Abstract

KULKARNI, PARIKSHIT S. CAD based computer vision for pose determination of wooden parts. (Under the direction of Dr. Denis Cormier)

Furniture manufacturers today have to deal with the conflicting objectives of increased product variety with reduced lead-time and inventory levels. The prevalent sentiment in the industry is a build to order production strategy, which tends to increase the production costs. Furniture makers are hence looking for technological solutions to counter these issues. Fixture-less machining on CNC routers is an alternative gaining prominence among manufacturers to reduce setup times and increase productivity. However, in the absence of locating pins for mounting the parts, the orientation and part placement becomes critical and small errors can increase rejection rates. Updating the CNC part programs after determining the exact orientation and location of the part blanks is a likely solution. This thesis represents an effort to develop a computer vision based system to enable this. Some of the popular techniques in computer vision for object recognition were hence implemented. These however did not prove worthwhile as the efficacy of the algorithms was undermined by the texture from wood grains. A template matching procedure was identified as being better suited for the purpose because of its inherent tolerance for noise. The generation of the template to account for hidden lines and actual projected lengths, both functions of the viewing direction and camera location, was however the biggest challenge. The fact that most furniture manufacturers have a very large product range that needs machining on the router compounds the problem. A novel CAD based template generation algorithm was hence proposed and implemented. The developed algorithms enabled orientation determination with a negligible error, and the average error in locating the part was less than 0.2 percent. The approach was therefore found to have a potential in reducing the lead times while specifying tighter tolerances to reduce raw-material consumption. Finally, some future work to completely automate the process and to extend the applicability of the approach has been proposed.

CAD based Computer Vision for Pose Determination of Wooden Parts

By

PARIKSHIT S KULKARNI

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Industrial Engineering

Raleigh, NC

2001

APPROVED BY

Dr. C. G. Healey	**Dr. Y. S. Lee**

Dr. Denis R. Cormier

Chair of the Advisory Committee

Personal Biography

Parikshit Kulkarni was born on 1st May 1977 in Amravati, India. He did his schooling at Model English School, Mumbai followed by pre-university at K V Pendharkar College, Mumbai. After schooling, he joined Dr Babasaheb Ambedkar Technological University, Mumbai, in 1994 for further education. After graduating with a degree in Bachelor of Technology in Mechanical Engineering, Parikshit worked at the Industrial Design Center, Indian Institute of Technology, Mumbai. He then moved to North Carolina State University, Raleigh, USA, for graduate studies in August 1999. After completion of his Master's degree in Industrial Engineering, he plans to work with Numerical Technologies, Inc. in Raleigh.

Acknowledgements

The author would like to express deepest gratitude to his advisor Dr. Denis Cormier for his continuous encouragement, support and invaluable guidance throughout the course of this research project. He would like to thank his committee members, Dr. C. G. Healey and Dr. Y. S. Lee for their enthusiastic involvement and being on this committee.

Thanks to Mr. Tushar Mahale, and Jason Low for a lending helping hand during the course of this research project.

Finally, the author expresses thanks to his family members for the encouragement and support they have given him in all his ventures.

TABLE OF CONTENTS

List of Figures……vi

List of Tables……ix

1. INTRODUCTION 1
   - 1.1 CHARACTERIZATION OF THE FURNITURE PRODUCTION ENVIRONMENT 1
   - 1.2 FIXTURELESS MACHINING 3
2. PARADIGMS IN COMPUTER VISION 7
   - 2.1 ARCHITECTURE OF A VISION SYSTEM 7
   - 2.2 DIGITAL IMAGES 9
   - 2.3 OPERATIONS ON IMAGES 10
     - *2.3.1 Thresholding 10*
     - *2.3.2 Edge detection 13*
     - *2.3.3 Mean filtering 16*
3. LITERATURE REVIEW 19
   - 3.1 OBJECT RECOGNITION 20
     - *3.1.1 Edge Detection 22*
     - *3.1.2 Region growing 22*
     - *3.1.3 Template Matching 23*
   - 3.2 CAD BASED COMPUTER VISION 24
   - 3.3 STATEMENT OF NEED 28
   - 3.4 PROPOSED SOLUTION FOR INCREASING PRODUCTIVITY 29
4. PRELIMINARY INVESTIGATION 32
   - 4.1 THE HOUGH TRANSFORM 33
   - 4.2 REGION GROWING 39
   - 4.3 CONTOUR FOLLOWING 45
   - 4.4 TEMPLATE MATCHING 52
5. METHODOLOGY 54
   - 5.1 OVERVIEW 54
     - *5.1.1 CAD Models 57*
   - 5.2 TEMPLATE GENERATION 58
     - *5.2.1 Generating the list of unique vertices 58*
     - *5.2.2 Generating the list of unique edges 59*
     - *5.2.3 Generating the list of surfaces 62*
     - *5.2.4 Identification of edges on surface boundaries 64*
     - *5.2.5 Rendering the solid model for template generation 66*
     - *5.2.6 Scan conversion of the rendering 71*
     - *5.2.7 The Template image 73*

5.3 TEMPLATE MATCHING 74
5.4 UPDATING PARAMETRIC PART PROGRAMS 81
6. IMPLEMENTATION AND RESULTS 84
6.1 IMPLEMENTATION 84
6.2 RESULTS 88
6.3 LIMITATIONS 102
7. CONCLUSIONS AND RECOMMENDATIONS 105
7.1 CONCLUSIONS 105
7.2 FUTURE WORK 106
8. REFERENCES 109
9. APPENDIX 111
9.1 STL FILE FORMAT 112

List of Figures

Figure 1.1 - Layout of a flexible manufacturing cell integrated with a vision system..... 3
Figure 1.2 - A CNC router with a vacuum fixture 4
Figure 1.3 - A sample part that is machined in multiple setups. (a) Part orientation after first setup, (b) Finished part..... 5
Figure 2.1 - A typical computer vision system 8
Figure 2.2 - (a) A sample digital image with 256 gray levels; (b) Gray values within a 6x6 grid inside (a)..... 10
Figure 2.3 - A sample input image used for thresholding..... 11
Figure 2.4 - (a) Binary image with a threshold value of 155, and (b) Binary image with a threshold value of 210 12
Figure 2.5 - (a) Gray Scale image of a part blank, and (b) Resultant image upon performing edge detection on (a)..... 15
Figure 2.6 - (a) Grayscale values from a sample digital image, and (b) Resultant image after application of the Sobel kernel..... 16
Figure 2.7 - (a) Image of a sample part blank, and (b) Resultant image after the application of a mean filter..... 18
Figure 3.1 - Hypothesis generation (a) Edge based (b) Surface based 21
Figure 4.1 A sample image captured with the current vision setup 32
Figure 4.2 Parametric equation of a line 33
Figure 4.3 (a) Image of a part blank; (b) The gradient image of (a); and (c) Hough space for the image in (b)..... 35
Figure 4.4 (a) A sample part blank, and (b) The de-Houghed image with pixels clustered along lines..... 37
Figure 4.5 (a) A part blank with pronounced wood grains; and (b) The de-Houghed image. 38
Figure 4.6 A 4-connected region..... 40
Figure 4.7 An 8-connected region..... 41
Figure 4.8 (a) Part blank of a furniture piece; (b) The resultant image upon thresholding (a); and (c) Blobs in the image 44

Figure 4.9 (a) Part blank for a drawer front; and (b) Delineated blobs for the image in (a) 45
Figure 4.10 (a) Freeman direction codes for a zone of radius 1; and (b) Freeman codes for a zone of radius 2 46
Figure 4.11 (a) A sample part blank; (b) Image generated upon application of contour following on (a); and (c) Lines identified after clustering pixels in (b) 51
Figure 4.12 Illustration of the template matching procedure 52
Figure 5.1 (a) Resultant of edge detection on a sample image, and (b) A sample template for pose determination of image in (a) 55
Figure 5.2 Methodology for template generation and pose determination 56
Figure 5.3 Tessellations in an STL file 58
Figure 5.4 Interpolation of z values along polygon edges and scan lines. 68
Figure 5.5 Rendering of a part for pose determination 71
Figure 5.6 Image generated by scan converting the rendering in Figure 5.5 72
Figure 5.7 Template generated after cropping image in Figure 5.6 74
Figure 5.8 (a) Part blank for a drawer-front, (b) Input image for template matching and (c) Selected template, which gives the best match 77
Figure 5.9 Image illustrating the match between the template and input image 78
Figure 5.10 (a) The region interest for a part blank; and (b) The bounding box for a part 80
Figure 6.1 Dialogue box to select the bitmap image of a part 85
Figure 6.2 Dialogue box to select the solid model of the part 85
Figure 6.3 Dialogue box to input the bounding box and expected orientation of the part 86
Figure 6.4 Menu selection to search for the object's pose 87
Figure 6.5 Solid model of (a) Inner side of the CD case; and (b) The outer side 88
Figure 6.6 Image of the part blank before machining 'NC State' 89
Figure 6.7 Inputs to determine the pose of image in Figure 6.6 89
Figure 6.8 (a) Image after performing edge extraction on image in Figure 6.6; (b) The selected template; and (c) Image showing the match between the template and input 91
Figure 6.9 Points of interest for machining the part 92

Figure 6.10 (a) Part image with incorrect orientation; and (b) Input parameters for pose determination 93
Figure 6.11 (a) Image after performing edge extraction; (b) The selected template image; and (c) Image illustrating the match between the images. 95
Figure 6.12 (a) Inner side of the drawer front; and (b) Outer side of the part 96
Figure 6.13 (a) The input image of the drawer front; and (b) Input parameters for the algorithm 97
Figure 6.14 (a) Resultant image upon edge extraction on Figure 6.13 (a); (b) The selected template image; and (c) Image illustrating the match between the input and template 99
Figure 6.15 (a) The input image; and (b) The input parameters 100
Figure 6.16 (a) Input image for template matching; (b) The selected template; and (c) Image illustrating the match between the input and template 101
Figure 6.17 (a) Image with presence of noise; and (b) Resultant image after edge extraction and cropping. 103
Figure 7.1 Determination of the maximum possible deviations 107

List of Tables

Table 6.1 Comparison between manually measured and algorithmically computed values for sample points 92
Table 9.1 The STL file format 112

1. Introduction

1.1 Characterization of the Furniture Production Environment

For many years, furniture manufacturing was an industry that was relatively untouched by technological advances of the industrial revolution. Even until the early 20th century, furniture was built by hand, one piece at a time. From the 1950's through the present day, the industry has witnessed a steady increase in the adoption of specialized mass production equipment. While this equipment is capable of processing lumber very quickly, the changeover time between jobs is extremely long. This type of equipment has forced companies to produce parts in large batches. An ensuing result of this strategy is that large finished goods inventories must be carried. Another result of large lot sizes is that a given item may only be scheduled for production a few times per year. Consequently, lengthy delivery lead times have plagued the industry despite the high finished goods inventory. The problem is greatly compounded by the fact that furniture is very much a style oriented industry where fashions go in and out of style very quickly. Companies that stock large amounts of finished goods risk the prospect of having to write off massive losses if styles go out of vogue. Furniture makers are therefore striving to pursue new technologies that allow them to efficiently produce items more frequently in smaller lot sizes.

With today's burgeoning global economy, matters have become even more challenging for furniture manufacturers in the U.S., as they are experiencing an alarming erosion of domestic market share at the hands of inexpensive imports. As an incentive, many companies have been offering customers more choices and trying to improve customer responsiveness by dramatically reducing order delivery lead times. Increased product variety with reduced lead-time and inventory levels however, are conflicting objectives. To satisfy these goals, the prevailing wisdom in the furniture industry is that finished goods inventories should be greatly reduced in favor of build-to-order production strategies. According to this strategy, a company does not build a piece of furniture unless an order has already been placed for it. The natural consequence of a build-to-order production strategy is that lot sizes are quite small compared with what they used to

be, and consequently the cost of production is higher. To combat the challenges presented by small lot manufacturing, conceptually, the easiest alternative for a factory is to run production in the 2nd and/or 3rd shifts. However, the relatively low wages offered in furniture factories make it difficult for manufacturers to attract enough people who are willing to work in the 2nd and 3rd shifts.

Computer integrated manufacturing (CIM), aimed at automating the various steps in manufacturing, offers a potential solution. CIM necessitates the integration of various subsystems of automated manufacturing such as CAD, CAM, computer vision and robotics. Vision systems in these setups play a pivotal role in tasks such as inspection, recognition and guidance for robots. Moreover, the real-time feedback obtainable with a vision system enables handling of in-situ uncertainties for tasks such as robot path planning, and provides for corrections required to annul motor backlash. Machine vision systems are thus an important cog in the wheel for CIM. Manufacturing setups incorporating CIM systems are usually implemented as Flexible Manufacturing Cells (FMC's) such as the one shown in Figure 1.1. This is the FMC currently in use at North Carolina State University in the Department of Industrial Engineering.

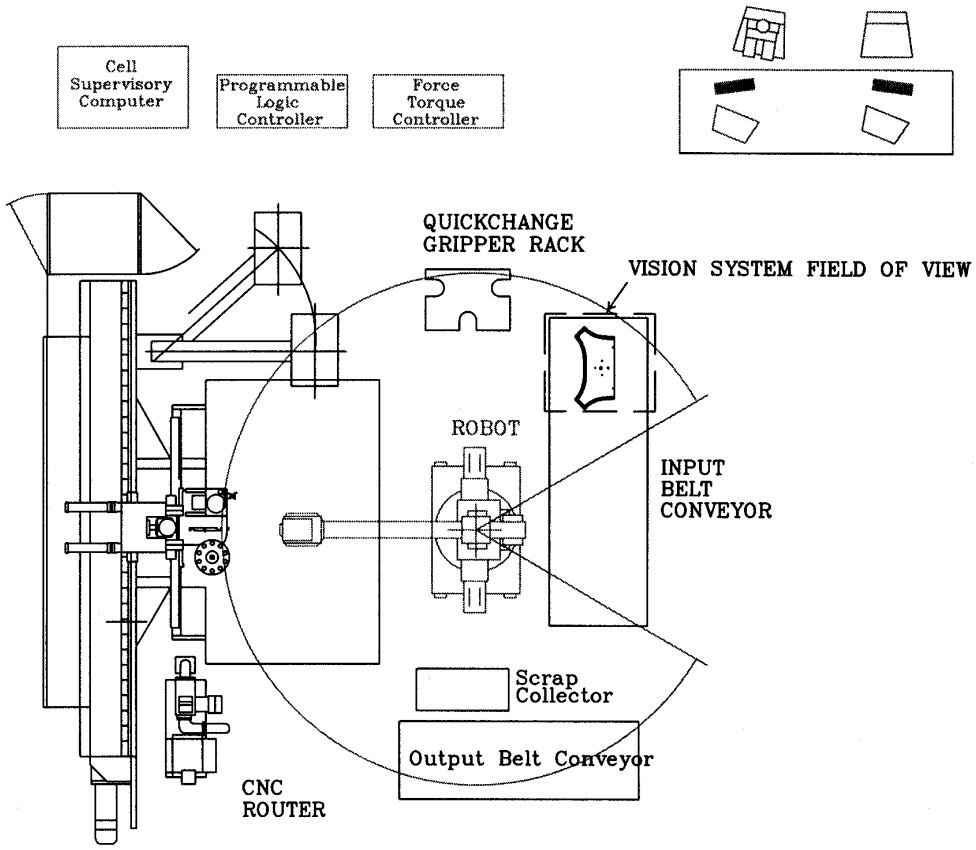


Figure 1.1 - Layout of a flexible manufacturing cell integrated with a vision system

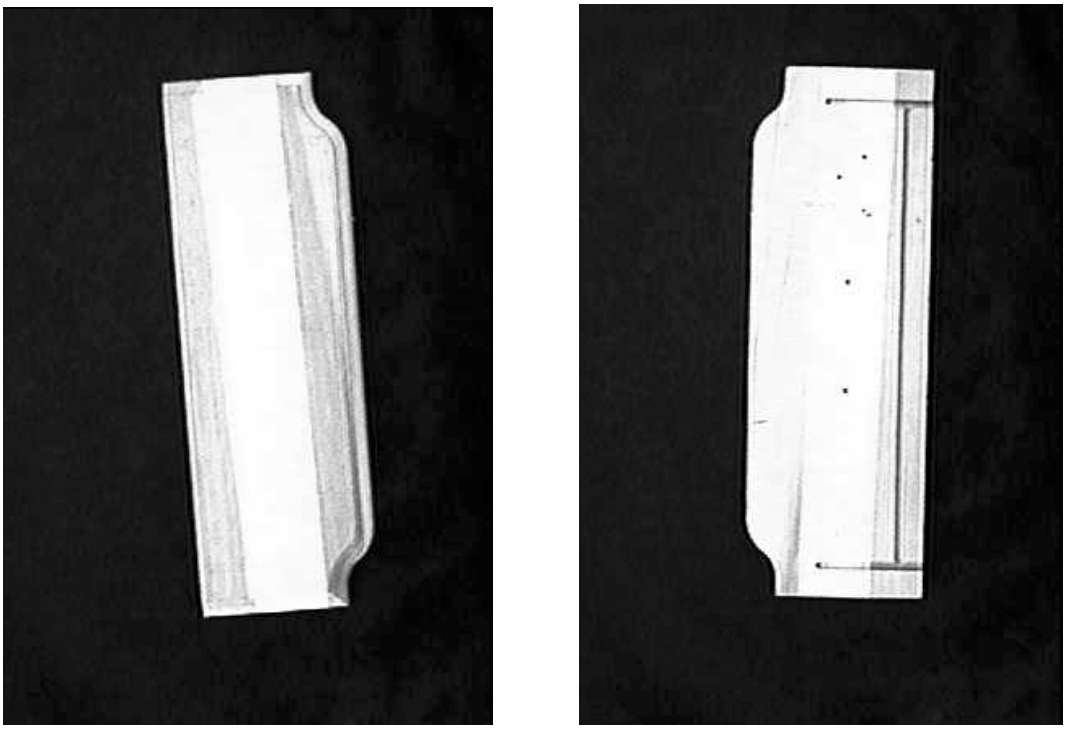
1.2 Fixtureless machining

The CNC router has become one of the most widely used pieces of equipment in furniture FMC's. In furniture FMC's, CNC routers are usually used for machining flat part blanks that are laid out on the bed of the router and held in place with vacuum suction. The part blanks are machined to the desired shape according to pre-programmed instructions called part programs. A typical setup for the CNC router is shown in Figure 1.2.

Figure 1.2 - A CNC router with a vacuum fixture

Part placement on the CNC router bed is usually done manually, as robotic manipulation is still very capital intensive. At the present time, there are several different types of vacuum fixture. The type shown in Figure 1.2 uses “pods” in which the machine operator manually removes plugs beneath those areas that will be covered by a workpiece. Furthermore, pegs are inserted as needed to serve as hard stops to accurately locate part blanks prior to machining. While these types of vacuum fixtures are quite convenient, they can be quite labor intensive. More recently, so-called “universal fixtures” have begun to appear that do not require manual removal of plugs or placement of stop pins. The simplest form of universal fixture is a porous sheet of medium density fiberboard (MDF) through which a heavy vacuum is drawn. Parts are laid flat on the porous MDF, and the slow vacuum is sufficient to hold the parts steady during machining. Due to the

absence of locating pins for mounting the parts, the orientation and part placement become critical, and small errors can increase rejection rates. Updating the CNC part programs after determining the exact orientation and location of the part blanks seems a likely solution. Moreover, many of the components used in making furniture need to be machined in multiple setups. An example part, a drawer front, is shown in Figure 1.3 (a) and Figure 1.3 (b). In such cases, the FMC's productivity can be increased tremendously if the drawer front can be machined on a CNC router without the need for specialized locating devices and time consuming part setups. Moreover, the orientation of the drawer front during the second setup is dependent on the geometry of the features that have been machined in the first setup. In such circumstances, updating the CNC programs to take into account the precise orientation of the part when it has been placed on the router bed can greatly reduce rejections.



(a) (b)

Figure 1.3 - A sample part that is machined in multiple setups. (a) Part orientation after first setup, (b) Finished part

It is the determination of the exact part location and orientation on the bed of the CNC router that is the focus of this thesis. The proposed method employs the use of computer vision technology. In order to provide the necessary background, the fundamentals of a standard computer vision system are covered in Chapter 2. A review of the current literature on the subject of object recognition with computer vision techniques is presented in Chapter 3. The need for a CAD based vision system for the current application is also discussed in Chapter 3. Chapter 4 discusses the implementation of some popular algorithms in object recognition. It also details the reasons for their unsatisfactory performance for wooden parts and identifies template matching as the preferred methodology. Chapter 5 details the implementation of the template matching approach and the algorithm for template generation for different orientations of the part. Testing, results and analysis are presented in Chapter 6. Lastly, conclusions and recommendations for future improvements are provided in Chapter 7.

2. Paradigms In Computer Vision

Computer vision is the process whereby a machine, usually a digital computer, automatically processes an image and provides information about the contents of the image. Quite often, the content is a machined part, and the objective is not only to locate the part, but to inspect it as well. Broadly speaking, computer vision includes two components - measurement of features, and pattern classification based on those features. For a typical industrial application, feature measurement encompasses the determination of part length, width, area of a surface in the image, etc. Once the features are measured, their numerical values are passed on to a process that implements relevant decision rules. These decision rules are usually fall under the heading of pattern classification. That is, given a set of measurements and knowledge about the possible classes to which the object might belong, the pattern classification rules determine which features are present in the image. The image on which feature measurement and pattern classification procedures are applied thus forms the starting point for a computer vision system. Acquisition and processing of the image for interpretation by the computer is hence a vitally important step. This chapter briefly outlines various subsystems of a computer vision setup along with an intuitive description of the image itself.

2.1 Architecture of a vision system

Computer vision has been used extensively for diverse tasks such as part recognition for material handling, inspection for quality control, and guidance tools for robots. In the context of the manufacturing environment, vision systems typically encompass five different components. These are:

- A method of automatically presenting the part to the system, which subsumes a mechanism for part acquisition, positioning, and registration.
- A sensing system which provides a representation that is understandable by computer systems.
- A process to extract pertinent features.

- A set of criteria upon which decisions are based.
- A system to release the part into separate bins, depending upon the decisions taken following image analysis.

