
            ch=getch();
            if (ch==27)
                  {
                  done=1;
                  closing_routines();
                  }
            else
                  {
                  prog_stat=-prog_stat;
                  }
            }
      //done=1;

      if (prog_stat==0)
            {
            printf("\n>> Performing STARTUP routines ...");
            if (starting_routines()!=-1)
                  {
                  if (belt_control==1)
                        {
                        prog_stat=1;
                        }
                  else
                        {
```

```
                        prog_stat=2;
                        }
                  }
            else
                  {
                  done=1;
                  break;
                  }
            }

      if (prog_stat==1)       // calibrate belt ...
            {
            printf("\n>> START BELT & ENCODER !! Press ENTER to
continue ...");
            ch1=0;
            while((ch1!=13)&&(ch1!=27))
                  {
                  ch1=getch();
                  }
            if (ch1!=27)
                  {
                  retval = belt_calibrate();
                  if (retval!=-1) prog_stat=2;
                  }
            else
                  {
                  printf("\n>> To QUIT press ESC again !!");
                  prog_stat=-prog_stat;
                  }
            }

      if (prog_stat==2)
            {
            printf("\n>> L - Load Prototypes | M - Make Prototypes
<M/L> : ");

            ch1=0;

      while(!((ch1=='m')||(ch1=='M')||(ch1=='L')||(ch1=='l')||(ch1==27)
))
                  {
                  ch1=getch();
                  }

            if ((ch1=='l')||(ch1=='L'))
                  {
                  CString ss=proto_file;
                  if (img1->Load_Proto(ss)==-1)
                        {
                        printf("\n>> ERROR : %s",img1-
>errmsg.GetString());
                        }
                  else
                        {
                        printf("\n>> Prototypes Loaded successfully
!!");
                        prog_stat=3;
```

```
                        }
                  }

            if ((ch1=='M')||(ch1=='m'))
                  {
                  char ss[300],nm[300];
                  CString s1;
                  printf("\n>> Prototype File (including path) : ");
                  fflush(stdin);
                  gets(ss);
                  printf(">> Enter Prototype Name. Do not give
extensions : ");
                  fflush(stdin);
                  gets(nm);
                  if (img1->Load_Image(ss)==-1)
                        {
                        printf("\n>> ERROR : %s",img1-
>errmsg.GetString());
                        }
                  else
                        {
                        CString pt;
                        s1=ss;
                        int pos=s1.ReverseFind('\\');
                        if (pos==-1) pt="";
                        else pt=s1.Left(pos+1);
                        //printf("\n[%s]\n[%s]",pt.GetString(),nm);
                        if (img1->Make_Prototype(pt,nm)==0)
                              {
                              printf("\n>> Prototype successfully
made.");
                              }
                        else
                              {
                              printf("\n>> ERROR : %s !!",img1-
>errmsg.GetString());
                              }
                        }
                  }

            if (ch1==27)
                  {
                  printf("\n>> To QUIT press ESC again !!");
                  prog_stat=-prog_stat;
                  }
            }

      if (prog_stat==3)       // process pieces ....
            {
                  /// AAAAAAAA
            printf("\n>> Press ENTER to begin processing pieces ...");
            ch1=0;
            while((ch1!=13)&&(ch1!=27))
                  {
                  ch1=getch();
                  }
            if (ch1!=27)
```

```
                  {
                  CString ss;
                  ss=con_path;
                  ss=ss+"results.txt";
                  printf("\n>> Feed pieces ...");
                  mfp=fopen(ss.GetString(),"r");
                  if (mfp!=NULL)
                        {
                        __time64_t long_time;
                        struct tm * ttm;

                        fclose(mfp);
                        _time64( &long_time );
                        ttm = _localtime64( &long_time );

                        CString st;
                        st.Format("RR_%02d%02d_%02d%02d%02d.txt",ttm-
>tm_mon,ttm->tm_mday,ttm->tm_hour,ttm->tm_min,ttm->tm_sec);
                        st=con_path+st;
                        rename(ss.GetString(),st.GetString());
                        }
                  mfp=fopen(ss.GetString(),"w");
                  process_pieces();
                  fclose(mfp);
                  prog_stat=4;

                  printf("\n\n>> Press ENTER to continue ... !!");
                  ch=0;
                  while((ch!=13)&&(ch!=27))
                        {
                        ch=getch();
                        }
                  if (ch==27)
                        {
                        printf("\n>> To QUIT press ESC again !!");
                        prog_stat=-prog_stat;
                        }
                  }
            else
                  {
                  printf("\n>> To QUIT press ESC again !!");
                  prog_stat=-prog_stat;
                  }
            }
      if (prog_stat==4)
            {
            closing_routines();
            prog_stat=0;
            }
      if (prog_stat==5)
            {
            // debug routine ...
            clock_t t1,t2;
            img1->Load_Proto(proto_file);
            img1->Load_Image(tt_file);
            img1->im_name = "test.bmp";
            t1=clock();
```