A schematic representation of a typical computer vision system is shown in Figure 2.1

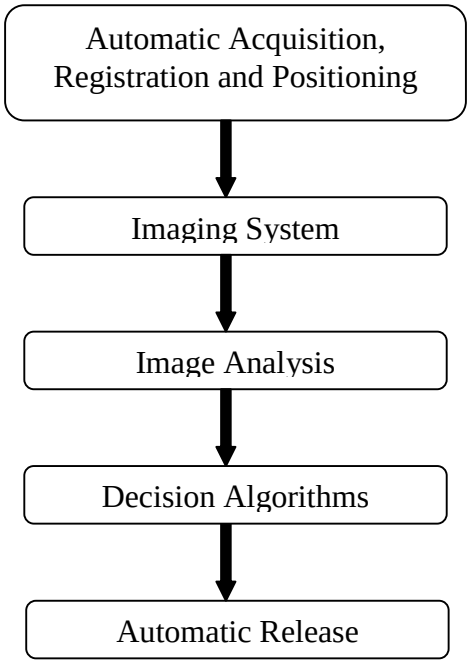


Figure 2.1 - A typical computer vision system

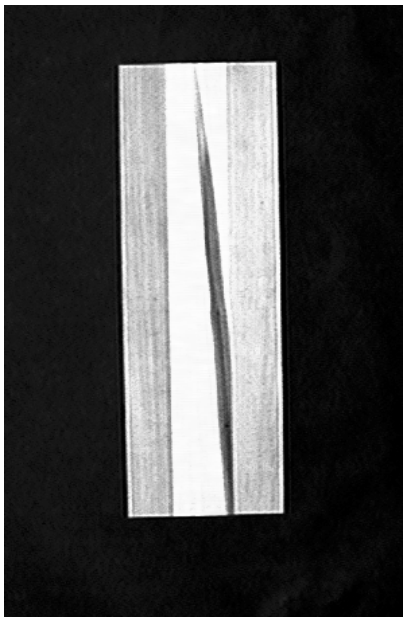
(source: David Vernon, 1991)

A computer vision system employed within a manufacturing setup usually has components to automate the process of object recognition. Mechanisms such as robotic manipulators and specialized fixtures are typically used for the purpose of presenting the object to the vision system. In Figure 2.1, this sub-system is shown as the block for automatic acquisition, registration and positioning. Thus, the object is picked up from, perhaps, a conveyer belt and is positioned suitably in the field of view of the vision system. The imaging system block has sub-systems for lighting, image acquisition, and pre-processing. Lighting is an extremely important aspect of any vision system, as it affects properties such as color perception, shadows, etc. The image acquisition system generates an analogue representation of the image scene. The task of the acquisition and

pre-processing systems is to convert this analog signal into a digital image using hardware components such as frame grabbers, and to manipulate the resultant image to facilitate subsequent extraction of information. This manipulation includes removal of noise and aberrations in the captured image. The analysis stage is concerned with the extraction of explicit information regarding the contents of the image. This includes information such as object position, size, orientation etc. This facilitates transformations on the image which render it useful for the decision making process. The algorithms used in the decision making process are used to interpret the data from the previous stage. Decisions for the acceptance or rejection of a part or the initiation of some other action is done in this stage. The last stage is responsible for releasing the part from the fixtures or positioning devices of the vision system. The part may be placed into lots categorized according to the decision made earlier in the system. These systems thus form the building blocks of a typical computer vision system.

2.2 Digital images

The data captured by the camera in a vision system is usually a continuos analogue function of some property in the visual scene. This is typically the reflectance function of the scene, that is, the intensity of light reflected at each visible point in the scene. A continuos analogue representation cannot be interpreted conveniently by a computer, hence a sampled and quantized representation must be used. This is achieved with a frame grabber that converts the analogue signal into a digital image by sampling and quantization. The frame-grabber samples the video signal in some predetermined fashion, usually in an equally spaced square grid, and quantizes the reflectance function at those points into integer values called gray levels. Each integer gray-level value is referred to as a pixel and is the smallest discreet accessible address of a digital image. The number of gray levels in the scale is the quantization or gray-scale resolution of the system. The setup used in the current system quantizes the brightness into 256 discreet levels. Brightness levels of zero and 255 correspond to black and white respectively. Frame grabbers that quantize the image into shades of red, green and blue are also available. A sample digital image with discreet values for a small portion of the image are shown in Figure 2.2.



223	223	204	204	200	222
173	201	212	228	210	215
182	188	214	223	213	201
191	205	198	196	213	215
198	211	203	203	210	215
174	205	205	215	202	210

(a) (b)

Figure 2.2 - (a) A sample digital image with 256 gray levels; (b) Gray values within a 6x6 grid inside (a)

2.3 Operations on images

Various operations are performed on digital images to extract the necessary information. Most vision systems have specific image operations, and the operations performed on the captured images need not be universally applicable. The operations usually are pertinent only to a particular setup, and the illumination scheme and background for the part mainly affect the efficacy of the operation. Thresholding is one commonly used operation that converts a gray level image into a binary (black and white) image. In a binary image, black is represented as 0 and white is 1. Thus there are only two levels of gray in a binary image.

2.3.1 Thresholding

Thresholding is used mainly to process the image of a scene so that the resultant image exhibits extremely high contrast. The problem is thus to convert a gray scale image,

typically with 256 or 512 levels of gray, into an image with just two levels of gray. This is done as follows. For each pixel in the input image, if the gray level for that pixel is greater than or equal to a fixed threshold value, the corresponding pixel in the new thresholded image is set to white. Otherwise, the corresponding pixel in the thresholded image is set to black. The thresholded image therefore has just two levels of gray (i.e. it is a binary image). The threshold value is selected as some number between 1 and 254. The resultant image thus depends upon the selection of the threshold that is chosen. The following figures show the effect of choosing different values for the threshold.

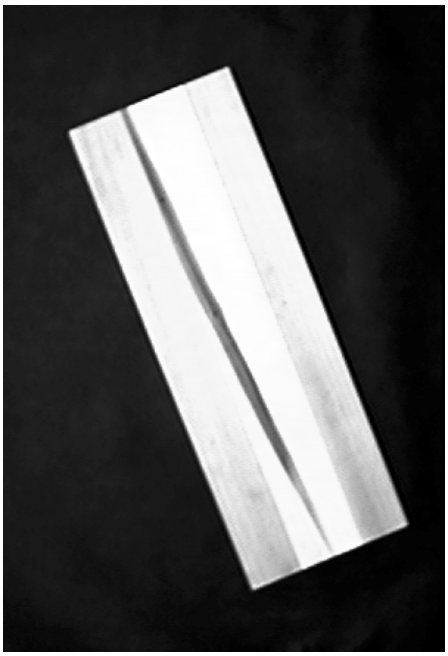


Figure 2.3 - A sample input image used for thresholding

(a) (b)

Figure 2.4 - (a) Binary image with a threshold value of 155, and (b) Binary image with a threshold value of 210

The appearance of the binary image thus depends to a large extent on the value of the threshold that is chosen. The value of the threshold for a particular application depends on the illumination system, the background, and optical properties of the part being imaged. Changes in any of these three factors will affect the output image considerably, so threshold values are typically selected on an application by application basis. For example, experimentation with the test setup used for this research indicated that a threshold value of around 190 was suitable.

It is worth noting that in many cases, the actual grayscale values in an image do not span the full 256 levels of gray. In other words, the lowest observed grayscale will be greater than 0, and the highest observed grayscale will be less than 255. In these cases, it is helpful to enhance the contrast of the image prior to thresholding. This is done by adjusting the grayscale values such that they extend from 0 to 255. If b_{min} and b_{max} denote the minimum and maximum brightness (grayscale) values found in an image, and $b_{i,j}$ is the brightness of the pixel in row *i*, column *j*, then the enhanced contrast of pixel *i, j* is found as follows:

$$b'_{i,j} = (b_{i,j} - b_{\min}) \times \frac{255}{(b_{\max} - b_{\min})} \qquad \textbf{Equation 2.1}$$

Note that if $b_{i,j} = b_{max}$, the above equation yields a new brightness of 255. If $b_{i,j} = b_{min}$, the above equation yields a new brightness of 0. All brightness values between b_{min} and b_{max} are linearly scaled into the [0, 255] range.

2.3.2 Edge detection

Edge detection is a computation that is performed in most object recognition algorithms. It is in essence a segmentation procedure, where edges are viewed as local discontinuities in intensity. The gradient function of the image, a first-order derivative of the intensity, has high values for pixels that lie along the edges in the image. It is then possible to extract the edges by simply choosing a suitable threshold for the value of the gradient. Thus, the most important task is estimation of the first-order derivative. This is accomplished with the help of edge detection operators or kernels. An edge detection kernel essentially calculates the differential in the intensities of the image to determine the pixels that constitute the edge. The Sobel operator [Vernon, 1991] estimates the partial derivatives in the X and Y directions. This is used to compute both the gradient direction and the magnitude. The kernels are square matrices that are placed at each pixel location in the image. The brightness value at the respective pixel is multiplied with elements of the matrix. The X and Y Sobel kernels are given as:

$$Hx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \textit{and} \quad Hy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

where *Hx* and *Hy* are the partial derivatives along the X and Y direction of the intensity of the image.

In mathematical form, suppose that the image function is expressed as *f(x,y)*, where the function *f* returns the gray level value at location *(x,y)*. The Sobel operator estimates the partial derivative in the X-direction over a 3 x 3 region centered at *(x,y)* as:

$$S_x = \{f(x+1, y-1) + 2f(x+1, y) + f(x+1, y+1)\} - \{f(x-1, y-1) + 2f(x-1, y) + f(x-1, y+1)\}$$ **Equation 2.2**

Likewise, derivative in the Y-direction is:

$$S_y = \{f(x-1, y-1) + 2f(x, y-1) + f(x+1, y-1)\} - \{f(x-1, y+1) + 2f(x, y+1) + f(x+1, y+1)\}$$ **Equation 2.3**

This essentially takes the difference of a weighted average of the image intensity on either side of *f(x,y)*. The gradient can then be estimated by calculating the root mean square (RMS) value as:

$$g(x, y) \approx S = \sqrt{(S_x^2 + S_y^2)}$$ **Equation 2.4**

The direction, θ, of the gradient at (*x*,*y*) is given as:

$$\theta(x, y) = \tan^{-1}(\frac{S_y}{S_x})$$ **Equation 2.5**

The image resulting from the computation of the gradient magnitude is thresholded to obtain edges in the image. Again, the threshold value that is chosen for the purpose of edge detection depends on the setup at hand. For the work described in this thesis, a value for the threshold is not hard-coded, but rather is computed during execution. The pseudo code for the procedure to calculate the edge finding threshold value is given below:

gmax = max_intensity(gradient_image)

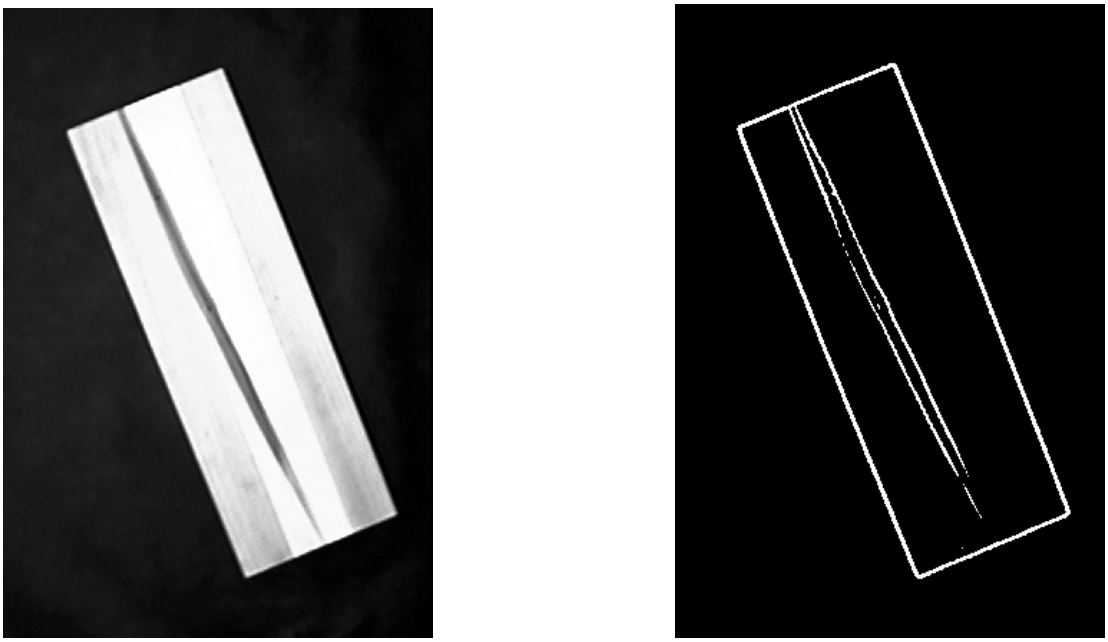
gmin = min_intensity(gradient_image)

a = (gmax-gmin)/7

threshold = a + (gmax – a)/8;

Here, the gradient image is the image obtained after computation of the RMS values of the partial derivatives along the X and Y directions. This procedure was found to work suitably for most images captured in the experimental setup. It also accounts for fluctuations in the quality of the captured image.

The above described edge finding routines are demonstrated in an example below. Figure 2.5(a) shows a captured image of a block of wood. Figure 2.6(a) shows grayscale values from a small selected region of the grayscale image. The intensity gradient value for each grayscale pixel in the selected region is computed using Equation 2-4, and the result is shown in Figure 2.6(b). Using the gradient threshold calculation procedure described above, a gradient threshold for this image was computed to be 191.Upon application of this threshold to the gradient image, the resulting image with edges is shown in Figure 2.5.



(a) (b)

Figure 2.5 - (a) Gray Scale image of a part blank, and (b) Resultant image upon performing edge detection on (a).

223	204	204	200	222	227
201	212	228	210	215	209
188	214	223	213	201	186
205	198	196	213	215	223
211	203	203	210	215	221
205	205	215	202	210	222

70.11	61.66	38.42	70.03	103.04	50.70
104.31	75.80	46.04	48.10	29.15	6.00
21.02	64.90	30.446	60.13	86.00	72.18
36.25	42.80	38.83	56.46	4.24	68.60
32.28	35.01	43.86	79.81	35.44	51.48
59.01	51.92	52.15	76.90	35.38	108.23

Figure 2.6 - (a) Grayscale values from a sample digital image, and (b) Resultant image after application of the Sobel kernel

2.3.3 Mean filtering

Mean filtering is a simple, intuitive, and easy to implement method of smoothing images and thus removing noise. The procedure does this by reducing the amount of intensity variation between one pixel and the next. Mean filtering is accomplished by applying a low pass filter that removes signals with low frequencies in the input image. The low pass filter uses a square matrix that is applied successively to each pixel in the image. The size of the kernel affects the extent of smoothing. Large kernels cause greater blurring or smoothing of input images. The kernel used for 3 x 3 mean filtering is:

$$M = 1/9 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

This simple kernel can be very effective in reducing the noise in an image. Consider a 3 x 3 region in a digital image as shown below:

$$A = \begin{bmatrix} 100 & 100 & 100 \\ 100 & 50 & 100 \\ 100 & 100 & 100 \end{bmatrix}$$

In the region under consideration, 8 out of the 9 pixels have an intensity of 100. The pixel at the center has a value of 50, which obviously is noise. Application of the mean filter will help in reducing the noise by contrast stretching. Most of these kernels are “circulant” matrices. This means that each column is simply a rotation of the adjacent

column. Thus at the last column, data from the first two columns is used to apply the kernel. Thus, application of the kernel given in M on A yields:

$$R = \begin{bmatrix} 94.44 & 94.44 & 94.44 \\ 94.44 & 94.44 & 94.44 \\ 94.44 & 94.44 & 94.44 \end{bmatrix}$$

This is the resultant image after application of the mean filter on the image in A.

Mean filtering was used in this implementation mainly to reduce the effect of grain boundaries in wooden part blanks. A sample output after application of a mean filter is shown in Figure 2.7.

(a)

(b)

Figure 2.7 - (a) Image of a sample part blank, and (b) Resultant image after the application of a mean filter

The mean filter is thus used to reduce the noise in the input images and to blur the effects of wood grains.

3. Literature Review

As mentioned in Chapter 1, most furniture manufacturers are currently striving to reduce lot sizes as a means of improving order delivery lead times. Unfortunately, smaller lot sizes result in more frequent job changeovers. Since a machine does not produce parts while it is being set up for a new job, the production rate tends to decline as the number of set ups increase. The industry as a whole is therefore very much interested in reducing set up times. The so-called universal fixtures mentioned in Chapter 1 appear to be very promising as a means of reducing set-up time. Universal fixtures are not without their problems, however, as they typically lack locating pins that enable the machine operator to accurately position part blanks on the bed of the machine. Some systems, such as the Carter Laser Light System (http://www.carterproducts.com/) project a two-dimensional laser outline of the location where parts should be placed on the bed of the machine. However, the laser beam itself has a substantial width, and the operator still places each part blank in its approximate location and orientation on the bed of the machine. Given the fact that part location and orientation on the bed of the machine are not particularly accurate, it is common practice in the industry to oversize part blanks by ½" – 1" in the X and Y directions. This provides a substantial machining allowance to compensate for positioning inaccuracies on the bed of the machine. It also adds a substantial amount of cost to each component. In an industry with extremely small profit margins, any solution that allows manufacturers to reduce the machining allowance saves valuable dollars.

For parts that require more than one machining set-up, approximate work-piece positioning is simply not acceptable. This is due to the fact that features produced during the second machining set-up will not be accurately positioned relative to features produced during the first machining set-up. Keeping these observations in mind, the research described in this thesis advocates the use of computer vision technology to accurately identify the location and orientation of wooden part blanks on the bed of the machine prior to machining. Using parametric CNC programming capabilities, it is then possible to update offset and orientation parameters in the CNC program that compensate for the actual orientation and location of parts on the router bed. As computer vision is

the core component of the proposed methodology, the remainder of this chapter provides a detailed survey of current practices in the field of industrial computer vision.

3.1 Object Recognition

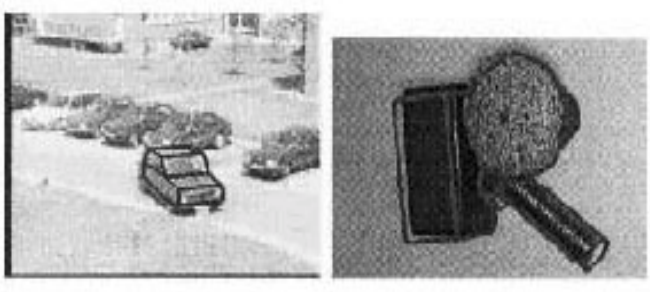
Object recognition is the sub-field of computer vision whose goal is to recognize objects from image data. Often, it is used to estimate the positions and orientations of the recognized objects in the 3D world. The images to be analyzed may be 2D gray-scale or color images or 3D range data images. Vision systems are broadly separated into three classes based on the dimensionality of objects and the data collection method. These are (1) recognition of 2D objects from 2D data, (2) recognition of 3D objects from 3D data, and (3) recognition of 3D objects from 2D data. The recognition system has two major components, namely: a modeler and a system that performs the matching of stored representations with those derived from the sensed image. Depending upon the type of modeler and matching technique used, vision systems can be further classified into different types.

The representation and creation of a model for comparison with computer acquired images for recognition has been an actively pursued area of research for quite some time. Manual construction of the object, where a human operator builds a description of the model was one of the earliest methods tried and is presented by Bolles et al [Bolles, 1986]. This method is, however, not very attractive for applications where the set of objects to be recognized is large or changes frequently.

A second approach followed by many researchers uses learning by examples, where models are constructed automatically using prototypical features extracted from images of the object to be identified. Martin et al [Martin, 1986] presented one of the first papers using this approach. They used images captured from various viewpoints to model the object. However, models built in this manner are limited in their precision by the quality of the sensor and precision of the registration techniques. The technique also suffers the same drawback as the earlier one, where, as the set of objects to be identified increases, the efficacy goes down drastically.

A method that offers great promise while being used for identification, where the number of parts to be recognized is quite large, is the one where preexisting models are adapted for recognition. Flynn et al [Flynn, 1991] have presented one of the earliest works dealing with this approach. A database of objects that are candidates to be identified by the system is created before hand. The database is populated with information that can act as decision-making variables when comparisons with images acquired from the vision system are made. These are the predominant types of model generation techniques currently used in the field of machine vision for object recognition.

As mentioned earlier, vision systems can also be classified into different types based on the techniques used for matching the data acquired from images with that extracted from the models. One of the most popular techniques is one where edge or surface matching is performed to establish a match with the model. Byne et al. [Byne, 1998] present a system where the object's model is augmented with functional information about its appearance. Some of the parameters used by Byne were color, surface finish and lighting conditions. This information is then used for surface and edge based recognition of objects of interest with hypothesis generation. A preprocessing step in this method is image segmentation, where the pixels in the image are grouped together as ones belonging to individual edges or surfaces. Some of the relevant techniques used for this are discussed. Figure 3.1 shows an example where parts are being recognized with edge and surface segmentation.



(a) (b)

Figure 3.1 - Hypothesis generation (a) Edge based (b) Surface based

(Source: Byne, 1998)

3.1.1 Edge Detection

During edge based recognition, one of the first computations performed on the image is edge detection. The edge detectors typically are linear operators, which are used to perform convolution on digital images. Edge detectors can be classified into different types depending upon the approach followed. The four distinct approaches are:

- *Gradient and difference based edge detection*: In this approach, local edges are defined as a transition between two regions of significantly different intensities. The gradient function of the image, which measures the rate of change, returns large values in transitional boundary areas. These values are thresholded to get pixels that qualify as edges. This is the approach described in Chapter 2.

- *Template based edge detection*: Edges are assumed to follow a step-like pattern. The approach to edge detection is to try to match templates of ideal step edges with regions of same size at every point in the image. Several templates are used, each representing an ideal step at a different orientation.

- *Edge fitting*: The approach to edge detection using edge fitting models an ideal edge as a step discontinuity in intensity at a given location in the image. A step function is defined for the search, and the degree to which each part in the image fits the function is computed. A threshold operation performed on the obtained values produces the pixels that form edges in the image.

- *Statistical techniques*: The edge is defined as a boundary between two inhomogeneous sub-regions. The ratio of (a) the gray level standard deviation of the combined region to (b) the product of the gray level standard deviations of both individual regions is computed. The ratio is raised to the power of the number of pixels in the respective region and thresholded to get the edge pixels.

3.1.2 Region growing

The goal of region growing is to use image characteristics to map individual pixels in an input image to a set of pixels called regions. Thus segmentation of the image into regions

partitions the entire image into quasi-disjoint regions. Various algorithms have been developed to perform the task of region growing. However, the effectiveness of the algorithm depends heavily on the application area and input image. Images with sharp contrasts can be segmented very effectively even with simple local techniques, whereas, difficult scenes such as outdoor scenes cannot be segmented satisfactorily even with the most sophisticated algorithms. Under such circumstances, region growing is used more conservatively to preprocess the image for more knowledgeable processes. Some of the more popular approaches used for region growing are:

- *Local techniques*: Pixels are placed in a region on the basis of their properties or the properties of their close neighbors. Blob coloring is a simple and elegant algorithm wherein the image is scanned with an L shaped kernel to identify various regions in the image.

- *Global techniques*: In this method, pixels are grouped into regions on the basis of the properties of large numbers of pixels distributed throughout the image. Thresholding is a common operation performed to achieve this. The value of the threshold is picked after searching the histogram of gray level of the image, and finding the minimum separating the two peaks. A variation of this algorithm first splits the original image into smaller rectangular regions, and a threshold value for each sub-region is determined.

- *Splitting and merging techniques*: These techniques merge and split regions using graph structures to represent the regions and boundaries. The merging of disjoint regions is performed using both local and global criteria.

3.1.3 Template Matching

Template matching is one of the most popular techniques used for object recognition. It is a simple filtering method used to detect a particular feature in an image. An operator called as a template can be used to detect the presence of an object if its appearance is known apriori. The technique involves translation of the template to every possible position in the image, and evaluation of a measure of the match between the template and

the image at that position. If the similarity measure is large enough, then the object is assumed to be present. The check for the object can be performed either locally or globally depending upon the template being used. It is not uncommon to use local features in the target image as templates for searching. Apart from the distinction between global and local template matching, the other aspect that subdivides this technique is the measure of similarity between the image and the template. Commonly used similarity measures are the summation of difference between the image and the template, and cross correlation. A metric based on the Euclidean distance between two sectors is also used as a measure of similarity.