```
            img1->Process_Image();
            t2=clock();
            printf("\nTook : %f",(double)(t2-t1)/CLOCKS_PER_SEC);
            done=1;
            }

      // end of while ...

      }
printf("\n\n>> TERMINATING PROGRAM ...");
//closing_routines();
ch=getch();
delete img1;
}

int belt_calibrate(void)
{
int done=0;
double del_err1,del_err2;
char ch;
int choice;
float ff;
CString ss;

im_count=0;
first_trig=-1;
service_ret=1;

ss=con_path;
ss=ss+"belt.txt";
mfp = fopen(ss.GetString(),"w");
//printf("\nStart Encoder & Belt ... ");

printf("\n>> CALIBRATION STARTED !!");
last_trig=clock();

// capture trigger .......
errChk(imgSessionWaitSignalAsync(ssn_id,IMG_EXT_TRIG2,IMG_TRIG_POLAR_AC
TIVEH,calibrate_routine,NULL));
while(!done)
      {
      if (im_count >= 15) {done=1;service_ret=0;}
      }

// end trigger ....
errChk(imgSessionTriggerClear(ssn_id));
fprintf(mfp,"\nMIN : %f",min_trig);
fprintf(mfp,"\nMAX : %f",max_trig);
fclose(mfp);
printf("\n>> CALIBRATION ENDED !!");

del_err1 = fabs(min_trig - belt.del_time)/belt.del_time * 100.0;
del_err2 = fabs(max_trig - belt.belt_max_time)/belt.belt_max_time *
100.0;
if ((del_err1 > belt.belt_tol)||(del_err2 > belt.belt_tol)) // belt
speed has changed too much ...
      {
```

```
      printf("\n\n>> Belt Speed has changed. \n>> Do you want to set
current belt speed as default ? <Y/N> : ");
      ch=getch();
      if ((ch=='y')||(ch=='Y')) choice=1; else choice=0;
      if (choice==1)
            {
            printf("\nEnter corresponding belt speed : ");
            scanf("%f",&ff);
            belt.belt_speed = (double)ff;
            belt.del_time = min_trig;
            belt.belt_max_time = max_trig;
            write_config_file();
            return(0);
            }
      else
            {
            return(-1);
            }
      }
else
      {
      printf("\n>> Belt Speed within tolerance.");
      }

return(0);
}

int starting_routines()
{
if (read_config_file()==-1)
      {
      printf("\nERROR opening CONFIG.TXT !!");
      return(-1);
      }
else
      {

      // open interface ...

      // ---- remember .......
      errChk(imgInterfaceOpen("img0",&int_id));
      errChk(imgSessionOpen(int_id,&ssn_id));
      //prog_stat=1;
      //mfp = fopen("results.txt","w");
      //img1 = new CMyImage();
      //needs_del=1;
      }
return(1);
}

void closing_routines()
{
//if (needs_del==1)
//    {
//    delete img1;
//    img1=NULL;
//    needs_del=0;
```

```
//    }
imgClose(int_id,TRUE);
}

int read_config_file(void)
{
FILE * fp;
char s1[100],s2[100],s3[100],s4[100];


fp=fopen(con_file,"r");
if (fp==NULL) return(-1);

fgets(s1,99,fp);
fgets(s2,99,fp);
fgets(s3,99,fp);
fgets(s4,99,fp);
fgets(proto_file,199,fp);
proto_file[strlen(proto_file)-1]='\0';

belt.belt_speed=atof(s1);
belt.del_time = atof(s2);
belt.belt_max_time = atof(s3);
belt.belt_tol = atof(s4);

// belt wait delay ...
//belt.belt_acq_delay = 0.4;
//belt.belt_acq_delay=0.1;
belt_acq=belt.belt_acq_delay;

acqRect.left=30;
acqRect.top=0;
acqRect.width=1024 - 2*30;
acqRect.height=500;

fclose(fp);
return(1);
}

int write_config_file(void)
{
FILE * fp;

fp=fopen(con_file,"w");
fprintf(fp,"%f\n",belt.belt_speed);
fprintf(fp,"%f\n",belt.del_time);
fprintf(fp,"%f\n",belt.belt_max_time);
fprintf(fp,"%f\n",belt.belt_tol);
fprintf(fp,"%s\n",proto_file);

fprintf(fp,"\n\n\nBELT SPEED\nMIN DELAY TIME\nMAX DELAY TIME\nSPEED
TOLERANCE\nPROTO FILE");
fclose(fp);
return(0);
}

int process_pieces(void)
```

```
{
// PP

//clock_t t1,t2;
//double dd;
int done,i;
CString ss;
char ch;

im_count=0;
service_ret=1;
need_pr=0;
//im  = &(img1->main_image);
last_trig=clock();
first_trig=0;
err_oc=0;
done=0;
max_trig = belt.belt_max_time;
min_trig = belt.del_time;
belt_tol = belt.belt_tol/100.0;
kp=0;
sys_busy=0;
trig_count=0;

fflush(stdin);
i=imgSessionWaitSignalAsync(ssn_id,IMG_EXT_TRIG2,IMG_TRIG_POLAR_ACTIVEH
,process_routine,NULL);
printf("\n>> Function Hook : %d",i);
errChk(i);

while(done==0)
      {
      //if (trig_count>100) done=1;
      //if (sys_busy==0) printf("\nSystem Idle !!");
      if (kbhit())
            {
            ch=getch();
            printf("\n>> User Abort !!");
            if (ch==27)
                  {
                  //On_Stop();
                  done=1;
                  break;
                  }
            }
      if (prog_stat!=3) {done=1;service_ret=0;break;}
      if (err_oc==1)
            {
            printf("\nERROR : Picture Snap failed !!");
            service_ret=0;
            img1->errmsg = imaqGetErrorText(imaqGetLastError());
            break;
            }
      if (err_oc==2)
            {
            service_ret=0;
            img1->errmsg = "Belt Speed changed !! Re-calibrate belt.";
```

```
            break;
            }
      }

printf("\n>> Out of Processing Loop ...");
// end trigger ....
service_ret=0;
errChk(imgSessionTriggerClear(ssn_id));
printf("\n>> Trigger released !!");

if (done==0)
      {
      printf("\n>> ERROR : %s",img1->errmsg);
      //On_Stop();
      }
return(0);
}


// ----------------------------------------

uInt32 process_routine(SESSION_ID sid, IMG_ERR err, uInt32 signal,
void* userdata)
{
if (service_ret==0) return(0);
// signal debounce ...
curr_trig=clock();
trig_dur = ((double)(curr_trig - last_trig)/ CLOCKS_PER_SEC);
if (trig_dur<min_trig_th)
      {
      last_trig=curr_trig;
      //printf("\n>> Trigger too soon ...");
      return(1);
      }
//trig_count++;
//printf("\n>>>> %d @ %f",trig_count,trig_dur);
//if (sys_busy==0)
//    {
//    sys_busy=1;
//    clock_t t1,t2;
//    t1=clock();
//    //my_sleep(3*CLOCKS_PER_SEC);
//    t2=clock();
//    printf("\n >>>> Slept for %f",(double)(t2-t1)/CLOCKS_PER_SEC);
//    sys_busy=0;
//    }
//else
//    {
//    printf("\n>>> SYstem busy ...");
//    }

////fprintf(mfp,"\nMFP : %f",curr_trig);
if (sys_busy==1)
      {
      fprintf(mfp,"\n[%f] System Busy !! Missed piece ...",trig_dur);
      last_trig=curr_trig;
      return(1);
```

```
      }
if (first_trig<=1)
      {
      first_trig++;
      last_trig=curr_trig;
      printf("\n>> %d",first_trig);
      return(1);
      }
//printf("\n>>> %d",im_count);
if (first_trig==2)
      {

      if (belt_control==1)
            {
            if ((trig_dur < (1-belt_tol) * min_trig)||(trig_dur >
(1+belt_tol) * max_trig))
                  {
                  err_oc=2;
                  //need_pr=2;
                  return(0);
                  }
            }
      if (sys_busy==0)  // not currently processing an image ...
            {
            clock_t t1,t2,t3,t4;
            double dd,dd1,dd2,dd3;
            CString ss;