The methodologies based on edge detection, region growing and template matching require edges to be distinctly discernible for the object recognition algorithms to be efficient. However, for most industrial parts, which have a smooth surfaces and not many features, solving the problem can become difficult. Cheung et al [Cheung, 1994] have addressed the problem with a system that estimates the shape from shading. Shape from shading is the analysis of shape from the brightness value or gray value of pixels in the image. The depth of the surface is computed from the surface normal of the path and its boundary conditions. Boundary conditions are obtained from the depth value at the start and end points of the path along the surface. The interpolation process during surface reconstruction, using boundary conditions and brightness information, was found to perform admirably for the task of object recognition.

As can be seen from the brief description above, many different approaches have been used for object recognition and pose determination. The use of CAD models to generate models for comparison with computer acquired images is an emerging area of research in the field. This technique promises to be very efficient especially in scenarios where part variation for recognition is quite high.

3.2 CAD based Computer Vision

In recent years, there has been considerable interest in using the ubiquitous computer-aided design (CAD) systems, which have become an integral part of the manufacturing process, in conjunction with computer vision systems. This blend will potentially enable

the vision system to make use of CAD models and features for purposes of training the vision system. CAD-based vision schemes typically take advantage of the data already available in the CAD system. One of the earliest attempts at this fusion was by Flynn et al [Flynn, 1991]. In this paper, an emphasis is laid on the topic of model building for 3D objects, with the process being automated to adopt conventional CAD models for computer vision. CAD models are used to derive the basic descriptions of the object geometry, to which inference procedures are applied to produce features that can be useful in object recognition. Geometric inference engines are used to build relational graph representations of 3D solids to infer view independent geometric properties. Polyhedral approximations are used for view dependent properties. The inferencing system is used to prune the search space for recognition.

The object database is represented as a list of objects, each with a relational graph containing a hierarchy of attributed geometric primitives that are connected by binary relations. The binary relations between two surfaces, curves or surface and curve describe surface and curve connectivity, proximity, orientation, and other relations. The paper thus presents a system that automates the process of model building for recognition. The process has been split into on-line and off-line phases. The off-line procedure allows the recognition process to become a traversal of an existing database, rather than its construction at recognition time. The identity and pose information for scene objects is calculated if the objects are represented in the object database. This information is computed and stored in a database as and when objects are modeled. View dependent models are created for sets of viewpoints in the view sphere (essentially a spherical body segmented into smaller entities), which provides a near uniform sampling of the finite space. For each viewpoint in the set, a range image of the object is generated, along with a synthetic segmentation that associates each surface point in the synthetic range image with the corresponding surface in the CAD model. Thus the task of object recognition using data from the CAD model is performed.

The generation of templates, or extraction of relevant information from a CAD modeler compatible with the vision system has many variants. Najjari et al [Najjari, 1996] present a 3D CAD based vision system for the recognition and localization of industrial parts in a

flexible assembly cell. The system uses stereo vision for obtaining 3D data about the scene. After the images are acquired, edge detection is performed and the detected edges are stored as chaincodes. The pixels in the image that are candidates for edges in the image are stored along with their location in the image. This linked list of edge points is called as the chaincode. A pruning mechanism tries to filter out noise from the detected edges when the chaincode is being built. A stereo vision algorithm is then applied for finding the recognition features. The outputs are lists of features that are combined into a 3D wireframe representing the scene. The recognition algorithm takes the observed wireframe outputs from the vision system and compares them with a set of model wireframes derived from the previous models, in order to select the best match. The models used for recognition are derived from a product data model (PDM). It is an interface between the CAD database and the recognition system which allows the automatic generation of new models used for recognition when new parts are introduced into the system.

The recognition system used in this implementation uses the paradigm of hypothesis generation and testing. The initially selected features called ‘start features’ initiate the process. A correspondence is then sought with model features. Whenever a match is found, a feature pair is constructed which references both the start feature and the corresponding model feature. A hypothesis is defined as a tuple of a model name and a geometrical mapping from the model to observation. It can be considered as the possible pose and identity of an object in the scene. During the matching step, all features in the model and observation are found that correspond according to the mapping in the hypothesis. If the match is poor or conflicts with a previous result, it is removed. For the remaining hypotheses, the mapping is improved upon using the extra information available at the current stage. This algorithm was found to increase the flexibility of the vision system, which in-turn improved the efficacy of the flexible-manufacturing cell.

Hansen et al [Hansen, 1989] introduce another alternative where geometric information obtained from the CAD modelers is used to construct a strategy tree which is used for object recognition. The strategy trees are used to provide a robust mechanism for 3D-object recognition and localization. Before processing the CAD models with these filters,

their representation is augmented with certain annotations, which include a description of the topology of the part. A winged edge model is constructed from the polygonal representation of the object serve as an index into the part. In the winged edge model (WEP), explicit information about the topology of the part is stored along with data about the various faces in the object, and the edges that constitute these faces. Information about dihedral edges, the ones that are formed at the place two faces are coincident, is also stored along with non-planar surfaces. In addition, for matching the visible surfaces with the solid model, aspect graphs and strategy trees are computed off-line, which are representative views of the object in various orientations. Once the trees have been generated and the most prominent feature identified, structural and pixel correlation are used. Structural verification refers to verifying spatial relations among the features that should be present in the scene. Pixel correlation refers to the verification technique of matching predicted depth, pixel by pixel, in a generated image and the sensed image. Thus the image is subjected to a set of filters to extract relevant information about the pose and location of the object. Summing up, the method analyzed geometric information of an object to determine the best strategy for recognition within the constraints of the sensing environment. The strategy tree that is constructed provides a model-based approach for the recognition and location of objects using 3D sensing techniques.

In addition to these approaches, researchers have tried using the CAD model renderer to generate a representation of the object. This is then used by the vision system to evaluate a match. Ude [Ude, 1997] presented a system in which models are generated automatically. The paper presents a novel technique for the integration of appearance, primitive and physics based representations of objects into a hybrid model which combines the strength of these depictions. The model consists of the root entity body that is connected to the CSG and boundary representation of the model. Faces, edges and points are the three levels of geometric entities contained in the B-rep representation. A list of nodes that contain type-specific data to describe the underlying geometry is also included. The visibility information about various features in the object is computed and stored separately since it is a time consuming activity and hence is not performed at the time of recognition. Moreover, it is also view and environment dependent. To overcome this problem, the set of all possible viewpoints are computed and hidden line removal or

ray tracing techniques are used to get the list of visible features. These visibility tables are computed by considering a sphere around the object divided iteratively into smaller regions, with each region acting as the current viewpoint. Each cell of the visibility table is connected with one of the sampled viewpoints and is indexed by this viewpoint.

The objects' description is then enhanced by the photometric properties of the material that it is made of. A special attribute concept was developed, called the hybrid object model (HOM). Attributes are collected in linked lists, with one list being attached to each geometric node. Thus, information about the photometric properties of the object's face is attached to the model. Functions for specifying the ambient, specular and diffuse reflectance and the specular and diffuse fraction of reflectance for object's faces are defined. The recognition system starts by extracting line segments from a stereo image pair. To calculate the initial hypothesis about the object identity and pose, the detected line segments are brought together into groups that correspond to the edge groups contained in the model library. The grouping consists of a depth first search in a graph like data structure. After this, the reconstructed groups are matched with the model groups. Based on these results, the 3D hypothesis about the object's pose is calculated. To verify the pose hypotheses, visible and detectable model edges are projected onto the stereo image pair, and the offsets are calculated. If these are within a certain threshold, the hypothesis about the object and its pose is accepted. This method too was found to work very effectively for object recognition and pose determination.

3.3 Statement of Need

Manufacturers are extremely interested in universal fixtures, as they can dramatically reduce set up time between jobs. However, the part placement is approximate, so part blanks tend to be substantially oversized. This wastes expensive raw materials. Furthermore, parts requiring multiple machining setups are not well suited for universal fixtures. It can therefore be said that universal fixtures solve some problems and create new ones at the same time. The need in the industry is therefore to find or develop a technology that allows more accurate part placement and orientation on universal fixtures. Generally speaking, furniture manufacturers are cash-strapped companies that

cannot afford substantial investments in equipment. The solution must therefore be economical. Likewise, the solution must not require exorbitant amounts of manpower to implement, as furniture manufacturers typically have very lean engineering staffs.

A logical solution is to use some sort of inexpensive sensing technology to measure part placement and orientation. A computer vision based system promises to address these issues within the constraints mentioned. However, traditional computer vision systems recognize a very small number of parts that they are manually trained to identify. In the furniture industry, with its large product range, the number of parts to be recognized is well over 5,000. Manual training is therefore not an entirely viable solution. Natural textures present on the surface of part blanks also present a challenge that needs to be circumvented by the sensing technology. The development of a computer vision system for increasing the productivity of furniture manufacturers was thus the focus of this thesis. A brief outline of the proposed approach for the usage of universal fixtures has been given in the next section.

3.4 Proposed Solution for Increasing Productivity

The potential of universal fixtures in increasing the productivity of furniture makers has been outlined in the earlier sections. This section gives a brief overview of the approach to alleviate the problems arising out of their usage. Universal fixtures are mostly used on CNC routers to machine wooden part blanks. In the absence of locating pins, part location and orientation is only approximate and usually performed manually. The part programs for machining the blanks hence need to be updated automatically to account for inaccuracies in the orientation and placement of the part blanks. An advanced computer vision system capable of recognizing part blanks and their location/orientation as they appear on the bed of the machine needs to be developed for this purpose. The vision system will act as the eyes, which will reduce rejection with a more stringent inspection of part orientations before machining. A brief outline of the proposed approach to enable this is given below:

- A porous sheet of MDF is placed on the bed of the router, and a vacuum is drawn through the sheet. The part blanks are thus oriented and held in place for machining.
- Based on the set of parts to be machined, an optimal nesting of parts on the bed of the router is generated with a computer program.
- A parametric CNC part program is assembled automatically based on part nesting.
- A computer image of the parts in their nesting pattern is automatically generated and then projected onto the MDF sheet with LCD projectors. Notes can be embedded into the image to instruct machine operators. These will help in part orientation.
- The operator places part blanks manually in their respective approximate locations on the router bed.
- An image of the parts on the bed of the machine is captured.
- The vision system determines the exact location and orientation of each part on the bed of the router.
- Information about part location and orientation is used to update the CNC part programs to compensate for errors in placement.

The number of parts that the vision system will need to inspect and identify is very large for a typical furniture manufacturer. CAD based computer vision was hence chosen for solving the problem. Recognition of the part and determining its orientation and location accurately is vital for the approach. Various popular computer vision techniques for recognition were therefore investigated and a brief description of the efficacy of the approaches has been included in a later chapter. Among the approaches tried, the template-based technique showed promise in tackling the problem at hand. Generation of the template for matching was the most important step in this process, and being view dependent, it was computationally challenging. A method of constraining the search space was also needed. Since CAD modelers store enough information about the object

and its properties, template generation was performed in conjunction with the data extractable from these modelers.

A detailed description of the technique for template generation and its potential to determine the location and orientation of the part has been included in a separate chapter. This chapter also discusses the validity of the constraints applied for the algorithm and the computations of the measures of similarity between the template and captured images. The mechanism to update and modify the template for different viewpoints and part orientations, as well as the technique for restricting the search space for determining part location and orientation is also discussed.

4. Preliminary Investigation

The current effort is aimed at building a vision system capable of determining the orientations of furniture part blanks as they are placed manually on the bed of a router. The information about the orientation and location of the part is used for updating the CNC codes for further machining of the blanks. A system consisting of a camera, frame grabber and suitable lighting arrangement has been integrated with various algorithms for this purpose. The camera captures images, which are stored as 8-bit gray scale images, having 256 shades of gray. Traditionally, algorithms such as the Hough transform, region growing and contour following are used for object recognition and pose determination. These have been implemented and tested with a set of sample parts. However, preliminary testing has indicated that they are inadequate for addressing the case at hand. A brief description about these implementations and the reasons for their inadequacies is discussed here. Template matching is another approach that has been investigated as part of this research, and preliminary test results were promising. The reasons why template matching performed better than the other approaches is also considered here. A highly challenging sample image of a part blank used for the testing of algorithms is shown in Figure 4.1. This example illustrates the problems encountered while using traditional approaches for the problem under consideration.

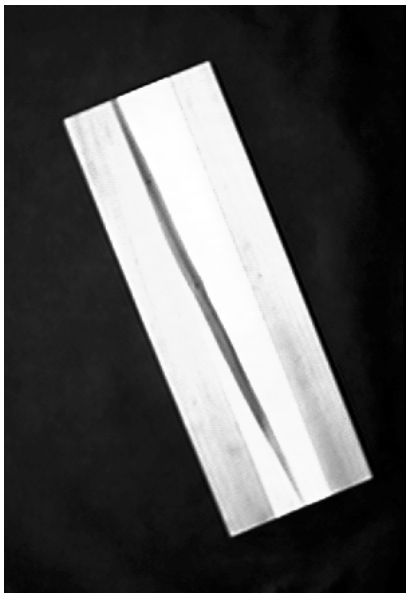


Figure 4.1 A sample image captured with the current vision setup

4.1 The Hough Transform

The Hough transform [Ballard, 1982] is a technique used to isolate curves of a given shape in an image. The classical Hough transform requires that the curve be specified in some parametric form, and hence it is most commonly used in the detection of lines, circles and ellipses, though a generalized form too has been developed. The formulation and algorithmic explanation of the Hough transform for line detection is discussed here. The equation of a straight line in parametric form is given as:

$$x\cos\phi + y\sin\phi = r \qquad \textbf{Equation 4.1}$$

where r is the length of a normal to the line from the origin and ϕ is the angle this normal makes with the X-axis as shown in Figure 4.2.

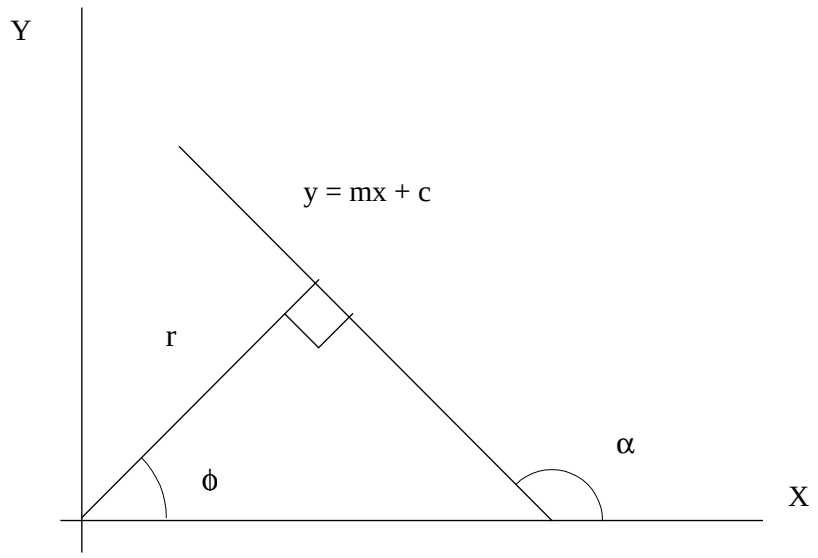
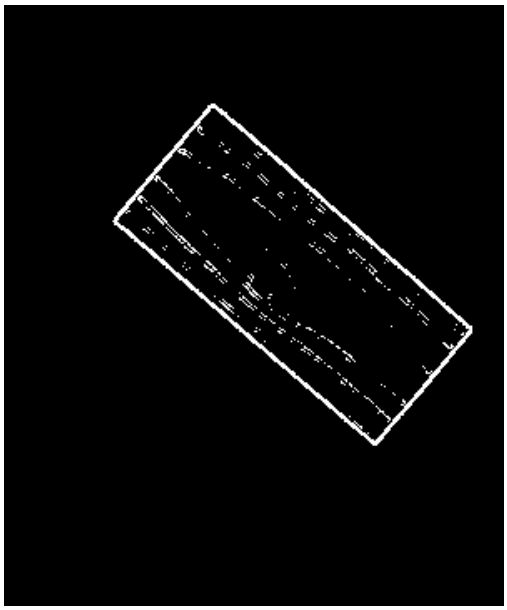


Figure 4.2 Parametric equation of a line

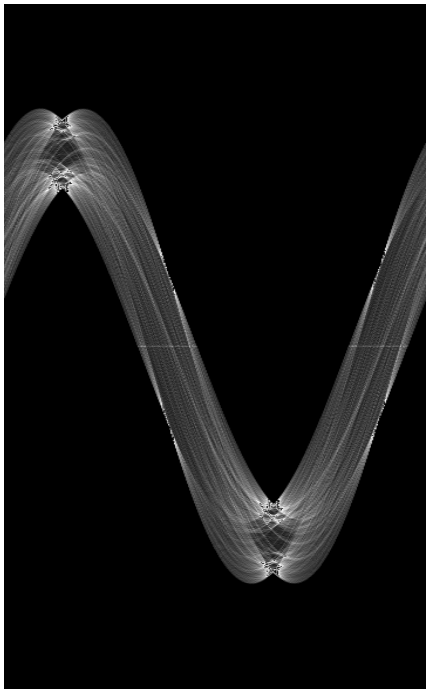
For a given line, r and ϕ are constants. In the case of a Hough transform however, we have points or pixels that lie on lines, but r and ϕ are unknown. The Hough transform identifies the values of r and ϕ for a given set of points. Thus, it clusters together pixels in the image, which can be defined by lines. The input for the Hough transform is thus a gradient image that is formed after extraction of edges in a gray scale image. Considering a point (x_i, y_i) in a gradient image, if all the possible values of r and ϕ for this point are plotted in the (r-ϕ) space (Hough space), then a sinusoidal curve is obtained. This

transformation between the image plane and the parameter space is known as the Hough transform. Thus, the Hough transform of a point in the image plane is a sinusoidal curve in the Hough space. The following figures show the construction of the Hough space for an example part.

(a)



(b)



(c)

Figure 4.3 (a) Image of a part blank; (b) The gradient image of (a); and (c) Hough space for the image in (b)

To construct the Hough space, the upper bound on r and ϕ need to be determined. The value of ϕ can only range from 0 to 360, while r, the length of the normal to a line in the image from the origin cannot exceed the length of the diagonal for the image. An array of appropriate size, called the accumulator array, is first constructed, and all elements are initialized to 0. For each pixel in the gradient image, the possible values of r for ϕ varying from 0 to 360 are computed. The appropriate point in the accumulator array is incremented for each iteration. All pixels in the gradient image thus vote for the identification of lines in the image. Collinear points in the image plane will thus give rise to transform curves which will intersect in one point. This point will hence 'accumulate' a greater value than others that do not define a line in the image. The accumulator array is then subjected to a threshold function that identifies pairs of r and ϕ that are candidates for defining lines in the image. The Hough space is usually quantized into 'buckets' of uniform size to reduce the computational burden to a certain extent. Thus, the space can be quantized into uniform buckets of say, 6 degrees each along ϕ and say, 2 units along r.

During construction of the accumulator array, ϕ is thus incremented by 6 units for each step, and r is approximated to the nearest integer along the 2 unit interval. The efficacy of the algorithm depends to a large extent on this quantization and the choice of a threshold for the accumulator array. This thus forms the first step for clustering the pixels in the image.

After the accumulator array has been created, and candidate (r, ϕ) pairs identified, an inverse Hough transform needs to be applied to identify the lines in the gradient image. This procedure is referred to as de-Houghing the image. A few interesting properties about the point-to-curve transformation that enable the clustering of pixels are given below [Duda, 1972]:

- Each pixel in the image corresponds to a sinusoidal curve in the parameter space.
- Each point in the parameter space corresponds to a straight line in the image.
- Pixels lying on the same straight line in the image space correspond to curves intersecting at a point in the parameter space.
- Points lying on the same curve in the parameter space correspond to lines through the same pixel in the image space.

To de-Hough the image, candidate (r, ϕ) pairs are first identified. The lines identified in the process are infinite and hence extend from the minimum to the maximum possible value along the height or the width in the image. Also, the slope of the line tan(α) can be easily computed given ϕ. From Figure 4.2, the possible values of α are identified as:

$$\alpha = 90 + \phi \text{ for } 0 <= \phi < 90 \qquad \textbf{Equation 4.2}$$

$$\alpha = \phi - 90 \text{ for } 90 <= \phi < 180 \qquad \textbf{Equation 4.3}$$

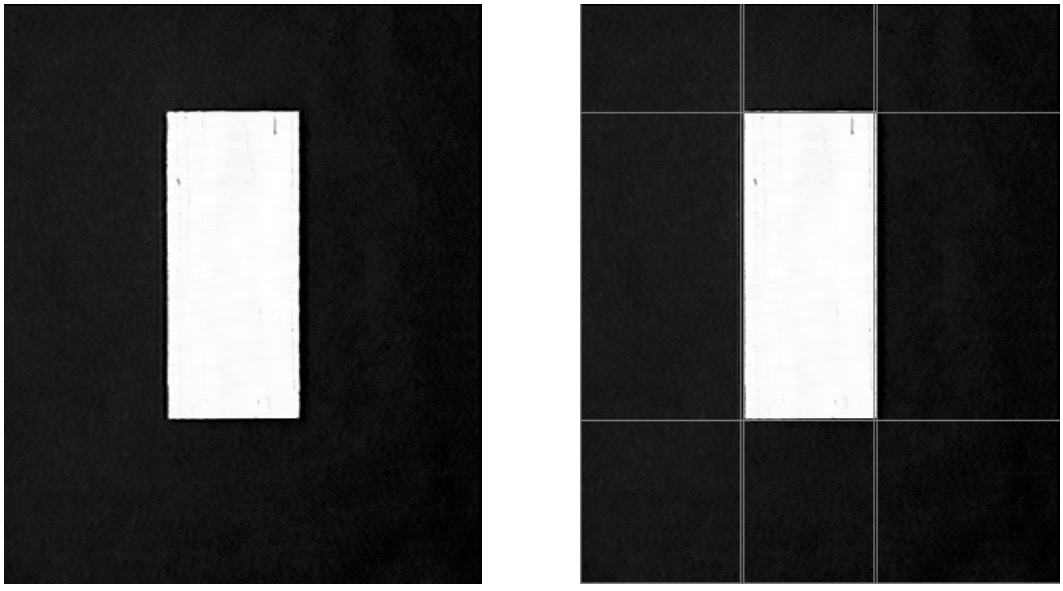
$$\alpha = 270 - \phi \text{ for } 180 <= \phi < 270 \qquad \textbf{Equation 4.4}$$

$$\alpha = \phi - 270 \text{ for } 270 <= \phi < 360 \qquad \textbf{Equation 4.5}$$

The point at which each line intersects *y = height* or *y = 0* and *x = width* or *x = 0* can now be determined from Equation 4.1, since r, ϕ and either of *x* or *y* is known. From this point, each pixel along the line can be determined by an interpolation along the height and computation of the corresponding value along the width. In this implementation, an increment or decrement of 1 is performed along the height. At each step, the corresponding width *x* is computed from Equation 4.1 as:

$$x = \frac{r - y * \sin\phi}{\cos\phi} \qquad \textbf{Equation 4.6}$$

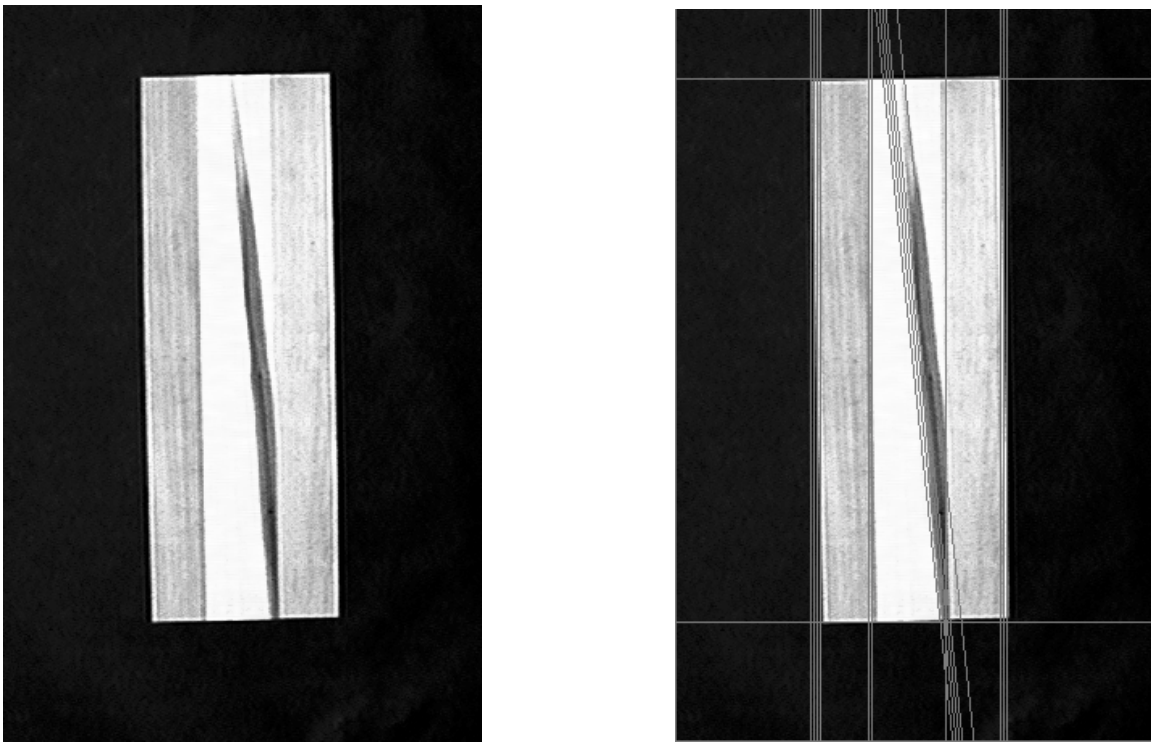
The lines can also be plotted using information about the slope of the line as computed in Equations 4.2-4.5. The bucket size used in this implementation was 6 degrees along ϕ and 2 units of length (this depends on the resolution of the vision setup) along r. The application of the Hough transform and the subsequent de-Houghing produced the results that were promising for some of the part blanks. Figure 4.4 (a) and Figure 4.4 (b) show an example part blank and the corresponding de-Houghed image. The value of the threshold used for the specific problem was 25 percent of the maximum value in the accumulator. This value was found to be 313.5 and was determined from experimentation.