            sys_busy=1;
            t1=clock();
            my_sleep((clock_t)(belt_acq*CLOCKS_PER_SEC));
            t2=clock();
            im_count++;
            //clock_t tt;
            //t1=clock();
            if (imaqSnap(sid,(img1->main_image),acqRect)==NULL)
                  {
                  err_oc=1;
                  return(0);
                  }
            t3=clock();
            // wait until picture is taken properly ...
            //if
(imgSessionWaitSignal(sid,IMG_AQ_DONE,IMG_TRIG_POLAR_ACTIVEH,10000)!=0)
            //    {
            //    err_oc=1;
            //    return(0);
            //    }
            img1->im_name.Format("Image_%04d.bmp",im_count);
            if (img1->write_images==1)
                  {
                  ss = img1->im_path + img1->im_name;
                  imaqWriteBMPFile(img1-
>main_image,ss.GetString(),FALSE,NULL);
                  }
```

```
            //printf("\nBefore %s : %d , %d",img1-
>im_name.GetString(),img1->piece_found,img1->is_dirty);
            img1->piece_found=-1;
            printf("\nIn ... ");
            if (img1->Process_Image()==-1) return(0);
            printf("Out");
            t4=clock();
            dd=(double)(t4-t1)/(CLOCKS_PER_SEC);
            dd1=(double)(t2-t1)/(CLOCKS_PER_SEC);
            dd2=(double)(t3-t2)/(CLOCKS_PER_SEC);
            dd3=(double)(t4-t3)/(CLOCKS_PER_SEC);
            fprintf(mfp,"\n%.3f\t%s",dd,img1->im_name.GetString());
            if (img1->piece_found!=-1)
                  {
                  printf("\n%s : %s",img1->im_name.GetString(),img1-
>pnames[img1->piece_found]);
                  }
            else
                  {
                  printf("\n%s : UNKNOWN ",img1->im_name.GetString());
                  }
            if (img1->piece_found>-1)
                  {
                  fprintf(mfp,"\t%s",img1->pnames[img1->piece_found]);
                  if (img1->is_dirty==1)
                        {
                        fprintf(mfp,"\tDIRTY");
                        printf("  > DIRTY [%.3f,  %.3f]",ceil(img1-
>sym_x*img1->sym_fact),ceil(img1->sym_y*img1->sym_fact));
                        }
                  else
                        {
                        fprintf(mfp,"\tCLEAN");
                        printf("  > [%.3f,  %.3f]",ceil(img1-
>sym_x*img1->sym_fact),ceil(img1->sym_y*img1->sym_fact));
                        }
                  }
            else
                  {
                  fprintf(mfp,"\tUnkn\tXXXXX");
                  }
            fprintf(mfp,"\t%.3f\t%.3f\t%.3f",dd1,dd2,dd3);
            }
      }

last_trig = curr_trig;
sys_busy=0;
return(1);  // re-instate call back ...
}


// used in calibration of belt ...
uInt32 calibrate_routine(SESSION_ID sid, IMG_ERR err, uInt32 signal,
void* userdata)
{
// signal debounce ...
curr_trig=clock();
```

```
if (((double)(curr_trig - last_trig)/ CLOCKS_PER_SEC)<min_trig_th)
return(1);
if (service_ret==0) return(0);
im_count++;
if (first_trig<0) {first_trig++;last_trig=clock();return(1);}
if (first_trig==0)
      {
      first_trig=1;
      min_trig = ((double)(curr_trig - last_trig)/ CLOCKS_PER_SEC);
      max_trig = min_trig;
      fprintf(mfp,"\n%f",min_trig);
      }
else
      {
      trig_dur = ((double)(curr_trig - last_trig)/ CLOCKS_PER_SEC);
      fprintf(mfp,"\n%f",trig_dur);
      if (trig_dur < min_trig) min_trig=trig_dur;
      if (trig_dur > max_trig) max_trig= trig_dur;
      }

last_trig = curr_trig;
return(1);
}

void my_sleep( clock_t wait )
{
clock_t goal;
goal = wait + clock();
while( goal > clock() );
}