(a) (b)

Figure 4.4 (a) A sample part blank, and (b) The de-Houghed image with pixels clustered along lines

While the results shown in Figure 4.4 appear to be promising, the Hough transform did not perform well in cases where the part blank had a pronounced wood grain pattern. Moreover, the value of the threshold that is used with each image varies and can only be determined from experimentation. Thus it is difficult to determine the object's pose without any input from the user. The wood pattern can also lead to a de-Houghed image with numerous spurious lines. These spurious lines make it even more difficult to determine an object's pose. The Figures 4.5 (a) and (b) provide an illustration of this problem. The value of the threshold used in this case was 37 percent of the maximum value in the accumulator. This value was determined as 462.87.



(a) (b)

Figure 4.5 (a) A part blank with pronounced wood grains; and (b) The de-Houghed image.

The Hough transform method for determining the object’s pose in the current scenario was hence found to be inadequate. The task of trying to remove unwanted edge pixels is very difficult because of the nature of the noise. These pixels are present in the image after edge extraction because of the pattern that is generated by wood grains. The texture

of wood grains varies from one species to other and also depends largely upon the direction in which the specimen has been cut. Thus modeling the patterns and then removing them was almost impossible. The noise in these images is present only on the inside of the actual specimen. However, it was not possible filter this just by removing pixels on the inside of the bounding edges as details about part geometry would have been lost. Moreover, the Hough transform requires explicit searching for different types of geometric entities. In the figures above, only straight lines are considered. In the presence of curved edges on the part, an explicit search for each curve has to be performed. This can prove to be a very difficult task to accomplish because of the fact that many furniture parts have lots of ornate designs. These were the main issues that prevent the use of the Hough transform in this approach. Region growing is another approach that is used frequently for object recognition by surface based segmentation. This was hence used as an alternative approach, the details of which are given in the next section.

4.2 Region Growing

The goal of region growing is to use image characteristics to map individual pixels in an input image to sets of pixels called regions. An image region can correspond to a world object or a meaningful part of one. Thus, region growing provides a means to segment the image into blobs or chunks of pixels. That is, regions do not have dimensional overlaps, and no pixel belongs to the interior of more than one region. One of the region growing techniques that has been implemented as part of this research is the Blob coloring algorithm [http://www.cs.rochester.edu/u/brown/172/assts/UFind.html]. It was hypothesized that with this procedure, it would be possible to identify the various faces in the captured image. This could then be used for the calculation of metrics such as the surface area of each face. Information about the surface area would be used in conjunction with the CAD modeler to relate the computed areas with those of surfaces in a given object. Thus the orientation of the object would be determined depending upon the location of faces in the object. The algorithm for the technique is explained as follows.

An L shaped template is placed at each pixel in the image, and the algorithm is applied to each pixel. The template for the algorithm is:

$$\begin{matrix} & X_U \\ X_L & X_C \end{matrix}$$

where X_C is the current pixel. X_L stands for the pixel to the left of the current pixel and X_U is the pixel above the current pixel. The image is then scanned from left to right and top to bottom. The L-shaped kernel used in the algorithm looks at 4-connected regions. Since an L shaped kernel is the simplest possible shape, it has been chosen. Another possible shape would be a + shaped kernel. However, the location of the central pixel to cover the entire image can be a challenge. The concept of a 4-connected and an 8-connected region is as follows:

- *4-connected region*: The region obtained by a series of 4 way movements along north, south, east and west is called as a 4-connected region. In Figure 4.6, the central pixel is said to be 4-connected to the other pixels.

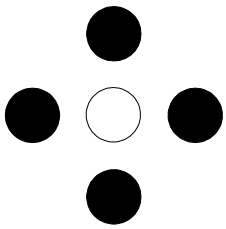


Figure 4.6 A 4-connected region

(source: http://www.cs.ucdavis.edu/~ma/ECS175_S00/Notes/0411_a.pdf)

- *8-connected region*: The region obtained by a series of 8 way moves along north, south, east, west, north-east, north-west, south-east, south-west is called as an 8-connected region. In Figure 4.7, the central pixel is said to be 8-connected to the other pixels.

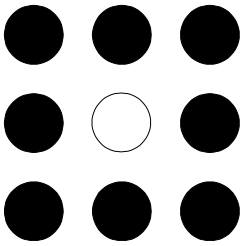


Figure 4.7 An 8-connected region

(source: http://www.cs.ucdavis.edu/~ma/ECS175_S00/Notes/0411_a.pdf)

In order to apply one of these kernels with the region growing algorithm, the gray scale image is first converted to a binary image by applying a thresholding function. The value of the threshold is computed programmatically as listed in Chapter 2. This binary image is the input for the blob-coloring algorithm. The coloring scheme to identify blobs is as given below:

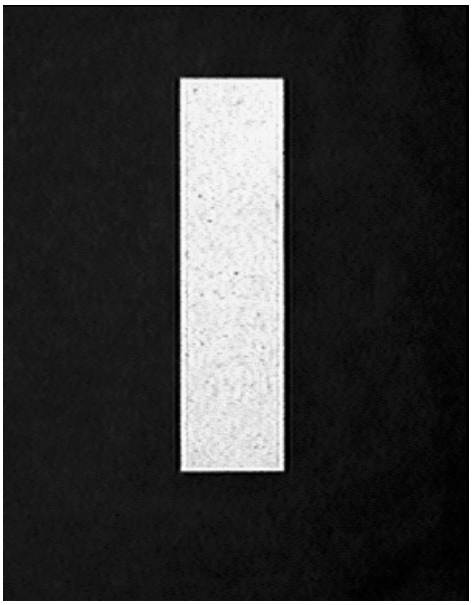
Let the initial color *k* be 1.

```
for(i = 0; i < height; i++){
    for(j = 0; j < width; j++){
        X_U = (width*(i-1))+j;
        X_L = (width*i)+j-1;
        X_C = (width*i)+j;
        if(f(X_C) != 0){
            if( f(X_U) == 1 && f(X_L) == 0)
                color(X_C) = color(X_U);
            if( f(X_L) == 1 and f(X_U) == 0)
                color(X_C) = color(X_L);
```

```
                    if( f(X_L) == 1 and f(X_U) == 1){
                        color(X_C) = color(X_L);
                        blobcolor = color(X_L);
                        colorU = color(X_U);
                        for(k = 1; k < height; k++){
                            for(m = 1; m < width; m++){
                                X_n = (width*k)+m;
                                if(color(X_n) == blobcolor)
                                    color(X_n) = colorU;}
                        }
                    }
                    if(f(X_U) == 0 && f(X_L) == 0){
                        color(X_C) = k;
                        k++;}
                }
            }
        }
```

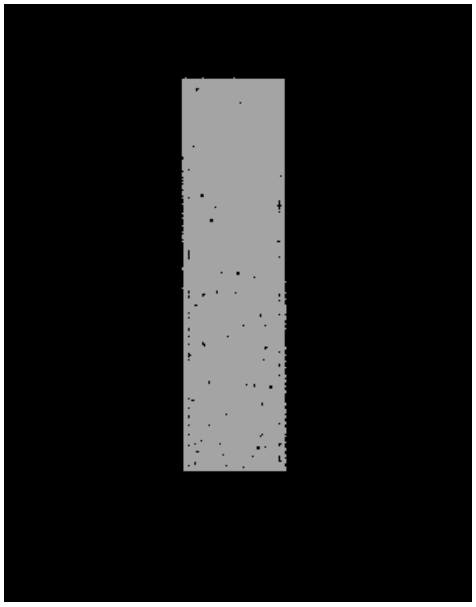
In the algorithm above, *f(A)* returns the brightness at the pixel given by *A*. The function *color(B)* assigns a numeric value to the pixel specified by *B*. It thus assigns a color to the pixel according to the "blob" that it belongs to.

This algorithm was applied to images of wooden part blanks in order to segment the images into surfaces. The results were quite encouraging for part blanks that did not display significant wood grain. Figure 4.8 (a) shows an example of a part for which the algorithm is well suited. The thresholded image was generated using a threshold of 191. The resultant thresholded image is shown in Figure 4.8 (b), whereas Figure 4.8 (c) shows the blobs that were identified in the image.



(a)

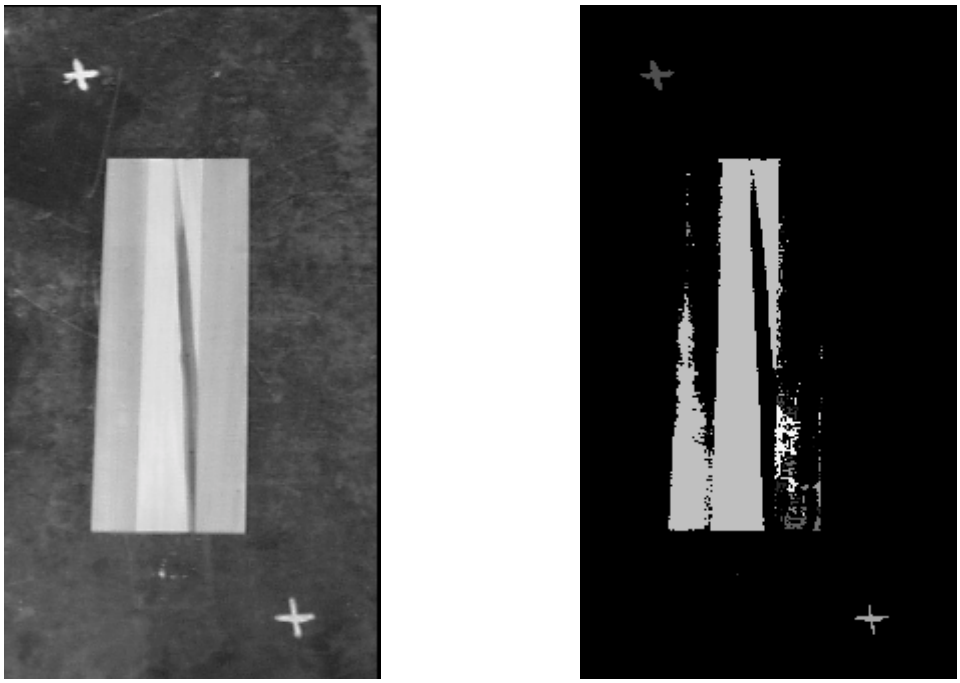
(b)



(c)

Figure 4.8 (a) Part blank of a furniture piece; (b) The resultant image upon thresholding (a); and (c) Blobs in the image

In contrast to the results shown in Figure 4.8, the blob coloring algorithm gave unsatisfactory results when applied to images displaying pronounced wood grain. This is due to the fact that the wood grain leads the algorithm to identify numerous false regions within a part blank. Figure 4.9 (a) illustrates a part blank which highlights the inadequacy of the blob coloring technique for the current application.



(a) (b)

Figure 4.9 (a) Part blank for a drawer front; and (b) Delineated blobs for the image in (a)

4.3 Contour following

The contour following algorithm [Vernon, 1991] is an approach that uses no domain-dependent information and 'follows' the boundary or contour solely on the basis of locally derived data. The technique essentially starts with a pixel or point that is believed to be on the contour of the object, and grows the boundary by adding neighboring pixels in the contour direction. A gradient image resulting from the application of edge detectors on a gray scale image thus forms the starting point for the algorithm The contour direction that is followed is usually the direction perpendicular to the gradient at the current pixel. Since there is no prior information about the contour and its properties, the algorithm searches for neighbors in both directions perpendicular to the gradient direction. By convention, the forward direction is defined to be +90 degrees with respect to the gradient direction. Likewise, the reverse direction is defined to be –90 degrees with respect to the gradient direction. The selection of a candidate pixel to be added to the

boundary depends on the deviation of the gradient direction from one point to another. The threshold used for making this decision has a profound effect on the results of the algorithm. There usually is no set rule to select this threshold, and the value is typically chosen from experimentation. For this implementation, a threshold value between 3.15 and 5.15 degrees was used. This pruning is required in order to avoid following boundaries into noisy areas. The noise in an image is usually randomly distributed, hence the change in gradient from one pixel to another is likely to be haphazard.

During the tracing phase of the algorithm, a zone with a radius of 1 pixel is first searched. If this search does not return a suitable candidate, then the search zone is increased to be one with a radius of 2 pixels. The algorithm also takes into account the fact that the next candidate pixels need not necessarily lie along the direction perpendicular to the gradient direction. This can happen when lines that are at an angle to the horizontal are being rasterized. The algorithm hence checks the adjoining pixels in the counter clockwise sense according to the concept of Freeman direction codes [Vernon, 1991]. The Freeman direction codes provide a means to search for potential neighbors to the current pixel. These are used in deciding the next pixel that should be tested for being a potential candidate to extend the contour. The search is carried out first at unit radius, and then extended to search at a distance of two units. Figure 4.10 (a) and Figure 4.10 (b) illustrate the concept of the radius of a search zone and the Freeman code. The application of the Freeman code leads to the development of boundary chain codes.

Figure 4.10 (a) Freeman direction codes for a zone of radius 1; and (b) Freeman codes for a zone of radius 2

In the figure, *C* stands for the current pixel and the numbers give the order in which each of the neighbors is checked so as to add it to the contour. As can be seen from the codes, the search direction is counter clockwise. This is done in order to prevent the algorithm from searching randomly. If the continuity of the boundary codes is broken for some reason, the next boundary point from the gradient image is chosen as the starting point. The boundary following algorithm terminates when the search procedure does not return any valid boundary point, or the original starting point for the algorithm is the next candidate, or all the boundary points in the gradient image have been used as starting points for the algorithm. Once the boundary chain codes are generated, the pixels need to be arranged in a contiguous manner so as to delineate the lines in the image. This is the most important task in the algorithm, as it segments the image at the edge level. The pseudo code to enable this is given below:

The contour following algorithm first tries to generate contiguous lines of pixels by searching for neighbors depending on gradient information. It then tries to club these lines together with the function *CombineLines*

```
Edge_Points = GetEdgePoints(Input_image);

for(i = 0; i < height; i++){

   for(j = 0; j < width; j++){

      if(Edge_Points(j, i) == 1) {

         start.x = j;

         start.y = i;

         if(check[(width*i)+j] == 0){

            next_pixel = GetNextPixel(start, 1);

            if(next_pixel != NULL) {

               check[(width*start.y)+start.x] == 1;
```

```
        new_line.First_Pt = start;
        new_line.Second_Pt = next_pixel;
        size = pTheLine.GetSize();
        if(size == 0)
           AddSegment(pTheLine, new_line);
        else{
            result = Check_continuity(pTheLine, new_line);
          if(result == continuos)
                AddSegment(pTheLine, new_line);
          else{
                segments = pTheLine.current_segments;
                if(segments > 8)
                      AddLine(pTheLine, AllLines);
          }
      }
  }
 else{
       next_pixel = GetNextPixel(start, 2);
‘ Repeat the same procedure with search radius of 2
  }
```

```
        }
      }
  }
```

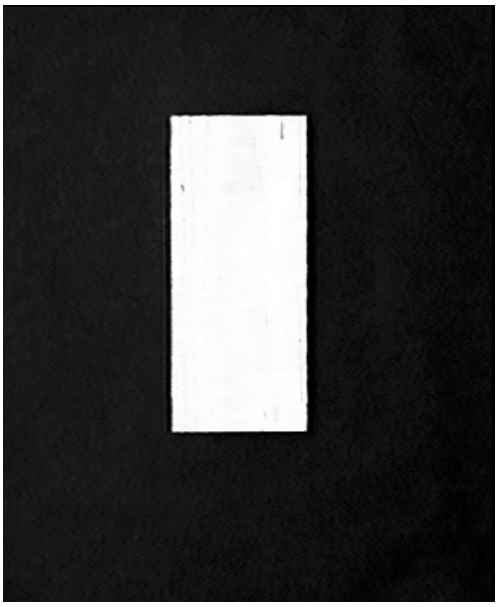
Final_Lines = CombineLines(AllLines);

The function *GetEdgePoints* applies the Sobel operator discussed in Chapter 2 to the input image and extracts candidate edge pixels from the image. The function *GetNextPixel* searches for neighbors following the Freeman direction codes from the current pixel. It also does the search at a radius of 1 and/or 2 pixels. This function also applies the threshold for deviation from the current gradient direction. This is required in order to prevent following the contour into noisy areas. The threshold applied has been mentioned above. The array *check* is used in order to check if the pixel returned by *GetNextPixel* has already been visited. The variable *pTheLine* is a pointer to a data structure that holds to starting and ending points of the various segments that make up the current line, and is a dynamic array. Similarly, the variable *AllLines* holds the various lines in the image, and this too is a pointer to a data structure to hold the various line segments. It is also a dynamic array and grows in size as required. A line has been defined as a contiguous set of pixels, where the number of segments in the line is greater than 8 as has been explained above. The function *Check_continuity* checks if the segments are contiguous merely by checking the distance between the ending point of the previous segment and the starting point of the current segment. Thus the algorithm generated the Freeman direction codes and creates line segments that are contiguous.

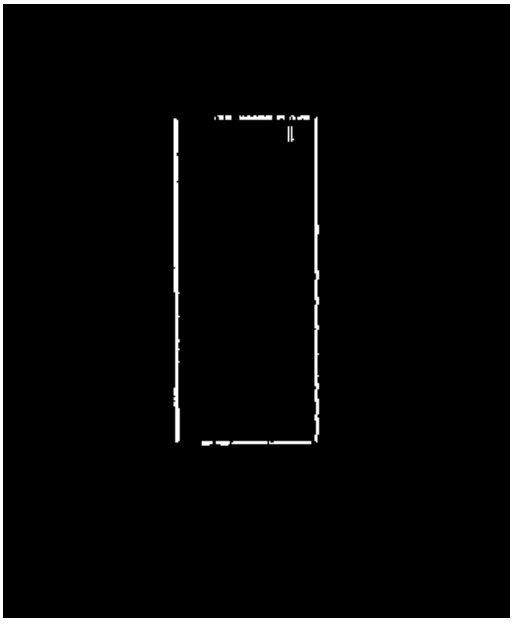
In the function *CombineLines*, the algorithm tries to club together line segments that might lie along the same line and try to segment the image into various lines. It thus tries to perform an edge-based segmentation of the image. This is required because of the fact that contiguous line segments returned from the previous part of the algorithm tend to have lines that could be clubbed into a single line but are segregated because of some missing pixels in the interim. The function again follows the Freeman direction codes to choose the pixels that need to be considered as potential candidates that can be used to

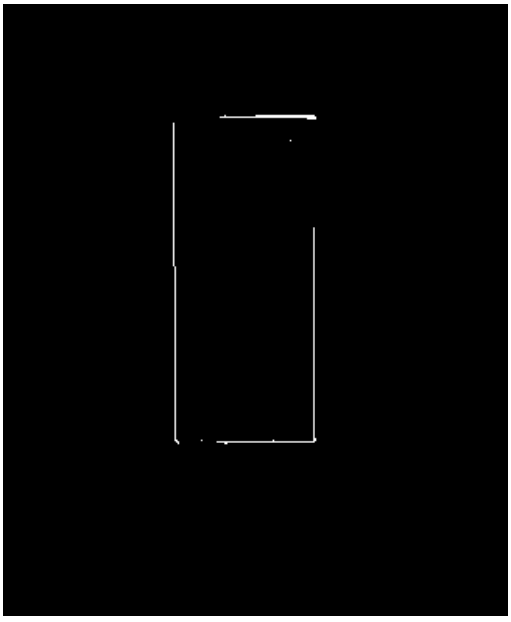
extend the contour. The function chooses each line segment stored in the array *AllLines*, and then tries to add edge point from *Edge_Points* choosing the pixel to be tested based on the Freeman direction codes. The threshold for pruning pixels based on noise is not applied. This was an extension to the algorithm that was developed since the results from the earlier part of the algorithm were not very promising. It was hypothesized that the gradient information generated by the Sobel operator was probably not very accurate, and hence *CombineLines* aimed at filling gaps among the already segregated lines in the image was developed.

In the current implementation, the boundary codes are generated as explained above. Edge based segmentation is performed next by clustering the pixels identified as boundary pixels. During the contour following procedure, the pixels are clustered into arrays as contiguous sets. Thus, the first pixel for the contour following process initiates the array. The neighbor that is identified by the search procedure is added next to this array. The array expands as long as a suitable neighbor is obtained. This maintains the continuity of the line segment. Once the continuity is broken, the size of the current line segment is checked. If the number of pixels in the segment is greater than a certain threshold, the line is stored in the array. Otherwise, it is discarded. The value for the threshold that was chosen for this implementation was 8 pixels. This value was obtained from experimentation, and the results of the algorithm depend heavily on this value. After the array of contiguous line segments is created, they are clustered together to form line segments which are continuous. To enable this, the Freeman boundary codes are used to search for segments that begin from one of the neighbors suggested by these codes. Thus, depending upon the local gradient direction, the location of the next pixel is estimated by checking the directions perpendicular to the gradient direction in both the forward and reverse directions. If a match is obtained, and this pixel forms the start point of another line segment, then the segments are merged. This process is continued until all segments in the original array have been checked. Figure 4.11 (a) shows a simple example part that was used to test the contour following algorithm. Figures 4.11 (b) and Figure 4.11 (c) show the image after the contour following and pixel clustering steps respectively.



(a)





(b) (c)

Figure 4.11 (a) A sample part blank; (b) Image generated upon application of contour following on (a); and (c) Lines identified after clustering pixels in (b)

In Figure 4.11 (c), the algorithm has clubbed the pixels together and identified 8 lines. This means that spurious edges have been identified by the clustering algorithm. This is

possibly due to discontinuities in the contours in the image in Figure 4.11 (b). The contour following algorithm was thus found to be inadequate to handle the current task.

4.4 Template Matching

Template matching is essentially a filtering method used to detect specific features in an image. However, it requires apriori information about the exact description of the features as they appear in the image. It is a commonly used method for pattern classification. Moreover, template matching is a robust technique in that it works quite well even with noisy images, especially when the variations within a class are due to additive noise. A sample application of template matching is shown in Figure 4.12.

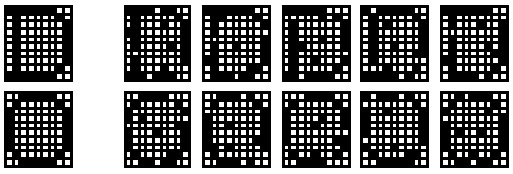


Figure 4.12 Illustration of the template matching procedure

(source: http://www.engr.sjsu.edu/~knapp/HCIRODPR/PR_simp/template.htm)

The template-matching algorithm requires a template that is compared with the target image for one or more specific features. The template image that is used for comparison hence needs to be exactly similar to the feature of interest. Generation of the template for matching is therefore an extremely important step in this process. Most instances of template matching call for the vision system to be trained to locate specific features in an image. To compute the presence or absence of the feature in the images, a similarity measure is computed, which reflects how well the image data matches the template for each possible template location. The point of maximal match is selected as the location of the feature. Two of the simplest and most popular metrics used for determination of the match are:

- *Number of agreements*: This metric involves counting the number of black and white pixels in the template that match with black and white pixels in the target image respectively (with reference to Figure 4.12). The number of matches is subjected to a

threshold and the match that gives the maximum count is selected. The choice of threshold usually depends on the images that are used with the algorithm. This is the maximum correlation approach.

- *Number of disagreements*: In this metric, the number of black pixels in the template that correspond to white pixels in the image, and the number of white pixels in the template that correspond to black pixels in the target image are counted (with reference to Figure 4.12). The match that gives the minimum number of disagreements is chosen. This is called as the minimum error approach.

Since the template matching approach inherently accounts for noise, it lends itself suitably for recognition of part blanks with patterns of wood grains. A template-matching algorithm was hence implemented and tested with some sample part blanks. The results from these tests were encouraging, and hence it was chosen as the approach for tackling the problem at hand. The biggest challenge with this approach, however, was construction of the actual template to be compared with the images captured from the vision system. Since typical furniture manufacturers have a large number of parts that need machining on the router, the number of parts to recognized with the vision system is quite large. Manual training of the vision system is therefore not a practical solution. It was hence decided to adopt a CAD-based template generation and image recognition approach. Details about the generation of the template, the choice of a metric for matching, and the mechanism for automatically updating the template to account for variations in orientations are described in the next chapter.

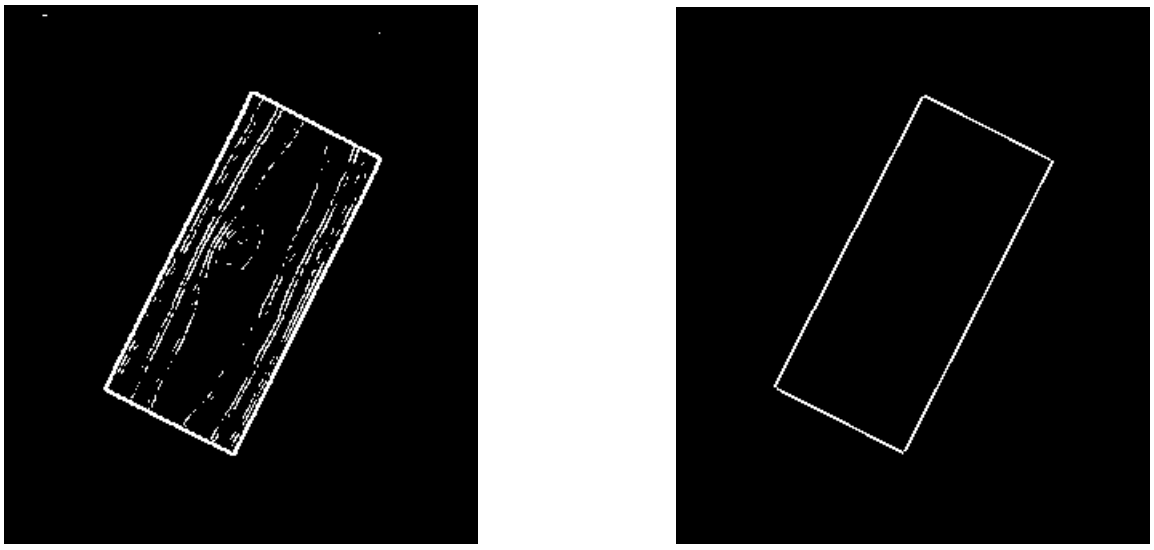
5. Methodology

5.1 Overview

As described in Chapter 4, some of the most popular approaches for object recognition and pose determination are not particularly well suited for use with furniture part blanks. It was found that wood grains produce patterns on part surfaces that can be confused for part edges. Moreover, image segmentation algorithms misinterpret the changes in brightness resulting from wood grains. Template matching with its inherent tolerance for noise in the image was found to have promise for the application. However, template matching, when used for object recognition, requires apriori knowledge of the object's exact appearance. Conventionally, a database of particular features in an object, or its appearance in a given orientation is generated beforehand during what is called the training phase. For each image, the template-matching algorithm searches through the template database and tries to make a match with the object as it appears in the captured image. A typical furniture manufacturing setup has thousands of parts that are machined on the router. Creating a template database of these parts as they appear in various orientations was therefore deemed to be infeasible. Moreover, the addition of new parts for machining with the router would require an update of the database. In the furniture industry, where styles come in and go out of fashion very quickly, this would be very labor intensive. Generation of the template in a way that was reasonable was hence the most significant hurdle for pose determination with this approach. An incremental algorithm where the template is generated automatically was hence developed. A detailed description of the methodology followed is presented here.

The proliferation of technological advances in furniture design and manufacture has made CAD modelers quite ubiquitous with furniture makers. These CAD models store much information about the geometry and appearance of each part. Moreover, since CAD modelers are used for designing furniture, they present a readily available database of the entire product range. It was hence decided to use a CAD-based approach to generate the template for pose determination. Using information about the object's geometry and appearance for generation of the template was however, a significant challenge. Wood

grains present on the surface of furniture part blanks create natural patterns. Generating these patterns on part surfaces to mimic the object's photometric properties was found to be impractical. This was primarily due to the large variations in patterns that are found naturally on wood parts. Moreover, mapping these textures on part surfaces for rendering was computationally intensive on account of its dependence on part orientation and geometry. It was hence decided to have a template that looked similar to the image resulting after the application of edge detection algorithms on the captured image. The images shown in Figure 5.1 (a) and Figure 5.1(b) the approach that was taken.



(a) (b)

Figure 5.1 (a) Resultant of edge detection on a sample image, and (b) A sample template for pose determination of image in (a)

A brief overview of the methodology that was developed for template generation and pose determination, is presented here. The flowchart in Figure 5.2 gives an outline of the procedure. A description of the steps involved is included later in the chapter.

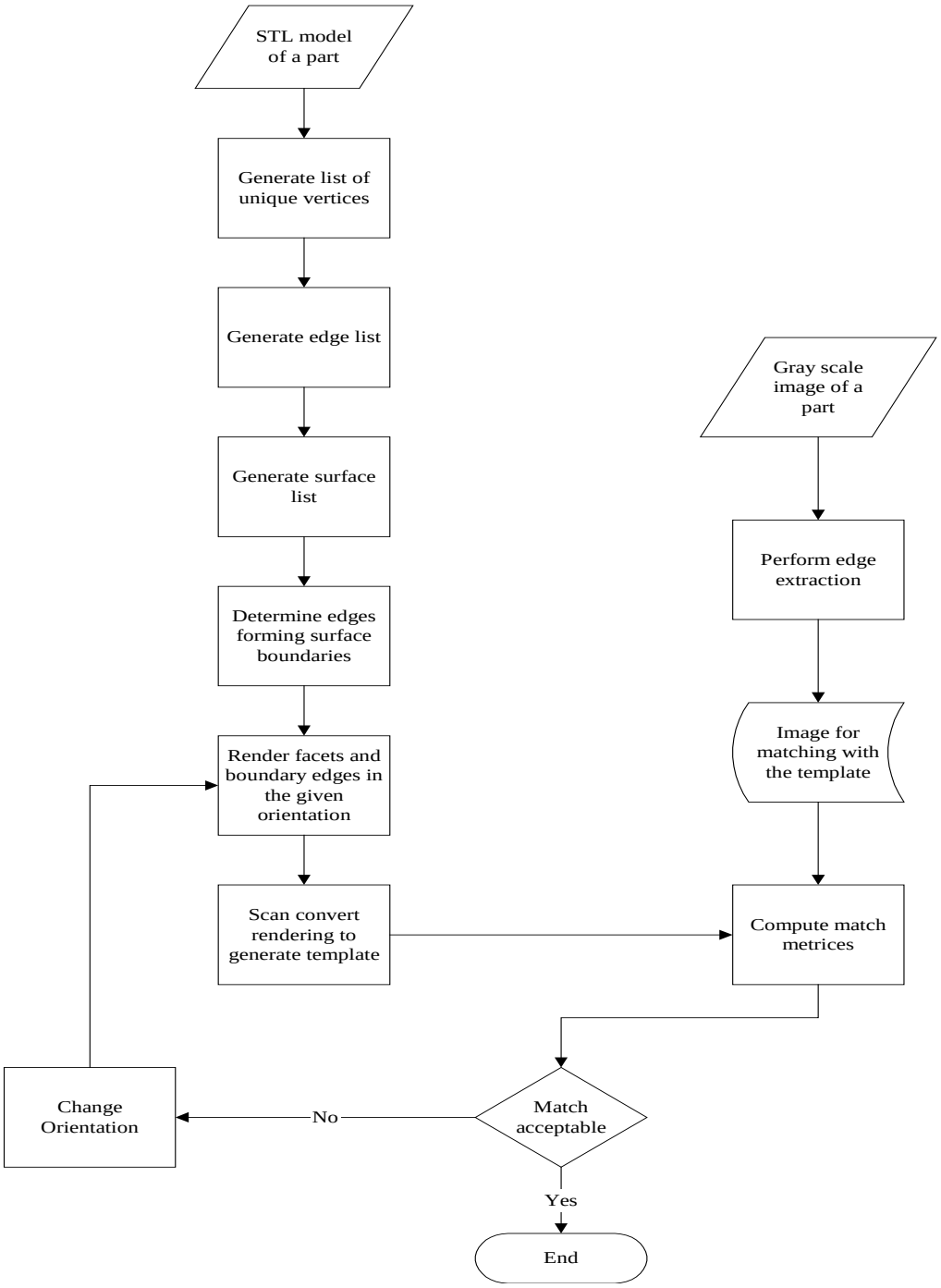


Figure 5.2 Methodology for template generation and pose determination

In order to generate templates from solid models as shown in this procedure, considerable pre-processing is required. The topology of the part needs to be generated as defined by Rock et al [Rock, 1992]. This involves generation of a list of unique vertices, an edge list and a surface list. These lists are then used for determining edges that form the

boundaries of each surface. Edges on surface boundaries are important because these are the geometric entities which are returned by the edge detection algorithm when it is applied to an image of the object. Moreover, only some of the edges are visible in the captured image, as it is a function of the orientation and location of the part with reference to the camera. Thus, a simulation of the working of a camera is needed to generate view dependent templates so as to enable pose determination. A detailed description of the algorithms developed to enable this is provided in this chapter.

5.1.1 CAD Models

CAD modelers store data about a part's geometry and appearance in different file formats. Because the work described in this document is intended to be universally applicable, the use of a widely available platform independent file format was desired. The stereolithography or STL file format was chosen as the preferred format for this implementation. One of the main reasons for choosing this format was the relative ease with which information about a part's geometry can be extracted from an STL file. In an STL file, a process of tessellation approximates the part geometry. The solid model is approximated with a number of triangular facets that are mapped onto the solid model so as to form a manifold tessellation. According to the definition of a manifold surface, exactly two facets share each edge of the triangles in the tessellation [http://www.unchainedgeometry.com/jbloom/dissertation/chptr4-nonman.pdf]. The STL file stores information about the vertices and the normal for various facets in the tessellation. The facet normals provide information about surface orientations. This is useful for generation of view dependent templates for pose determination. A detailed description of the STL file format is included in the appendix. Figure 5.3 illustrates the triangular mesh mapped on the surface of a part in an STL file.

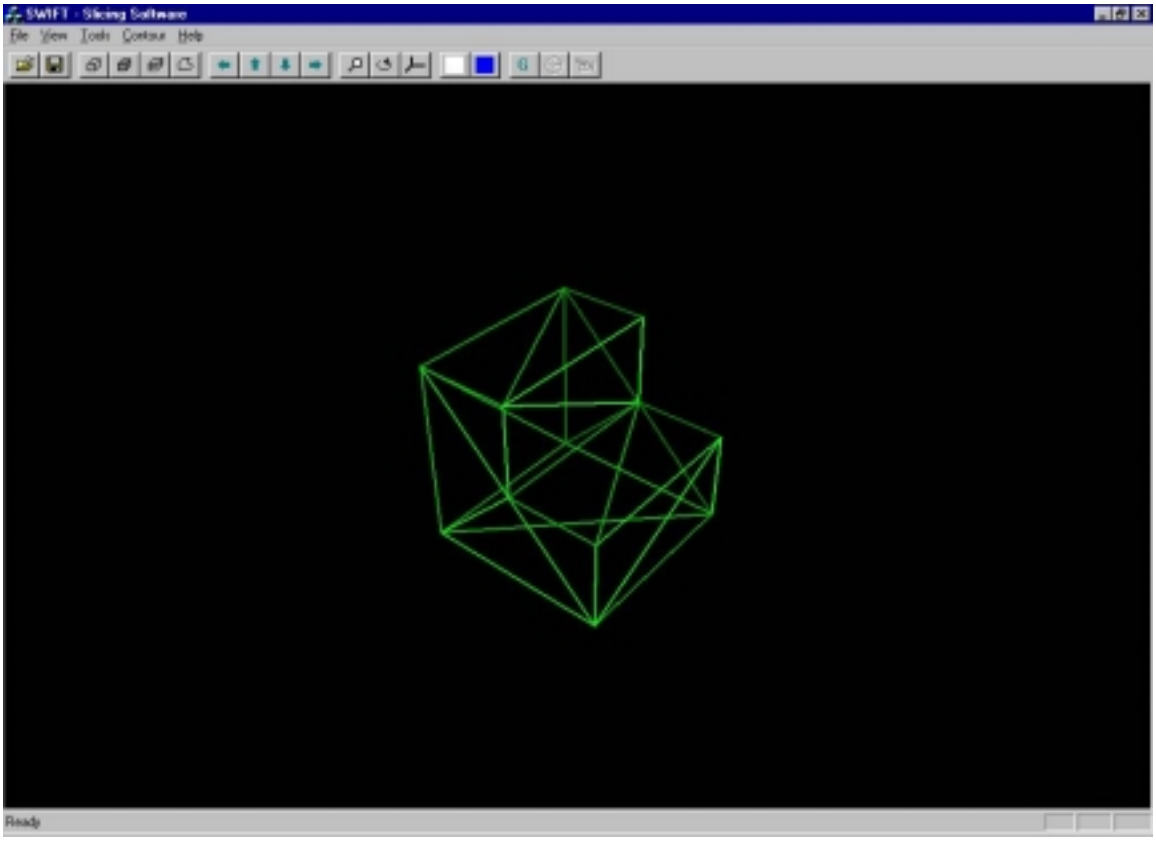


Figure 5.3 Tessellations in an STL file

5.2 Template generation

5.2.1 Generating the list of unique vertices

The STL file contains data about the vertices that define the facets in the tessellation. Each vertex of the tessellation is shared by more than one triangle. The number of triangles sharing a vertex is indeterminate and depends solely on the geometry of the part. Three vertices define each facet in the STL file, and hence vertices that are shared by different facets are repeated in the file. It is therefore necessary to generate a list of unique vertices so as to enable computation of the edge list and surface list. The algorithm for generating the list is fairly simple, and begins by storing the vertices of the first facet in the file. All the vertices of the remaining facets are then compared with those already stored. The metric for comparison is the distance between the vertices in the list and the new vertex which is to be added. An allowance for incorrect mapping of tessellations on the part surface has also been provided. Thus, if the distance between the

new vertex and any other vertex in the list is less than or equal to the allowed distance, it is marked as 'included' in the list. The allowable distance that was chosen for this implementation was 5 microns. Upon comparison, if the vertex is not found in the list, it is appended to the end of the array. The pseudo code for generation of the vertex list is:

```
vertex_list[0] = facet[0].vertex[0];
for(i = 0; i < Facets in file; i++){
        for(j = 0; j < 3; j++){
                added_to_list = false;
                for(k = 0; k < v; k++){
                        distance = GetDistance(facet[i].vertex[j],vertex_list[k]);
                        if(distance <= tolerance value)
                                added_to_list = true;
                        if(added_to_list == false){
                                vertex_list[v] = facet[i].vertex[j];
                                v++;}
                }
        }
}
```

In the algorithm, *v* is the number of facets in the vertex list, and the function *GetDistance* returns the distance of the j^{th} vertex of the i^{th} facet from the k^{th} entry in the array *vertex_list*. The list of unique vertices in the STL file is thus computed and stored in the array *vertex_list*.

5.2.2 Generating the list of unique edges

The list of unique edges in the tessellation is required to determine the list of surfaces and the edges that form the boundary of these surfaces. Since an STL file has a manifold tessellation, exactly two facets share each edge. The algorithm for generating a list of unique edges makes use of this and the fact that exactly two different vertices define each edge. The data structure to hold this list contains information about the two vertices that define the edge, and the facets that share this edge. The vertices are defined from the

index that is obtained from the vertex list that was generated earlier. The array is initialized to three times the number of facets in the file. The algorithm then sifts through the entire file, allocating the index of a given vertex in the vertex list to the appropriate elements of the edge list array. Assignment of the facets is done next. Since the algorithm indexes through the file with reference to the facet number, one of the facet numbers for the edge is the index of the facet that is being checked. After the end of this iteration, the edge list has information about one of the facets that the edge belongs to as well as the two vertices that define it. To identify the other facet that the edge belongs to, a search is made with reference to the vertices that define the edge. Thus, if there is another entry in the edge list that has the same vertices, but the facet number which has already been defined does not match, then facet number 1 of this edge is assigned as facet number 2 of the other. The facets that share each edge in the tessellation are thus obtained. The edge list generated at this point is however twice as big as the actual edge list. The list is finally pruned by checking the index numbers of the facets in the list and with the removal of duplicate records. The pseudo code for accomplishing the steps in the algorithm is given below:

Step 1: Initialize the array with information about the index of the vertices from the vertex list and to assign one of the facet which the edge belongs to:

```
for(i = 0; i < facets in file; i++){
        for(j = 0; j < 3; j++){
                index = Get_index_in_vertex_list(facet[i].vertex[j]);
                if(j == 2)
                        index1 = Get_index_in_vertex_list(facet[i].vertex[0]);
                else
                        index1 = Get_index_in_vertex_list(facet[i].vertex[j+1])
                edge_list[edgeno].vertex1 = index;
                edge_list[edgeno].vertex2 = index1;
                edge_list[edgeno].facet1 = i;
                edgeno++;}
}
```

Step 2: Assign the element *edge_list[edgeno].facet2* in the list :

```
for(i = 0; i < edgeno; i++){
        test_facet = edge_list[i];
        for(j = 0; j < edgeno; j++){
                if(vertices of test_facet == vertices of edge_list[j] &&
                test_facet.facet1 != edge_list[j].facet1){
                        edge_list[totalno].facet2 = edge_list[j].facetno1;
                        totalno++;}
        }
}
```

Step 3: Prune and return the edge list:

```
for(i = 0; i < totalno; i++){
        added_to_list = false;
        for(j = 0; j < edges_in_list; j++){
                if(facets of edge_list[i] == facets of final_edge_list[j])
                        added_to_list = true;
                if(added_to_list == false){
                        final_edge_list[j] = edge_list[i];
                        edges_in_list++;}
        }
}
```

The array *final_edge_list* contains the actual edge list. The number of unique edges in the tessellation is obtained from *edges_in_list*. Thus the list of unique edges is generated.

5.2.3 Generating the list of surfaces

The list of surfaces is required so as to enable identification of edges that are shared by two different surfaces. These are the edges that lie along the boundary of the surface. Since the STL file does not contain feature information, faces need to be identified with a heuristic procedure as presented by Cormier et al [Cormier, 2000]. The procedure is based on the observation that the boundary between adjacent faces is characterized by abrupt changes in surface curvature. Thus a change in the surface curvature between adjacent facets indicates that the facets might belong to different surfaces. The algorithm computes the acute angle between adjacent facets shared by an edge. If the angle is above a threshold, then the edge separating the facets is presumed to lie upon the boundary between adjacent surfaces in the part. This is used to identify all the facets that lie on a given surface, and also to determine the various surfaces in the part. The threshold used for the angle was 37 degrees. Assuming that an edge is shared by facets with surface normals *U* and *V* respectively then the value of the angle between the facets is computed from the dot product as:

$$U \bullet V = \|U\|.\|V\| \cos\theta = u_1 v_1 + u_2 v_2 + u_3 v_3$$

$$\text{where } \theta = \cos\left(\frac{u_1 v_1 + u_2 v_2 + u_3 v_3}{\|U\|.\|V\|}\right)$$

The algorithm makes use of the fact that for every facet, there are less than or equal to three neighboring facets. It thus gets the neighbors of a given facet, and then checks for the angle between surface normal of the remaining two facets. The neighbors of a given facet are obtained from the information contained in the edge list. The algorithm begins by assigning the first facet in the file to the first surface, and thus initiating the surface list. The neighbors for this facet are determined and added to the list of facets in the current surface if they have not been added and the angle between their normals is within the specified range. The neighbor which lies on the surface is then the seed for the

algorithm. If more than one facet is returned as being the neighbor of the current facet, then these are stored in a ‘check’ array that is updated after every search. A surface is deemed complete, when the ‘check’ array is empty and the last searches for a neighbor was NULL. The pseudo code for the procedure is as follows:

```
for(i = 0; i < no of facets; i++){
        Surface_list[surfaceno] = facet[i];
        surface_complete = false;
        start_facet = facet[i];
        while(surface_complete != true){
          neighbors = GetNeighbors(start_facet);
            if(neighbors != NULL){
              if(GetAngle(facet[neighbors[0]].normal, start_facet.normal) <= 37 &&
              facet[neighbors[0]].added_to_list == false){
                    Surface_list[surfaceno].facets[facetnumbers] = neighbors[0];
                    ToCheck[check] = neighbors[0];
                    facetnumbers++;
                    check++;}
        ‘ Repeat the procedure for all the neighbors, at most 3, which have been returned
                else{
                    if(check == 0){
                          surface_complete = true;
                          facetnumbers = 0;}
                    else{
                          start_facet = ToCheck[check-1];
                          check--;}
                }
              else{
                if(check == 0){
                    surface_complete = true;
                    facetnumbers = 0;}
```

```
                else{
                        start_facet = ToCheck[check-1];
                        check--;}
            }
        }
    }
}
```

The function *GetNeighbors* determines the neighbors of a given facet by looking up the edge list. During every iteration, it traverses the entire edge list until a match is found to look for facets other than the *start_facet*, which have the same edges as *start_facet*. The list of surfaces in the part is thus obtained in the array *Surface_list*. In addition, each element in the array also contains the list of facets that make up the surface.