void errChk(IMG_ERR err)
{
if (err<0)
      {
      char err_msg[1000];
      imgShowError(err,err_msg);
      printf("\nERROR : %s !!",err_msg);
      //imgClose(int_id,TRUE);
      closing_routines();
      //exit(0);
      }
}
```

APPENDIX – B

Camera Data Sheet

SPECIFICATIONS

PRODUCT SPECIFICATIONS

BASLER L100 SERIES

NEW HIGH RESPONSIVITY

Features/Benefits

- Superior image quality improves your image processing results
- Super compact size reduces the space needed in your installation
- Choice of three output types maximizes your system design flexibility
- 100% factory testing ensures consistent product quality
- LED indicators and test image generation capability reduce your integration time and aid troubleshooting
- Windows® setup tool lets you configure your camera with ease
- Electronic exposure time control provides maximum flexibility

Description

The L100 Series of digital line scan cameras is designed for industrial users who require superior image quality with high-speed data acquisition. The cameras can be triggered via an external sync signal or run in an internally controlled "free-run" mode. L100 cameras operate with a single voltage power supply and have simple cabling requirements. A combination of features such as digital shift, test images, and indicator LEDs, ensure that these versatile cameras provide an exceptional price/performance ratio.

Applications

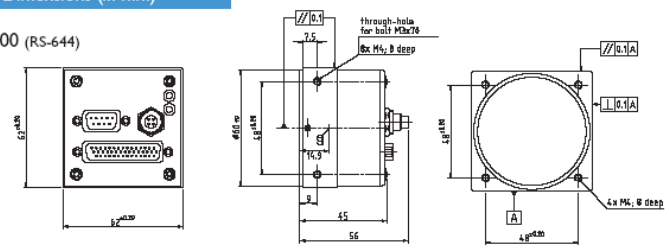
- Document scanning
- Print inspection
- Web inspection
- Production line item inspection
- OCR and 2-D bar code reading
- Many other vision applications

BASLER VISION TECHNOLOGIES

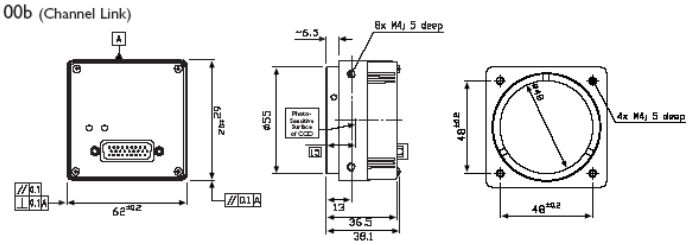
BASLER VISION COMPONENTS

Dimensions (in mm)

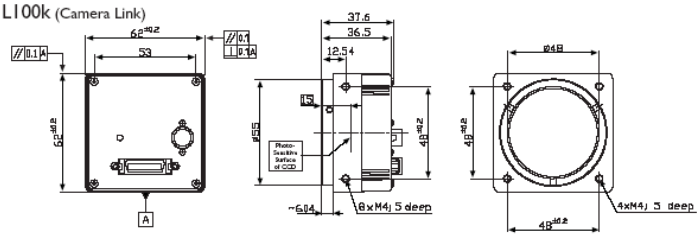
L100 (RS-644)



L100b (Channel Link)



L100k (Camera Link)



Specifications

	RS-644			Channel Link			Camera Link (Base)		
	L101	L103	L104	L101b	L103b	L104b	L101k	L103k	L104k
Sensor Size	1024 or 2048 pixels, available on all cameras								
Sensor Type	Linear CCD used on all cameras								
Pixel Size	10 µm x 10 µm with a 10 µm pitch								
Pixel Speed (MHz)	20	40	62.5	20	40	62.5	20	40	62.5
Max. Line Rate (KHz) 1k Sensor:	18.3	36.7	57.4	18.4	36.7	58.5	18.7	35.7	58.5
2k Sensor:	9.4	18.9	29.5	9.5	18.7	29.2	9.5	18.7	29.2
Video Output Formats	Single tap 8 bits or dual tap 8 bits			Single or dual tap 8 or 10 bits (selectable)			Single or dual tap 8 or 10 bits (selectable)		
Synchronization	Via external trigger or free-run*								
Exposure Control Models	Edge-controlled, level-controlled, or programmable, on all cameras								
Power Requirements	24VDC (±15%)			12VDC (±10%)			12VDC (±10%)		
max. W	6.0 W	7.0 W	8.0 W	6.0 W	8.0 W	10.0 W	6.5 W	8.5 W	10.0 W
Lens Mounts	C-mount or F-mount available for all cameras, F-mount recommended for 2k								
Housing Size (L x W x H)	45 mm x 62 mm x 62 mm			38.1 mm x 62 mm x 62 mm			37.6 mm x 62 mm x 62 mm		
Weight (with F-mount adapter)	max. 385 g			max. 270 g			max. 285 g		
Conformity	CE and FCC								

Specifications are subject to change without prior notice.

* Free-run not available on RS-644 cameras

BASLER VISION TECHNOLOGIES

Germany, Headquarters
Phone +49 4102 463 500
Fax +49 4102 463 599
vc.sales.europe@baslerweb.com

USA
Phone +1 610 280 0171
Fax +1 610 280 7608
vc.sales.usa@baslerweb.com

Singapore
Phone +65 6425 0472
Fax +65 6425 0473
vc.sales.asia@baslerweb.com

www.basler-vc.com

Appendix – C

Frame Grabber Data Sheet

Digital Camera Image Acquisition

NI 1424

- RS-422, LVDS, or TTL area and line-scan camera compatibility
- Full 8, 10, 12, 14, 16, 24, and 32-bit resolution (grayscale or color)
- 50 MHz pixel clock rate with up to 200 Mbytes/s acquisition
- 4 external TTL triggers (digital I/O lines)

NI 1422

- RS-422 or LVDS area and line-scan camera compatibility
- Full 8, 10, 12, 14, 16-bit resolution (grayscale or color)
- 40 MHz pixel clock rate with up to 80 Mbytes/s acquisition
- 4 external TTL triggers (digital I/O lines)