5.2.4 Identification of edges on surface boundaries

Edges forming the boundary of a surface are the most important features in the CAD models for the current application. These are the edges that will be visible in the image resulting from the application of edge detectors on the images of part blanks as they are placed on the CNC router. These edges are identified with the addition of information about surfaces in the data structure to store the edge list. From the surface list, it is possible to determine the surface that a given facet belongs to. This information about the association of a given facet with a particular surface is used identify the surfaces which border a given edge. If these two surfaces are different, then the edge under consideration lies along the boundary separating two surfaces. This approach however will not work for spherical or cylindrical surfaces to delineate surface boundaries. As a result, the template generation approach too will fail for such parts. However, since virtually all part blanks machined with a CNC router are flat, the proposed approach is deemed acceptable. The pseudo code for the approach is:

Step 1: Obtain the information about the surface that a given facet belongs to:

```
for(i = 0; i < no_of_surfaces; i++){
        facets = PartData.SurfaceList[i].facetcount;
        for(int j = 0; j < facets; j++){
                facetno = PartData.SurfaceList[i].index[j];
                PartData.FacetList[facetno].surface_no = i;
        }
}
```

The variable *facets* stores information about the number of facets in a given surface.

Step 2: Identify the association between a given facet and surfaces in the parts:

```
for(i = 0; i < no_of_edges; i++){
        facetno = PartData.EdgeList[i].facetno1;
        facet_no = PartData.EdgeList[i].facetno2;
        PartData.EdgeList[i].surface1 = PartData.FacetList[facetno].surface_no;
        PartData.EdgeList[i].surface2 = PartData.FacetList[facet_no].surface_no;
}
```

Step 3: Generate the list of edges on the surfaces:

```
for(i = 0; i < no_of_edges; i++){
        if(EdgeList[i].surface1 != EdgeList[i].surface2){
                edges_on_boundary[boundary] = i;
```

boundary++;}

}

The required edges are thus obtained and stored in the array *edges_on_boundary*.

5.2.5 Rendering the solid model for template generation

After relevant data about the solid model has been extracted from the STL file, it has to be rendered with a graphics API to enable template generation. The API that was used in this implementation was OpenGL. The solid model with surface boundaries is rendered and then scan-converted for generating the template image. A brief description of scan conversion is explained in a later section, while the mechanisms for rendering the model are explained here. The image that is used as a template must closely resemble the image obtained after application of the edge extraction algorithms on the captured image. This requires simulating the workings of a camera in a situation analogous to the actual setup used for capturing images. Thus, the orientation and location of the part with respect to the camera needs to be taken into account. This is due to the fact that length and visibility of an edge in the image will change depending on the location and orientation of the part. The OpenGL API has a function call which enables the generation of user specified view dependent renderings. Moreover, depth culling, the removal of pixels occluded by others in the rendered image, is also performed by the graphics API. The working of a camera can thus be simulated using the OpenGL graphics library. Depth culling in OpenGL is performed with the Z-buffer algorithm, and is explained in the following sub-section.

5.2.5.1 The Z-buffer algorithm

OpenGL uses the Z-buffer algorithm [Foley, 1991] for visible surface determination. This algorithm identifies pixels that are occluded by others during rendering. The occluded pixels are not rendered during scene generation, and thus the algorithm saves time. The algorithm requires a frame buffer, F, that stores color values, and a z-buffer Z, having the same number of entries, which stores the z-value of each pixel in the buffer. The z-buffer is initialized to zero, representing the z-value at the point furthest from the camera location. The frame buffer is initialized to the background color. The largest value that

can be stored in the z-buffer represents the z of the point nearest to the camera. Polygons in the scene are then scan converted into the frame buffer in any arbitrary order. During scan-conversion, if the polygon point being scan converted at, say, *(x,y)* is nearer to the camera location than the points whose color and location are in the buffers, then the new point replaces this point. The pseudo code for this algorithm is given below.

```
for(i = 0; i < ymax; i++){
        for(j = 0; j < xmax; j++){
                WritePixel(j, i, BACKGROUND_VALUE);
                WriteZ(j, i, 0);}
        for(i = 0; i < no_of_polygons; i++){
                NPixels = Pixels_in_polygon(i);
                for(j = 0; j < NPixels; j++){
                        pz = Polygons_Z(j, i);
                        if(pz >= ReadZ(j, i)){
                                WriteZ(j, i, pz);
                                WritePixel(j, i, ColorAt(j, i));}
                }
        }
}
```

In the algorithm, *ReadPixel* and *WritePixel* perform the tasks of reading and writing to the frame buffer respectively. Similarly, *ReadZ* and *WriteZ* read and write to the z-buffer.

Computing the Z height of a given polygon is critical, and the task is performed by the *Polygons_Z* function. In this computation, the equation of a plane, $Ax + By + Cz + D = 0$ is solved for the value of z. The value of z is obtained as:

$$z = \frac{-D - Ax - By}{C}$$

Now, if the value of z at a point (x,y) is known as, say z_1, it can be easily interpolated for the pixel at $(x + \Delta x, y)$. The z value is computed as:

$$z_n = z_1 - \frac{A}{C}(\Delta x)$$

However, the quotient A/C is constant and Δx usually has a typical value of 1. Thus the scan conversion can be performed very quickly with just a simple subtraction. The first value of z on the next scan line for changes in y can similarly be calculated with a decrement of B/C, where Δy is 1. Moreover, if the polygon is not planar, $z(x,y)$ can be determined by interpolating the z coordinate of the polygon's vertices along the pairs of edges, and then across each scan line. The methodology is illustrated graphically in Figure 5.4.

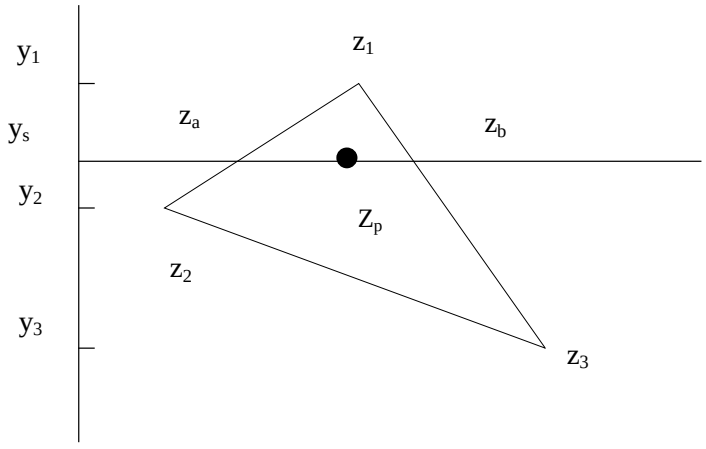


Figure 5.4 Interpolation of z values along polygon edges and scan lines.

(Source: Foley, 1991)

The values for z_a, z_b and z_c are computed as:

$$z_a = z_1 - (z_1 - z_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$z_b = z_1 - (z_1 - z_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$\text{and} \quad z_p = z_b - (z_b - z_a)\frac{x_b - x_p}{x_b - x_a}$$

The z-buffer for a scene is thus created. The geometric entities being rendered need not be polygons. Any object can be rendered provided a value for the color and z-value at a given pixel can be determined.

Depth culling, or hidden surface removal, is thus possible using the OpenGL API. Moreover, since the location of the camera can be controlled, renderings can be made to simulate view dependency. The other issue that needs consideration is the optics of the camera. A basic camera model has to be duplicated to model the camera lens' focal length and the resultant magnification factor. To enable this, few images of parts with known dimension were captured. The lengths of various edges in the captured image were measured in term of pixels. This gave a certain scaling factor for the mapping from real world to image space coordinates. Using this approach with the experimental setup, it was determined that 1 inch on the bed of the machine corresponded to 2.55 pixels. Generation of the template from the rendered image of the solid model gives a one to one mapping. Hence, the rendered image itself has to be modified to take this scaling factor into account. The camera in the experimental setup is affixed at a distance of 50 inches from the bed where parts are placed. Since the scaling factor is 2.55 for this setup, the simulation model uses a z value of 19.61 (50 inches/2.55 pixels) as the coordinate for locating the camera. The location of the camera is controlled with the OpenGL API call *gluLookAt*. The inputs for this function are the coordinates for the location of the camera, coordinates for the center of the scene, and a vector pointed in a direction perpendicular

to the direction of vision. The system was thus modeled to simulate the working of the camera used in the experimental setup.

During rendering, separate colors are assigned to the solid model and to the edge boundaries. This is necessary in order to differentiate between the pixels during scan conversion. The OpenGL API uses a scale of 0 to 1 for each component of light, viz., red, green and blue (RGB). A value of zero maps to zero intensity, while 1 maps to maximum intensity. Before rendering the solid model and the edges, depth culling is set on. Hidden surfaces are thus removed automatically by the API. For rendering the solid model, the values used are 0.6, 0 and 0 for the red, green and blue components respectively. The edges are rendered using values of 0, 0.8 and 0 for the red, green and blue components respectively. Depth culling is then set off, and the center (considering geometric extents) of the object is rendered in another different color. The values of the red, green and blue components used for this are 0, 0 and 1.0 respectively. All of these values were selected randomly. However, since white light is used as the light source in the scene, the values chosen for the components tend to affect the scan conversion algorithm. This topic is covered in the next section dealing with scan conversion of the rendered scene. A part that is rendered by the API with the above mentioned color scheme is shown in Figure 5.5.

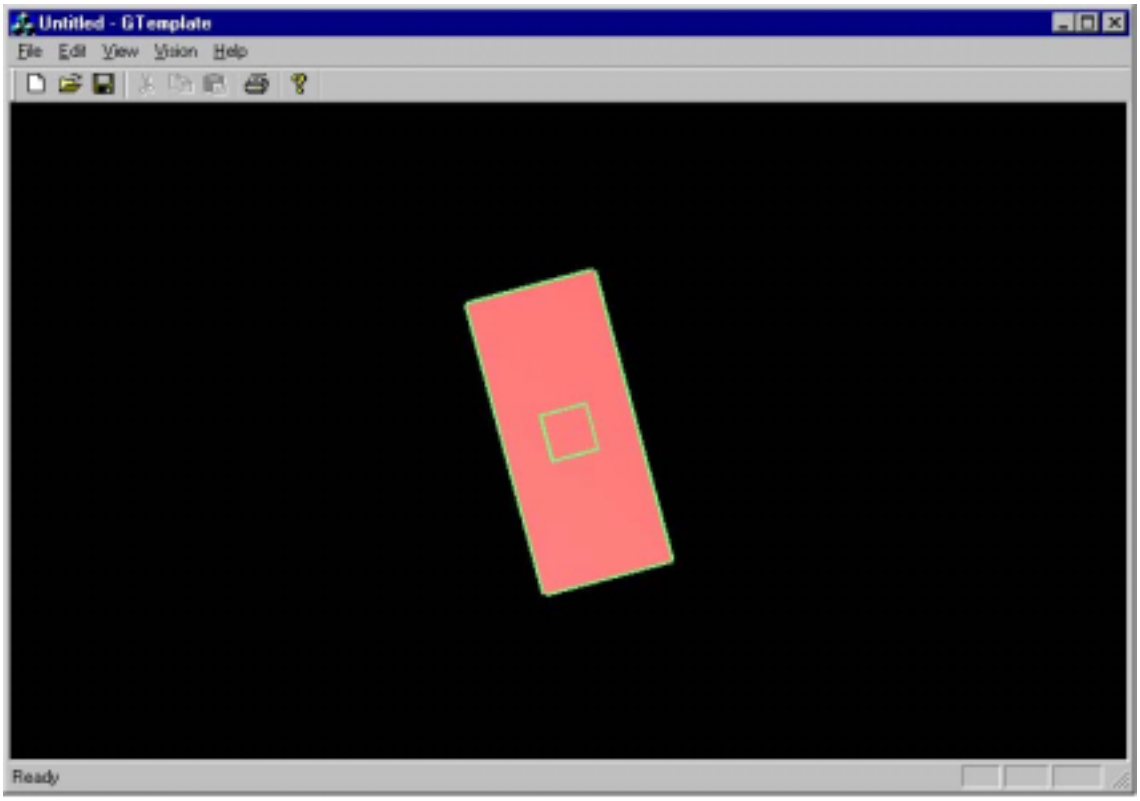


Figure 5.5 Rendering of a part for pose determination

5.2.6 Scan conversion of the rendering

The image for the template is obtained by scan converting the rendered image of the object and the boundary edges. The scan conversion is performed from left to right and top to bottom. RGB values for each pixel in the image is obtained using the OpenGL function call *glReadPixels*. This function accepts the coordinates for the bounding box of the viewport. The returned data is the information about the color of each pixel in the image contained in the specified viewport. The data is returned as a 3-byte number, with each byte being used for the red, green and blue components respectively. Since the template to be generated should look like the image after edge detection, only the edges in the rendered image need to be saved in the scan-converted image. The edges are drawn in the green color; hence it is possible to identify these pixels. This is done with the use of a threshold for the green component of the scanned image. Since white light is used for illuminating the scene, each pixel with a non-zero color value has components of red, green and blue. After scan-conversion, pixels with highest intensity for the green

component have a value of 255. However, the edges are drawn with a value of 0.8 for the green component and hence a value of 0.8*255 i.e. 204 is chosen as the threshold for the scan conversion. Thus, each pixel in the rendered image with a value greater than or equal to 204 for the green component is stored in the template image. The template generated for the rendering in Figure 5.5 after scan conversion is shown in Figure 5.6.

Determination of part location on the bed of the router is another issue that is of importance for the current implementation. Coordinates of the center are computed as a means of exactly locating the part. Depth culling is set off, and then the center is rendered in blue. The value of the intensity for blue is 1 while intensities for the other components are set at 0. After scan conversion, the pixel having a blue component value of 255 in the array of rendered pixels is the location of the center. The coordinates of the center in the template image are used to assist in the determination of part location on the bed of the machine. Thus the template is prepared and the location of the part determined with scan-conversion.

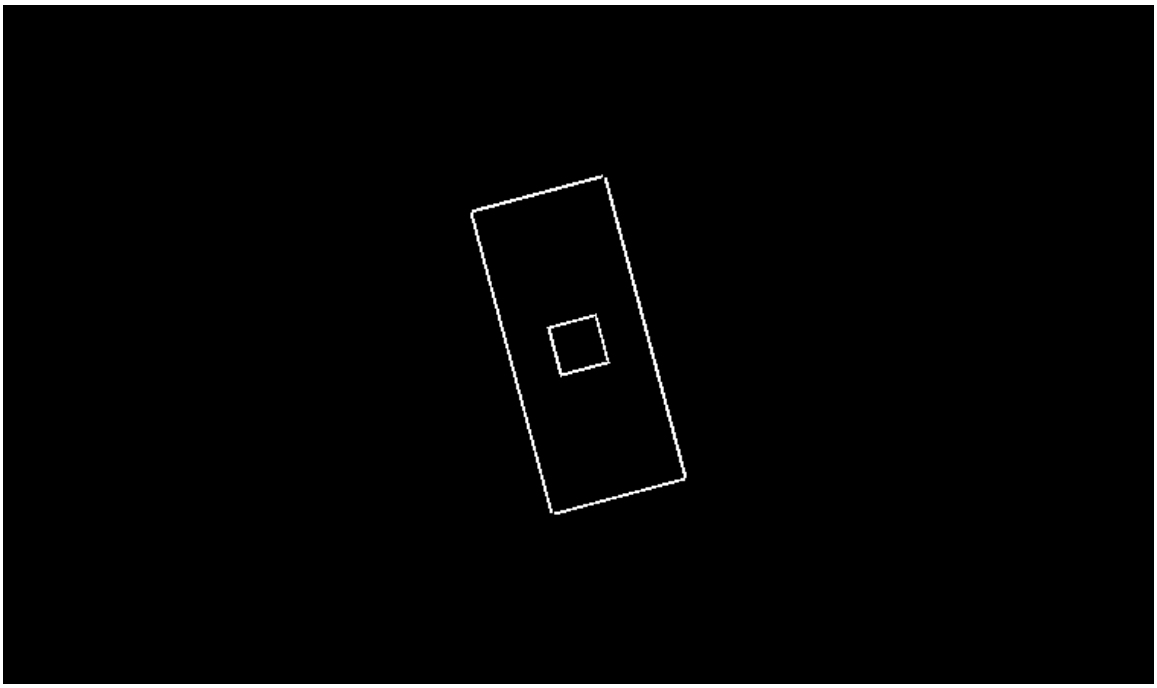


Figure 5.6 Image generated by scan converting the rendering in Figure 5.5

5.2.7 The Template image

After the rendered solid model is scan-converted, the template image has to be created for the matching process. During the matching process used in this implementation, the algorithm checks the template on a one-to-one basis with the actual image. The two images must therefore match as closely as possible. This requires that the image sizes be equal in order to get a metric that accurately reflects an actual match between the images. The scan-converted image is therefore cropped so that its size is equal to the region of interest specified in the captured image. In the scan-converted image, edge points are written with a brightness of 255. The algorithm to crop the image first computes the bounding box for these pixels. The centroid of this box is then computed, and the required area to be written to the cropped image is obtained by moving a distance equal to height/2 and width/2 along the ± X and ± Y directions. This approach works only if size of the viewport is greater than the specified region of interest. The pseudo code for this procedure is given below:

```
MaxPoint = GetMaxXY(Input_image);

MinPoint = GetMinXY(Input_image);

CenterPoint = (MaxPoint + MinPoint)/2;

Tx = CenterPoint.x – (RequiredWidth/2);

Ty = CenterPoint.y – (RequiredHeight/2);

Tx1 = CenterPoint.x + (RequiredWidth/2);

Ty1 = CenterPoint.y + (RequiredHeight/2);

for(i = Ty; i < Ty1; i++){

        for(j = Tx; j < Tx1; j++){

                if(Input_image(j, i) == 255){
```

Cropped_image(j, i) = 255;}

}

}

The cropped image is thus obtained with the aforementioned algorithm. The cropped image generated from the scan-converted image in Figure 5.6 is shown in Figure 5.7.

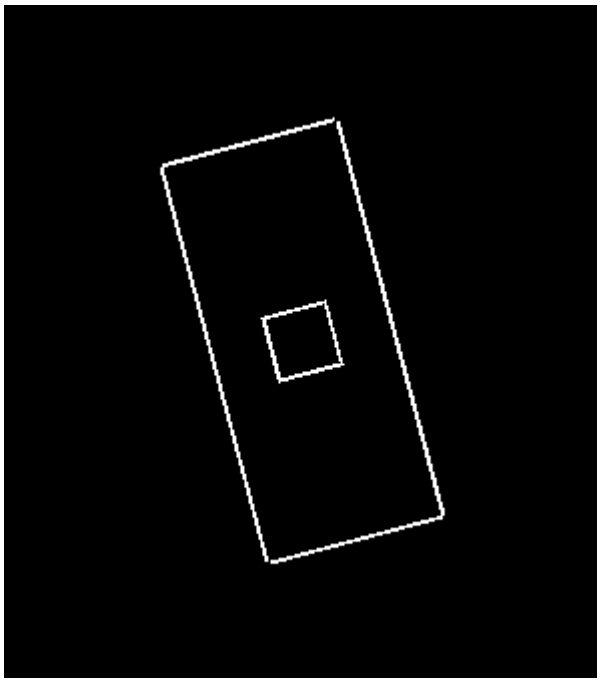


Figure 5.7 Template generated after cropping image in Figure 5.6

5.3 Template Matching

Template matching is the step that determines the level of similarity between the captured image (referred to as input image) and the generated template. As mentioned earlier, template matching is a robust technique that works admirably with noisy images. It handles variations in a class due to additive noise quite well. In the current scenario, wood grains can be considered as additive noise, as their frequency in the input image is low in comparison with the edge points. Different metrics are used to compute the degree of similarity between images that are being compared. The metric used in this implementation is a variation of the maximum correlation approach. In the maximum

correlation approach, the number of agreements with the black and white pixels in the two binary images is computed. In other words, for a given pixel in the input image, the corresponding pixel in the template image is checked. If the brightness levels match, then the counter is incremented by 1. A variant of this approach was used for the current scenario. It was hypothesized that counting the number of matches of black spots would be misleading on account of the wood grains that are present in the input image after application of the edge extractors.

The number of agreements between the input image and the template image for bright pixels in the input image is first computed. The metric that was used in this approach computes the percentage match between the template and the input image. Thus, a ratio of the number of matches between the template and the input image, and the number of bright pixels in the template, is computed for the purpose. The template generated in this method does not have any noise, especially wood grains, hence the match obtained with this metric reflects a better association between the input and the template image. After the percentage match is obtained, it is subjected to a threshold function in order to determine if the degree of match is above a certain minimum value. The value used for the threshold was 20 percent. This number was obtained from experimentation with a variety of sample parts, and it proved to work well under differing conditions. The pseudo code for this approach is as given below:

```
Npixles = BrightPixels(Template_image);

For(i = 0; i < height; i++){

    For(j = 0; j < width; j++){

        If(Input_image(j , i) == 255){

            If(Template_image(j, i) == 255){

                Match++;}

        }
```

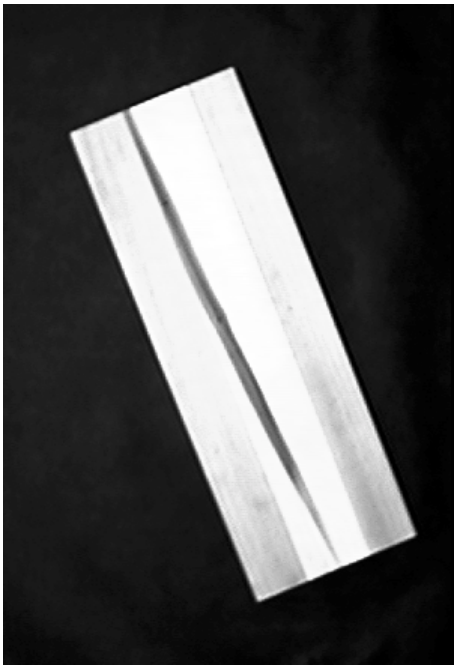
}

}

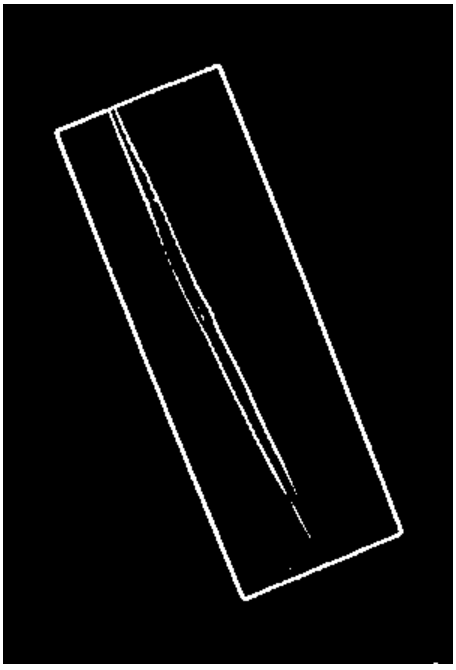
Degree_of_Match = Match/Npixels;

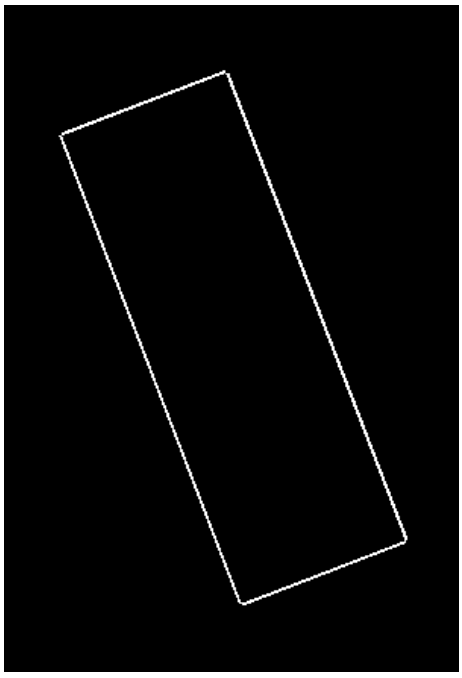
The function *BrightPixels* used in the algorithm determines the number of pixels in the image that have the maximum brightness level.