Models
NI 1424
- NI PCI-1424

NI 1422
- NI PCI-1422
- NI PXI-1422

Operating Systems
- Windows 2000/NT/XP/Me/9x

Recommended Software
- LabVIEW
- Measurement Studio
- NI Vision Development Module
  - IMAQ Vision
  - NI Vision Builder

Other Compatible Software
- C/C++

Driver Software (included)
- NI-IMAQ

Overview and Applications

The NI 1424 is a digital camera image acquisition board for PCI designed to acquire color and grayscale images and to control digital cameras. Designed for high-speed, large-image, high-resolution digital image capture, the board can capture into onboard memory up to 32 bits of data at a clock speed of 50 MHz, for a total acquisition rate of 200 Mbytes/s. Onboard memory (16 to 80 MB) gives you the flexibility to buffer images on the board for large image capture and sustained real-time throughput. Two versions of the board exist – one compatible with digital cameras that use RS-422 and another compatible with digital cameras that use LVDS.

The NI 1422 Series boards, in both PCI and PXI formats, are designed to acquire images from a wide range of digital cameras. These boards perform high-speed, large-image, high-resolution digital image capture, and can capture up to 16 bits of data at a clock speed of 40 MHz for a total acquisition rate of 80 Mbytes/s. Onboard memory (16 MB) gives you the flexibility to buffer images on the board for large-image capture and sustained real-time throughput. There are two versions of the board – one compatible with digital cameras that use RS-422 signals and another compatible with digital cameras that use LVDS. The NI 1422 Series is ideal for both industrial and scientific applications.

Digital Cameras

Digital cameras have several advantages over analog cameras. During transmission, analog video is more susceptible to noise than digital video. Digitizing at the CCD camera, rather than at the image acquisition board, increases the signal-to-noise ratio, resulting in better accuracy. In addition, standard digital cameras now come with 10 to 12-bit gray levels of resolution. The higher resolution is often necessary in medical, machine vision, astronomy, and scientific imaging applications.

INFO CODES
For more information, or to order products online visit *ni.com/info* and enter:
pci1424
pci1422
pxi1422
BUY ONLINE!

Easy Camera Configuration

Configure digital image acquisition from your digital camera with National Instruments Measurement & Automation Explorer, which is delivered with the NI-IMAQ driver software. This configuration utility is an interactive tool for setting up region of interest and camera control. Plus, you can use Measurement & Automation Explorer to control the serial interface for communicating with the camera. Both the PCI 1422 and the 1424 use a 100-pin SCSI-type connector that you can easily cable to your specific camera. In addition, Measurement & Automation Explorer includes specific camera setup files, so you can quickly configure your camera for acquisition. For a list of applicable cameras, visit *ni.com/camera*

Digital Image Acquisition

The NI 1424 can acquire four channels of 8-bit data, two channels of 10, 12, 14, or 16-bit data, or one channel of 24 or 32-bit data. The NI 1422 devices can acquire two channels of 8-bit data or one channel of 10, 12, 14, or 16-bit data.

These boards have a serial interface and four TTL or differential digital control lines for camera control. You can also use the adjustable onboard ROI window to minimize the amount of data transferred to PC memory.

Digital Camera Image Acquisition

Digital Transfer Rates

The base memory configuration for the NI 1424 is 16 MB for the board. You can use onboard memory to buffer large images before transferring them to PC memory, which increases the overall throughput. The 1422 can acquire data into onboard memory at a rate of up to 80 Mbytes/s and transfer data to PC memory at a rate of 100 Mbytes/s. The PCI-1424 can acquire data at 200 Mbytes/s into onboard memory.