The parts that are placed on the bed of the router are oriented in a fashion as specified by the nesting software. The parts are hence assumed to be off from the intended orientation by no more than ± 5 degrees. The algorithm to determine the pose thus looks through a finite space of possible orientations and computes the match for each orientation. An absence of symmetry in the part is also accounted for in the algorithm. The solid model of the part is thus oriented at 0 degrees along the X and Y-axes and at the specified angle for the Z-axis. The orientation is then changed by 5 with an increment if +1 degree in each step. The model is re-oriented at the specified angle and the orientation changed by 5 degrees with a decrement of 1 degree in each step. The object is then rotated by 180 degrees with respect to the Z-axis. The orientations are again changed by 5 degrees above and below the mean orientation, as specified by the nester. This procedure is repeated for the following sets of orientations: (180, 0); (180, 180) and (0, 180), where the representation used is (X orientation, Y orientation). The procedure for checking within a ±5 degree orientation from the intended orientation with respect to the Z-axis is repeated for all these sets. The steps illustrated above account for the absence of symmetry along the three axes. The template is generated at each step of this procedure, and is compared with the input image. The match metrics are computed for each of these setups, and the orientation that results in the maximum match is chosen as the part orientation. Application of the methodology for a drawer-front part blank is shown in the following figures.



(a)





(b) (c)

Figure 5.8 (a) Part blank for a drawer-front, (b) Input image for template matching and (c) Selected template, which gives the best match

The Figure 5.8 (a)-(c) show the template matching procedure being used for a part blank that is used for making a drawer-front. This part needs to be machined with multiple setups, and hence is an ideal candidate for increasing production efficiency with a universal fixture. The orientation of the part has been identified at 21 degrees with respect to the Z-axis. The coordinate system used in the system has the Z-axis directed into the plane of the image. The orientations along the X and Y axes are not really important, as the object is symmetric about the X and Y-axes for the setup shown. The algorithm computed a match of 67.92 percent for the scenario. The efficacy of the system can be illustrated by the image in Figure 5.9 that shows the template image overlapping the input image. Pixels for the template have been drawn with an intensity of 128, as against 255 used for the input image pixels.

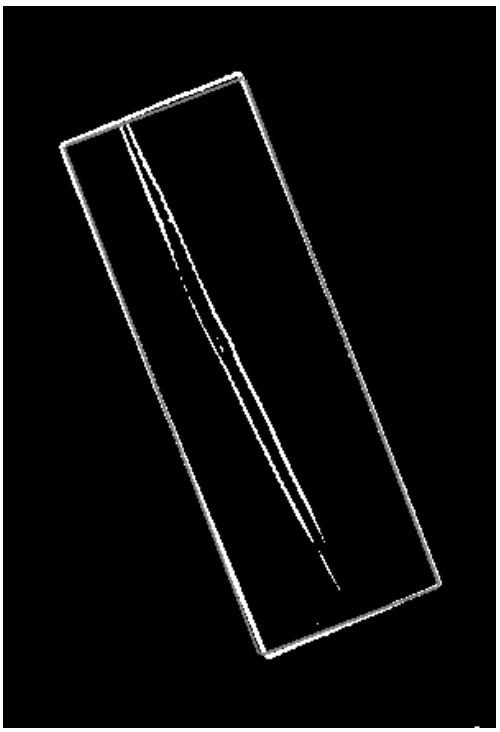
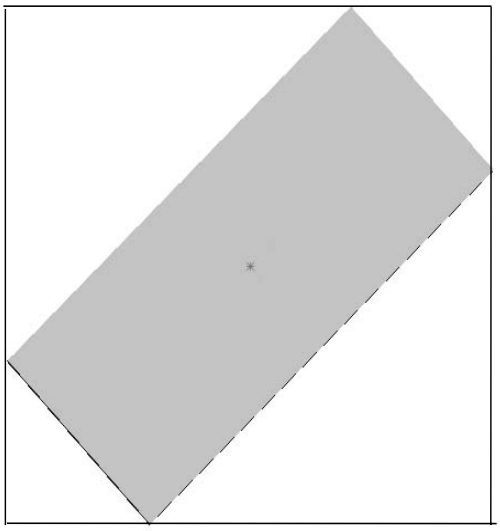


Figure 5.9 Image illustrating the match between the template and input image

After the object's orientation has been determined, coordinates of its body center need to be computed in order to locate the part accurately. The nesting software decides part location and orientation in order to maximize usage of space on the router bed. Extents and location of the bounding box for a part are know from the nester. The user is hence prompted to input the location of the lower left hand and upper right hand corners of the bounding box. The region of interest for a given image usually subsumes the corresponding part's bounding box. The region of interest in this application usually is the image that serves as an input to the algorithms. Figure 5.10 (a) and Figure 5.10 (b) should help to better illustrate the concept.

(a)

Upper right corner



Lower left corner

(b)

Figure 5.10 (a) The region interest for a part blank; and (b) The bounding box for a part

In Figure 5.10 (a), the region of interest is the entire image. The bounding box for the solid model of the part is shown in Figure 5.10 (b). Since the location of the bounding box in the router bed space is know, the region of interest too can be mapped backed onto the physical world space from the image space. From the STL file, coordinates of the part centroid are computed after determining the extents of the part along the X, Y and Z-axes. This point is drawn while rendering the solid model, and scan-converted along with edges in the object. Using this procedure, it is possible to locate the body center in the image space. Since the centroid can be located with respect to the region of interest as well, its location in the router bed space is also determinable. The location and orientation of the part in the physical space is thus determined.

5.4 Updating parametric part programs

The part blanks placed on the CNC router are machined with part programs. The ability to use parametric part programs with these machines enables updating the part programs once the actual orientation and location of the part is determined. This section gives a brief introduction of parametric part programming used with CNC machines and its use in this context to account for inaccuracies in part placement and orientation. Parametric programming is a feature that is present in CNC machines and is conceptually quite similar to sub-programming, which is quite popular with most CNC programmers. The features of parametric programming resemble features of computer programming languages and give the CNC control an ability to compensate for minor translation and rotation of parts on the bed of the machine. It enables CNC controllers to use variables, perform various basic arithmetic operations, provides for a decision making capability, and the ability to perform looping. These are all features akin to computer-programming languages, and are the features that make parametric part programming a powerful tool. A simple parametric program [Lynch, -] is given below to illustrate some of the features of parametric programming.

ABC (Program number)

#100 = 0 (Left side position in X)

#101 = 0 (Lower side position in Y)

#102 = 0 (Top surface of workpiece in Z)

#103 = 5.0 (Length of workpiece in X)

#104 = 3.0 (Width of the workpiece in Y)

#105 = 0.5 (Workpiece thickness)

#106 = 1.0 (Milling cutter diameter)

#107 = 400 (rpm for milling cutter)

#108 = 5.0 (Feedrate of milling cutter in inch/min)

N010 G54 G90 S#107 M03 T02 (Select the coordinate system, set absolute mode, turn spindle on and get tool ready)

N015 G00 X[#100 + #103 + #106/2] Y[#101 – 0.1 - #106/2] (Rapid move to the workpiece)

N020 G43 H01 Z[#102 + 0.1] (Provide tool-length compensation, move closer to work surface)

N025 G01 Z[#102 - #105 – 0.1] F50.0 (Fast feed past surface)

N030 Y[#101 + #104 + 0.1 + #106/2] F #108 (Mill right side of workpiece)

N035 G00 Z[#102 + 0.1] (Rapid move to top of surface)

N040 G91 G 28 Z0 M19 (Move to tool-change position)

N045 T02 M06 (Place drill two in the spindle)

N050 G54 G 90 S800 M03 T01 (Select coordinate system, absolute mode, start spindle, get tool 1 ready)

N055 G00 X[#100 + 0.5] Y [#101 + 0.5] (Rapid move to first hole position)

N060 G81 R[#102 + 0.1] Z [#102 - #105 – 0.18] F4.0 (Drill first hole)

N065 Y[#101 + #104 – 0.5] (Drill second hole)

N070 G80 (Cancel cycle)

N075 G91 G28 Z0 M19 (Return to tool-change position, orient spindle)

N080 M30 (End program)

In the program above, the parameters have been defined with the *#XYZ* notation. Thus, if the coordinates or numbers defining the parameters are changed, the program can be changed to account for inaccuracies or to machine a different but similar part. Thus, the parametric programming approach can lend itself suitably for the current application where the inaccuracies are accounted for by changing the part programs. Once the location of the center and rotation of the part on the bed space has been determined,

updating the coordinates used in the part programs is quite simple. Thus, the vision system can be used to reduce part rejection and provides the means to enable tighter tolerances for the parts being machined on the router. This can help in reducing the cost of fabrication for parts and lead to higher profitability for the manufacturer.

6. Implementation and Results

6.1 Implementation

The approach described in Chapter 5 was implemented using Visual C++ on the Windows NT platform. The OpenGL library was used directly with Visual C++. To handle bitmap files, a commercially available package called Vision Foundry (Data Translation) was used. Vision Foundry has an object oriented API, and the objects can be used to perform various manipulations with bitmap files. Of importance for the present implementation was the ability to read and write information to and from bitmap files using these API functions. The images used in the implementation had 8-bits of gray scale, resulting in 256 levels of gray. The CAD models used were stereolithography (STL) files, and the modeling was done using the SolidWorks CAD system. Since the system is used to determine the pose of an object placed on the bed of a CNC router, an experimental test station was constructed using Creeform steel tubing. A camera and a frame grabber were integrated with Data Translation's Acquire software to write the captured data to bitmap files. The application development was done using Visual C++, and it has a graphical user interface (GUI) to guide the user through each step. The user follows the steps below to determine the orientation and location of an object placed on the bed of the router:

- Select the appropriate bitmap (.bmp) file containing the image of the part whose orientation and location is to be determined.
- Select the CAD model (.stl) of the object.
- Input the values for the bounding box of the part in router bed coordinates, and the expected orientation of the part with respect to the Z-axis.

Commercial nesting software commonly used in industry computes the orientation of each part with respect to the Z-axis in order to maximize the usage of the CNC router bed. Depending upon the parts that are being machined in the given setup, the orientation of the parts can be varied. The bounding box of the part is obtainable from the nesting

software, and this information can easily be fed into the current application in lieu of asking the user to manually enter the data. The following figures show the various dialogue boxes where the user has to make suitable selections or provide data.

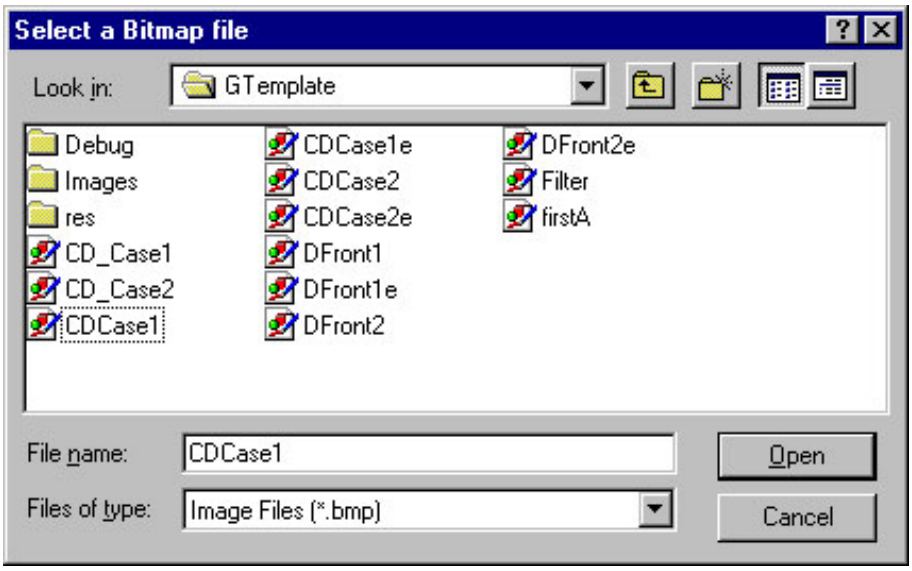


Figure 6.1 Dialogue box to select the bitmap image of a part

The first dialogue box prompts the user to select the bitmap file of the part whose orientation and location in the image are to be determined. This is followed by the selection of the relevant solid model of the part. Figure 6.2 shows the dialogue box to enable this selection.

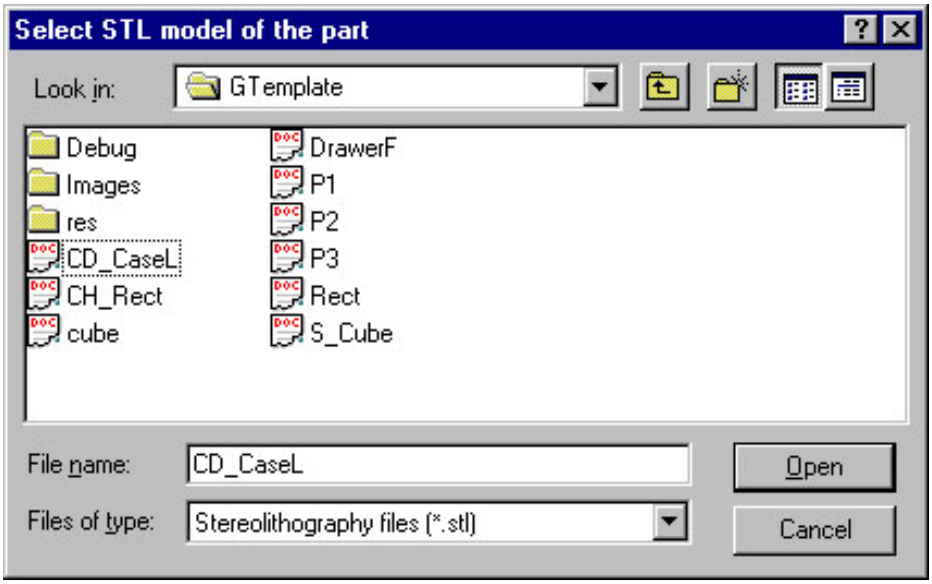


Figure 6.2 Dialogue box to select the solid model of the part

The second dialogue box involves selection of the CAD model to describe the part. If the user does not select the correct solid model, the degree of similarity between the image and the geometry from the model is quite low. The software applies a threshold to the similarity metric computed with the template matching procedure. If the degree of match is less than the prescribed threshold, then the match is considered unsuitable, and the user is notified about the incorrect selection. However, before the template matching is performed, the user has to input the expected orientation of the part with respect to the Z-axis, and the coordinates of the bounding box for the part in the router bed space. This information would normally be read in automatically from the nesting software. The match between the template and the image can be below the threshold if the expected orientation is incorrect. The efficacy of the procedure to check for inconsistencies in the input model or the initial orientation depends largely on the geometry of the actual part in the image and the selected solid model. If the two geomteries are very similar, the algorithm is likely to give erroneous results. Figure 6.3 shows the dialogue box to input the bounding box and the initial orientation.

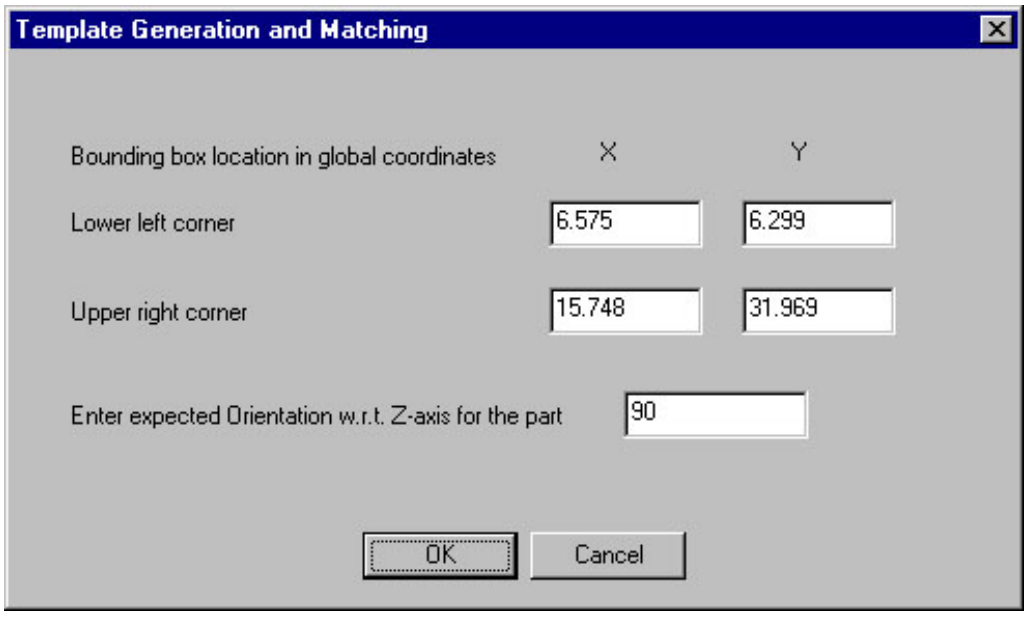


Figure 6.3 Dialogue box to input the bounding box and expected orientation of the part

After the appropriate selections have been made and relevant data input to the algorithm, the next step is the selection of the menu option to start the search for the correct pose of the object. This is done with the selection shown in Figure 6.4

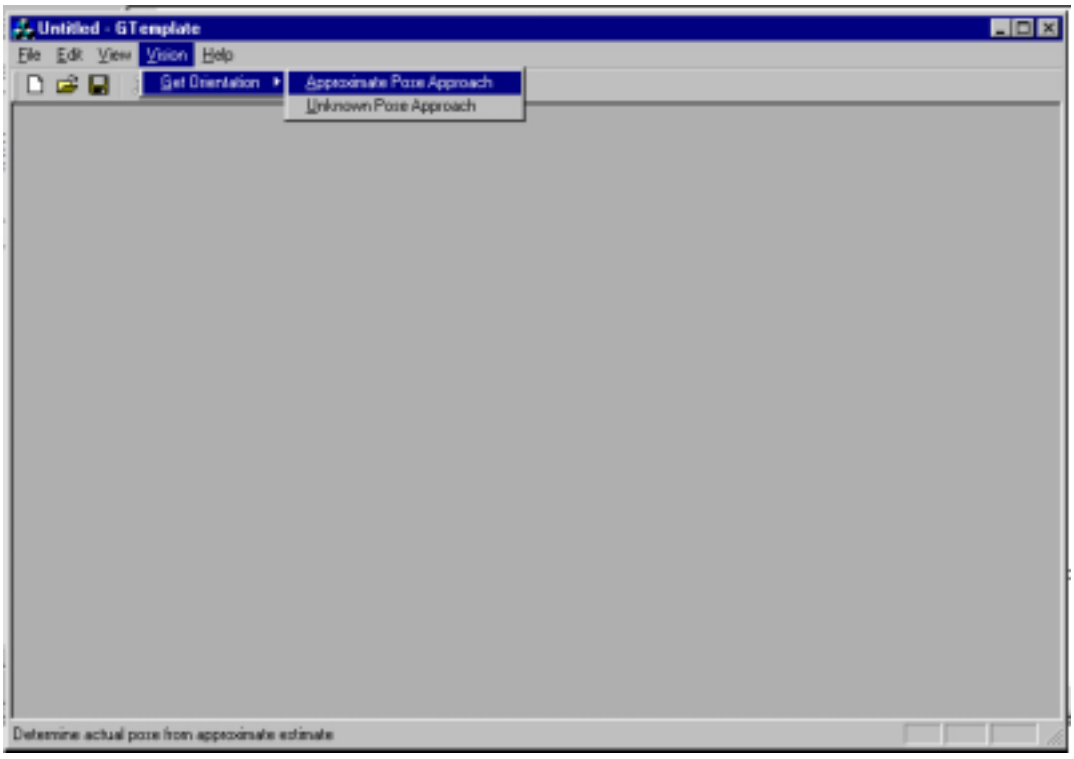


Figure 6.4 Menu selection to search for the object's pose

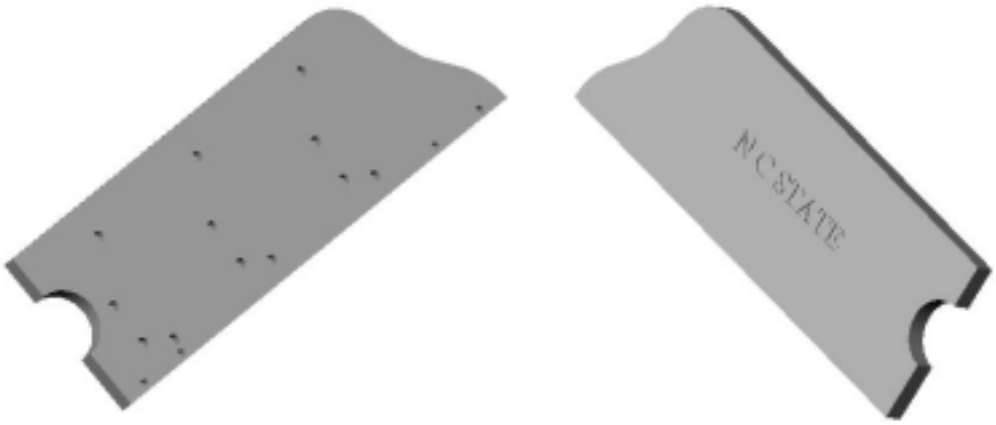
This selection enables searching the space within a ±5 degree range from the expected orientation with respect to the Z-axis. The algorithm also searches by flipping the part about the X and Y-axes to account for absence of symmetry along the respective axes. Thus, four iterations are performed with the initial orientations being (0, 0, *z*), (180, 0, *z*), (180, 180, *z*) and (0, 180, *z*), where *z* is the expected orientation about the Z-axis. Each iteration involves searching within the ±5 degree range about *z*. The maximum match between the template and the image that is returned by the software is selected as the likely pose for the object. The location of the part's center is then determined by scan converting the image with the center rendered in the blue color. From the location of the bounding box on the router bed space, it is possible to determine the location of the center in the physical coordinate space. The orientation of the object and the actual

location of the part are written to a text file ‘Pose.dat’. This approach was applied to some of the parts machined with a CNC router at the Furniture Manufacturing Center (FMC) at NC State University. The results from the tests are included in the next section.