Onboard Image Preparation

The 1424 has four 8-bit or two 16-bit LUTs for mapping the data from one value to another in real time. The 1422 has two 8-bit or one 16-bit LUT(s) for mapping the data from one value to another in real time.

In addition, setting a rectangular ROI window reduces the overall data across the PCI bus because the pixels outside the ROI are not transferred. You can reduce the amount of data transferred across the PCI bus by scaling or decimating the data in hardware. You can scale the image by keeping every second, fourth, or eighth pixel.

Digital I/O

Four general-purpose, bidirectional TTL digital I/O lines are available for triggering image acquisition.

NI 1424 Signaling

The RS-422/TTL version of the PCI-1424 can drive and receive either TTL or RS-422 level signals. The driver software can control these signal levels independently for data, control, and enable lines on the 100-pin connector. The LVDS (also known as EIA-644) version can drive and receive LVDS and TTL signals. Benefits of LVDS include less power consumption, longer cable lengths, less noise, and higher clock rates.

NI 1422 Signaling

One version of the NI 1422 can receive RS-422 data signals. It can also drive and receive RS-422 or TTL control and enable signals. NI-IMAQ can control these signal levels independently for data, control, and enable lines on the 100-pin connector. Another version of the NI 422 device is capable of receiving data from LVDS cameras. LVDS, also known as EIA-644, is a low-voltage differential signal protocol similar to RS-422 but with lower voltage levels.

Multiple-Channel Data Formatter

The multiple-channel data formatting circuitry is fully programmable, so you can simultaneously acquire from multiple channels to build up to the complete image.

Advanced Clock Generation

The PCI-1424 has two clock outputs for generating a frequency from 500 kHz up to 50 MHz for digital cameras that require an external clock. The NI 1422 has two clock outputs for generating a frequency from 500 kHz up to 40 MHz.

RS-232 Serial Interface

RS-232 serial lines are available on the 100-pin connector to control digital cameras that have a serial interface.

Camera Cables

National Instruments offers 2 m cables for digital cameras, including models by Balser, Dalsa, Hamamatsu, Roper Scientific, Pulnix, and others. Extension cables are also available (see the cable section). Contact National Instruments for additional cabling options or visit *ni.com/camera*

Advanced Triggering to Work with DAQ and Motion

IMAQ products are designed to work with National Instruments DAQ and motion products. The RTSI bus on the PCI-1422 and PCI-1424 routes timing and triggering signals between boards. The PXI-1422 uses the PXI trigger bus similarly. The buses can synchronize video acquisition between several IMAQ devices or between IMAQ, DAQ, and motion control devices.

I/O Connector Signals

The 100-pin SCSI connector connects to all digital video data inputs, digital enable inputs, camera control outputs, RS-232 serial interface, and the external trigger signals.

Digital Trigger Accessory

The IMAQ D2504-1, which has four BNC connectors for digital trigger lines, is for use with National Instruments digital camera cables.

Digital Camera Image Acquisition

Ordering Information

PCI-1424 RS-422/TTL 777662-01
PCI-1424 LVDS/TTL 777662-02
Includes NI 1424 board and NI-IMAQ software for Windows 2000/NT/XP/Me/9x

Cables

See page 609 for 2 m cables to connect the NI 1424 to different cameras.

Digital Trigger Accessory

IMAQ D2504-01 (1 m).....185298-01

Accessories

100-pin to 100-pin extension cable
- 2 m 184743-02
- 4 m 184743-04

Memory upgrades
- 32 MB 920130-32
- 64 MB 920130-64

Ordering Information

NI 1422 for RS-422 Digital Cameras
- PCI-1422 RS-422 777959-01
- PXI-1422 RS-422 777933-01

NI 1422 for LVDS Digital Cameras
- PCI-1422 LVDS 777959-02
- PXI-1422 LVDS.....777933-02

Includes NI 1422 device and NI-IMAQ driver software for Windows 2000/NT/XP/Me/9x.

Digital Trigger Accessory

IMAQ D2504-1 (1 m).....185298-01

Cables

Refer to Camera Advisor at *ni.com/camera* for the proper camera configuration.

Extension Cables

100-pin to 100-pin extension cable
- 2 m 184743-02
- 4 m 184743-04

Tech Tip

How do I use IMAQ hardware onboard memory?

Onboard memory buffers continue to acquire resources even if there is a delay associated with the PCI bus, operating system, or image processing. Use onboard memory for high-speed burst acquisitions up to 200 Mbytes/s. Learn more about NI-IMAQ and the acquisition modes (grab, ring, and sequence) designed for display and processing of high-speed images by visiting *zone.ni.com*

For more information, visit *ni.com/info* and enter: ex9kr5.

Digital Camera Image Acquisition

Specifications
NI 1424

External Connections

Trigger sense	TTL
Trigger level	Programmable (rising or falling)
Pixel clock sense	LVDS or RS-422 selectable (TTL or differential)
Pixel clock level	Programmable (rising or falling)
Enable sense	LVDS or RS-422 selectable (TTL or differential)
Enable level	Programmable (rising or falling)
Master clock drive	LVDS or RS-422 selectable (TTL or differential)
Master clock level	Rising edge
Control signal drive	LVDS or RS-422 selectable (TTL or differential)
Control signal level	Programmable (rising or falling)
Video data sense	LVDS or RS-422 selectable (TTL or differential)

Clocks

Master clock frequency range	500 kHz to 50 MHz
Pixel clock frequency range	500 kHz to 50 MHz

PCI Master Performance

Ideal	133 Mbytes/s
Sustained	100 Mbytes/s

Power Requirements

+5 VDC (±5%)	2.135 A
+12 VDC (±5%)	25 mA
-12 VDC (±5%)	20 mA

Physical

Dimensions

PCI	11.4 by 33.6 cm (4.5 by 13.2 in.)
PXI	10 by 16 cm (3.9 by 6.3 in.)

Environment

Operating temperature	0 to 50 °C
Storage temperature	-25 to 70 °C
Relative humidity	10 to 40% noncondensing

Specifications
NI 1422

External Connections

Trigger sense	TTL
Trigger level	Programmable (rising or falling)
Pixel clock sense	LVDS or RS-422 selectable (TTL or differential)
Pixel clock level	Programmable (rising or falling)
Enable sense	LVDS or RS-422 selectable (TTL or differential)
Enable level	Programmable (rising or falling)
Master clock drive	LVDS or RS-422 selectable (TTL or differential)
Master clock level	Rising edge
Control signal drive	LVDS or RS-422 selectable (TTL or differential)
Control signal level	Programmable (rising or falling)
Video data sense	LVDS or RS-422

Clocks

Master clock frequency range	500 kHz to 40 MHz
Pixel clock frequency range	500 kHz to 40 MHz

PCI Master Performance

Ideal	133 Mbytes/s
Sustained	100 Mbytes/s

Power Requirements

+5 VDC (±5%)	2.135 A
+12 VDC (±5%)	25 mA
-12 VDC (±5%)	20 mA

Physical

Dimensions

PCI	11.4 by 33.6 cm (4.5 by 13.2 in.)
PXI	10 by 16 cm (3.9 by 6.3 in.)

Environment

Operating temperature	0 to 50 °C
Storage temperature	-25 to 70 °C
Relative humidity	10 to 40% noncondensing

VITA

Sai Venu Gopal Lolla

Candidate for the Degree of

Master of Science

Thesis: IDENTIFICATION AND INSPECTION OF SILVERWARE USING MACHINE VISION.

Major Field: Mechanical Engineering.

Biographical:

Personal Data: Born in Rajahmundry, Andhra Pradesh, India, on 29th April 1980, the eldest son of Nageswara Rao Lolla and Umabala Bandhakavi.

Education: Graduated from Timpany Sr.School, Visakhapatnam, India in May 1997; received Bachelor of Technology in Mechanical Engineering from Jawaharlal Nehru Technological University in May 2002. Completed the requirements for the Master of Science degree with a major in Mechanical Engineering at Oklahoma State University in May 2005.

Experience: Graduate Teaching Assistant, Department of Mechanical and Aerospace Engineering, Oklahoma State University from August 2002 to December 2002. Graduate Research Assistant, Department of Mechanical and Aerospace Engineering, Oklahoma State University from January 2002 to May 2005.