6.2 Results

The methodology being developed is primarily intended to improve the productivity of furniture manufacturing by reducing setup times on the CNC router. The setup times for parts that need multiple setups can be reduced considerably with this approach. Hence, parts that need multiple setups were used to test the algorithm. To increase the difficulty of the testing procedure, instead of using simple rectangular part blanks, specimens requiring two machining setups had their first machining operation performed. The partially completed part blanks were then used to test the capabilities of the proposed approach. Accurate determination of the part orientation and location is vital for such pieces, as one side has already been machined, and the other side needs to be machined with reference to the features already created.

The side of a CD case was the first part that was used for testing the approach. Figure 6.5 (a) and (b) show the two sides for the part.



(a) (b)

Figure 6.5 Solid model of (a) Inner side of the CD case; and (b) The outer side

An image of the part before machining the 'NC State' text on the outside was captured and used as a test piece to determine the orientation and location of the part. Figure 6.6 shows the image used as input.

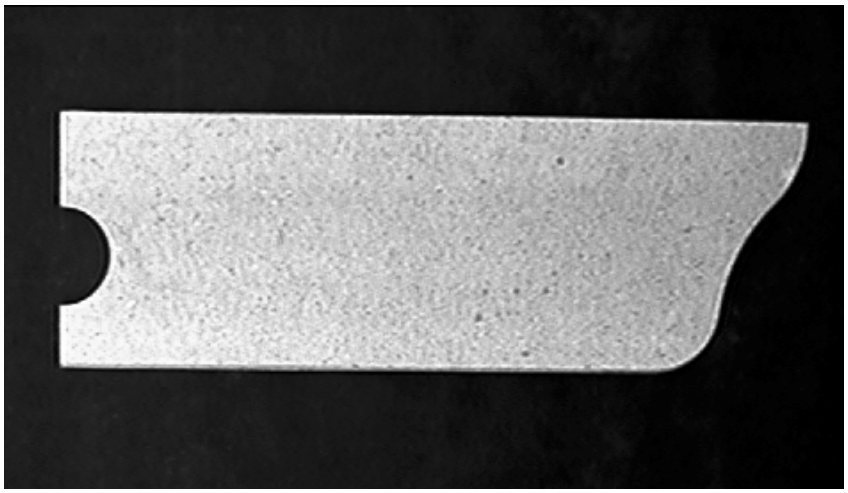


Figure 6.6 Image of the part blank before machining 'NC State'

The bounding box for the part on the table of the experimental setup was measured manually (in inches in this case) and input to the system for determining the part location. The approximate orientation for this case was 0°. This information in the actual system can be obtained directly from the nesting software. The inputs for the system are shown in Figure 6.7.

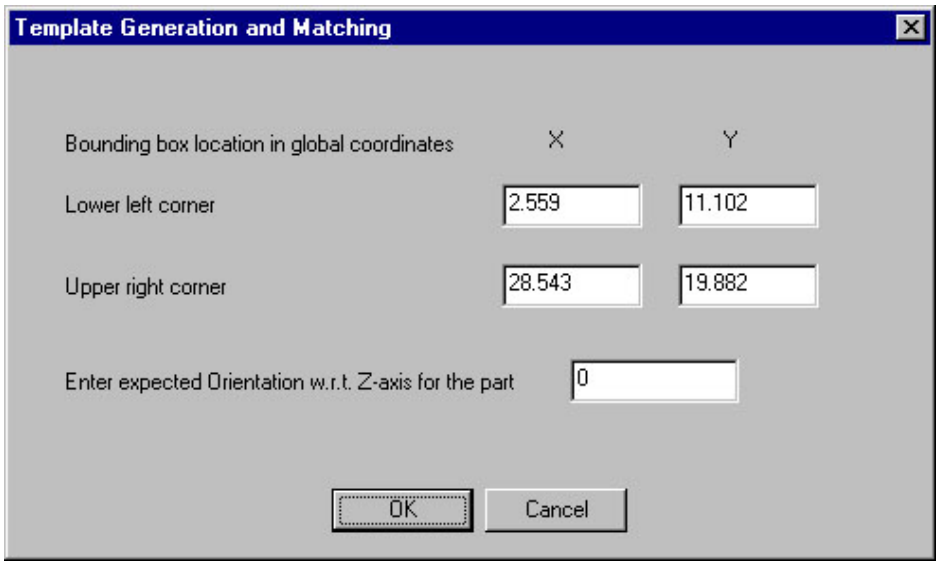
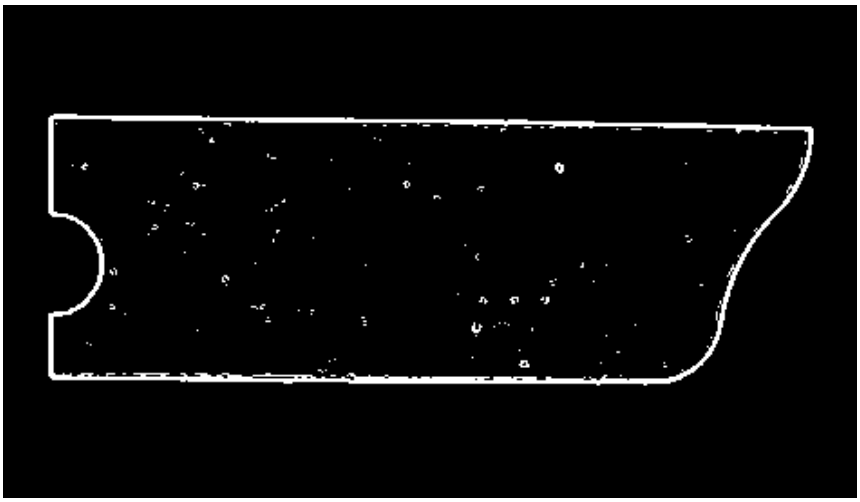


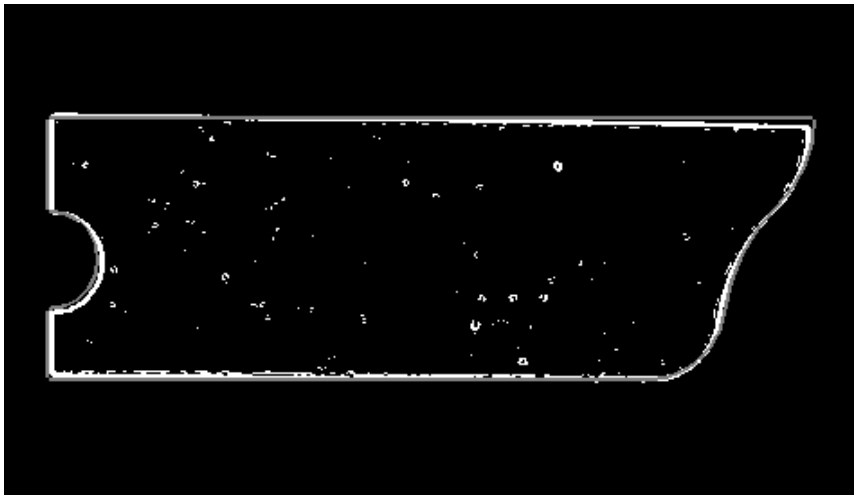
Figure 6.7 Inputs to determine the pose of image in Figure 6.6

The image generated after performing edge extraction, which is the image used to determine the part orientation and location is shown in Figure 6.8 (a). The orientation of the object is determined at 0° with respect to Z-axis, 0° with respect to the X-axis and 180° with respect to the Y-axis. These values account for the lack of symmetry along the Y axis in the solid model of the part. The image in Figure 6.8 (b) shows the template image that was selected, which gives the best match within ± 5° to the expected orientation.



(a)

(b)



(c)

Figure 6.8 (a) Image after performing edge extraction on image in Figure 6.6; (b) The selected template; and (c) Image showing the match between the template and input

The template image in this case gives a match of 49.33 percent. The speckles present in the image in Figure 6.8 (a) are caused by the texture of wood grains and result in the low number. The match however is accurate, and the appropriate angle is selected as the possible orientation of the object. The computed location of the center in this case is (15.551, 15.492). The value computed from manual measurement was (15.551, 15.492). Thus, there was no error in locating the center along the X or the Y axis. The measured orientation was exactly equal to the value determined by the algorithm. In order to machine the part, lets say four points as shown in the Figure 6.6 are of importance.

Figure 6.9 Points of interest for machining the part

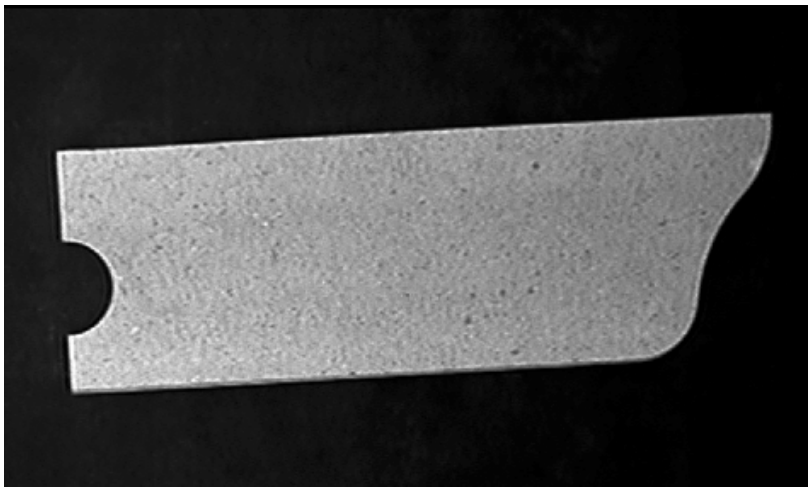
The locations of these points from manual measurement and those computed by the algorithm are given in the table below. The locations of the coordinates are the ones on the bed of the router. Since all of the coordinates considered have values of Z = 0, all of the measured coordinates are specified using only the X and Y values (in inches). The value for the Z coordinate for all the points is 0.

Point	1	2	3	4
Coordinates	(0, 0, 0)	(10.04, 0, 0)	(0, 1.18, 0)	(0, 2.51, 0)
Measured coordinates	(2.559, 11.102)	(28.543, 11.102)	(2.559, 14.370)	(2.559, 17.519)
Computed coordinates	(2.559, 11.102)	(28.543, 11.102)	(2.559, 14.156)	(2.559, 17.597)

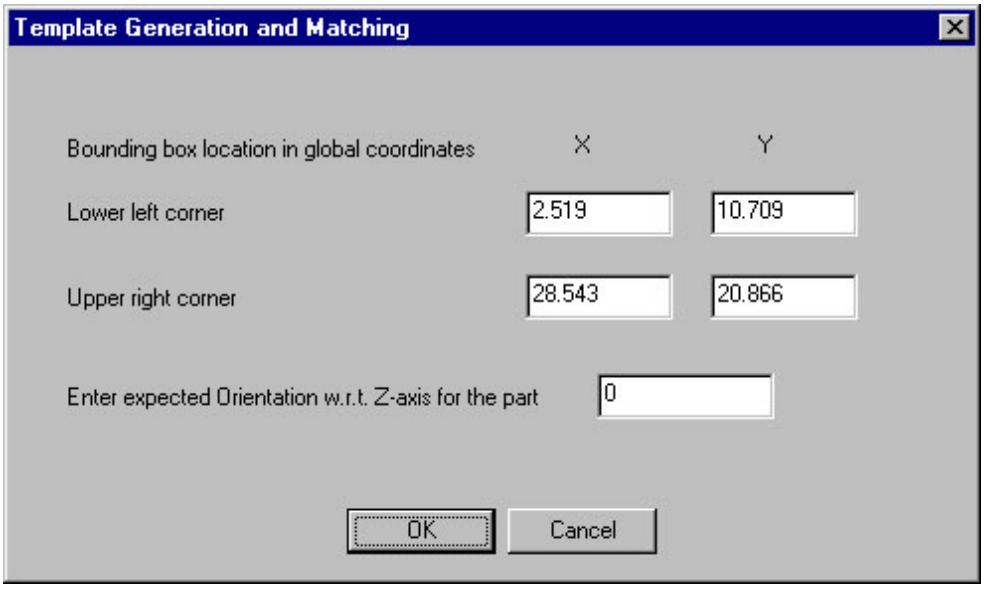
Table 6.1 Comparison between manually measured and algorithmically computed values for sample points

Using the data about the accurate location of the part, the parametric CNC part program can be updated to machine the part blank to the required level of accuracy. Moreover, the part blank need not be oversized, and the location of the feature ‘NC State’ on the outer side is likely to be machined to a greater degree of accuracy. The following example shows the same part that is ‘off’ from the specified orientation and location because of the inaccuracy in part placement. The use of the algorithm provides the actual pose of the

part. An image of the part in this incorrect location is shown in Figure 6.9 (a). The input parameters are shown in Figure 6.9 (b).



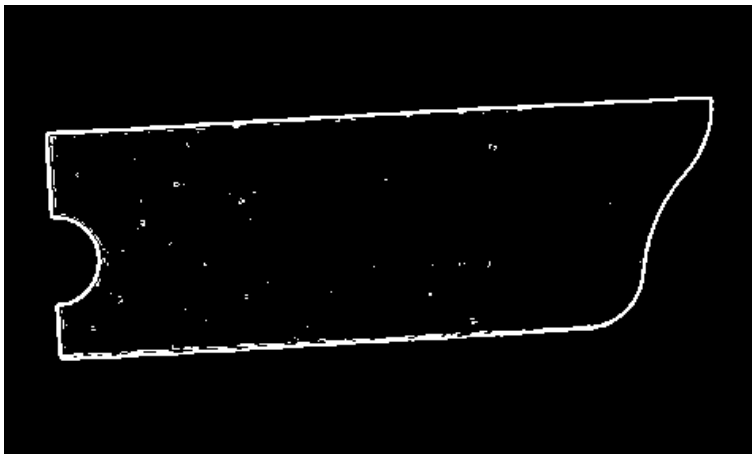
(a)



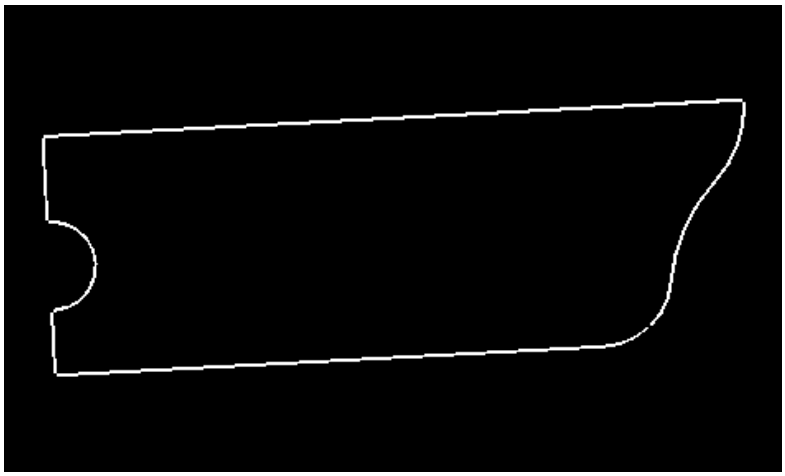
(b)

Figure 6.10 (a) Part image with incorrect orientation; and (b) Input parameters for pose determination

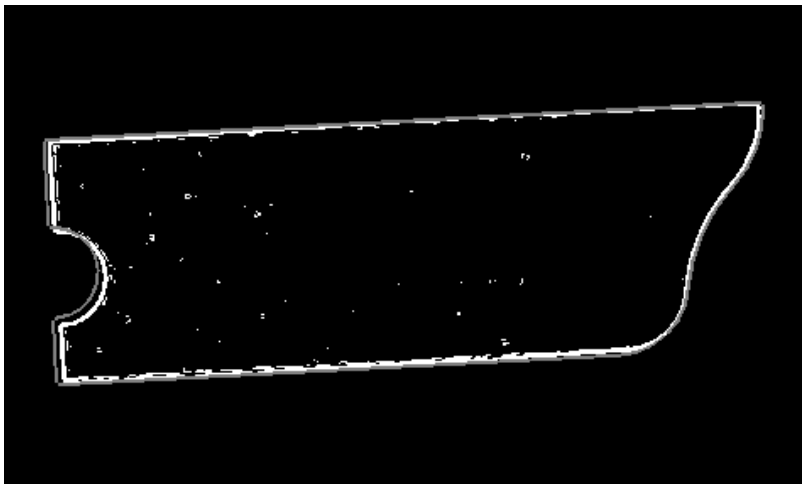
The following figures show the choice of the template and the match between the input and the template image.



(a)



(b)

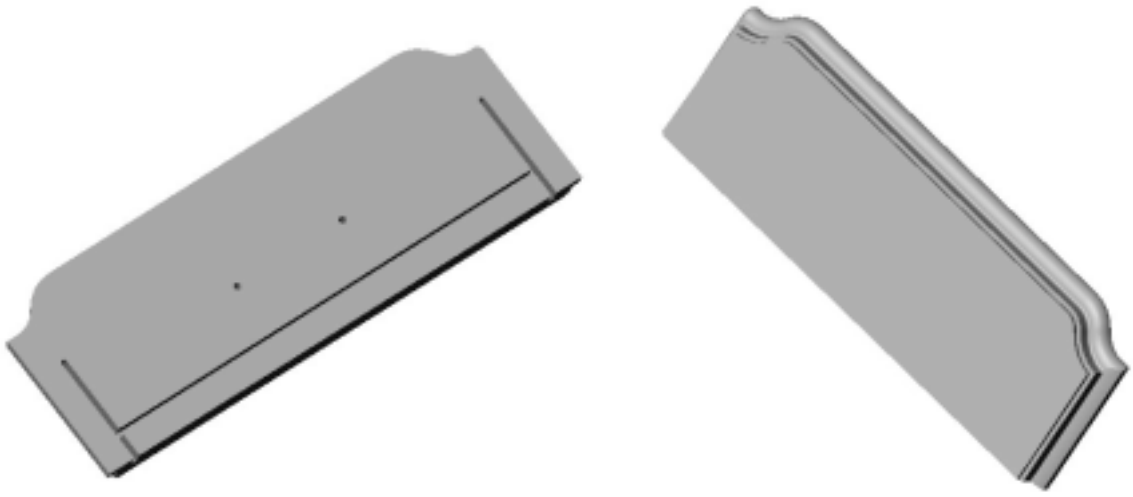


(c)

Figure 6.11 (a) Image after performing edge extraction; (b) The selected template image; and (c) Image illustrating the match between the images.

The orientation of the object in this case is computed as 3° with respect to the Z-axis, and 0 and 180° respectively with the X and Y-axes respectively. This matched exactly with the manually measured value. The center is determined to be at (15.531, 15.787). Location of the center from manual measurement is (15.531, 15.787). This value can now be used to update the location of the coordinates of interest to update the parametric part program as explained in the earlier chapter. The inconsistency in the location and orientation of the part can thus be taken into account before machining the part.

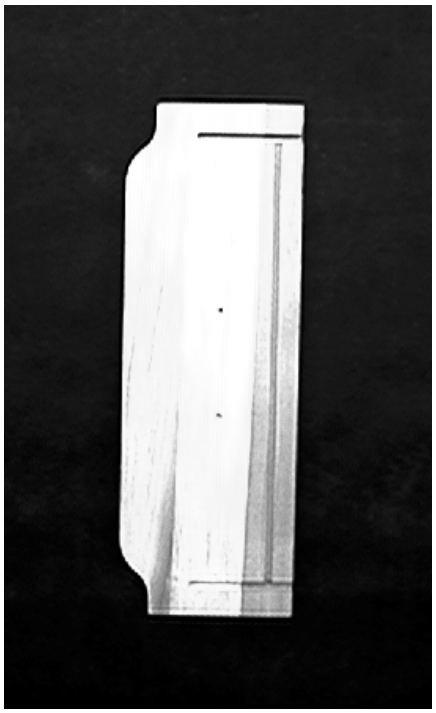
The drawer front was another piece that was used for testing the algorithm. This part too is machined in multiple setups. However, since a part blank with only one side machined could not be obtained, the algorithm was tested just to determine the actual pose of the part. Figure 6.12 (a) and Figure 6.12 (b) show the solid models from both the inner and outer sides of the part.



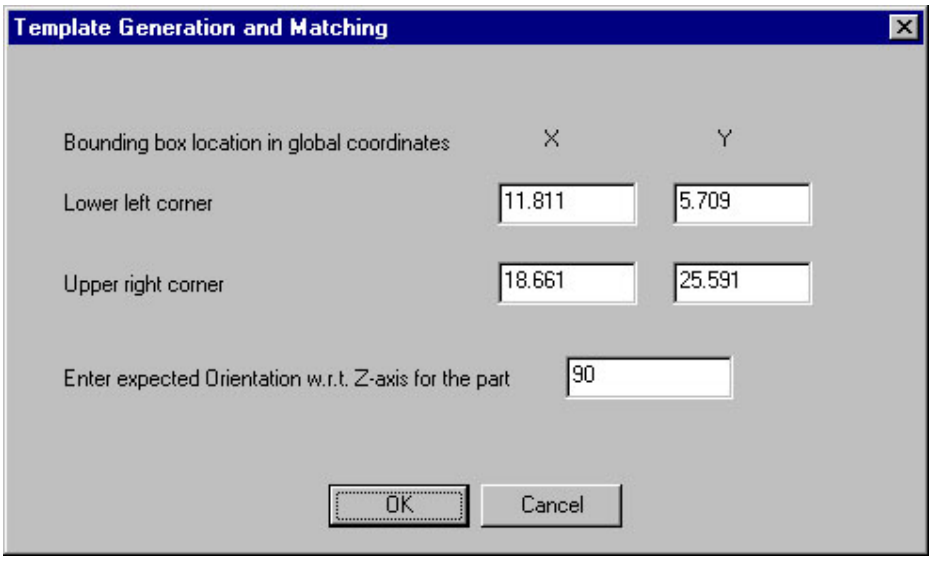
(a) (b)

Figure 6.12 (a) Inner side of the drawer front; and (b) Outer side of the part

The image of the part at approximately 90° with respect to the Z-axis was captured and used as input for orientation and location determination. Figure 6.13 (a) and 6.13 (b) show the input image and input parameters respectively for the algorithm.



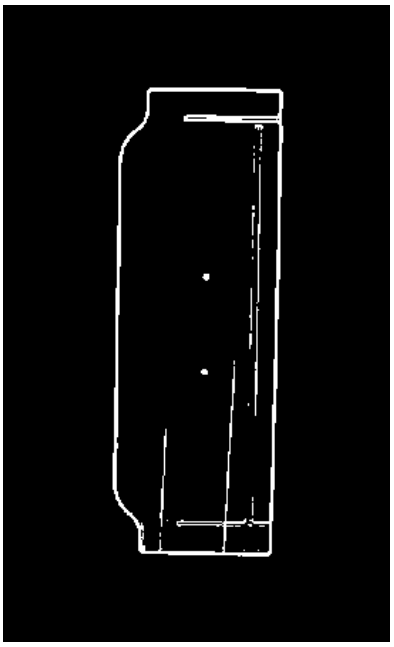
(a)



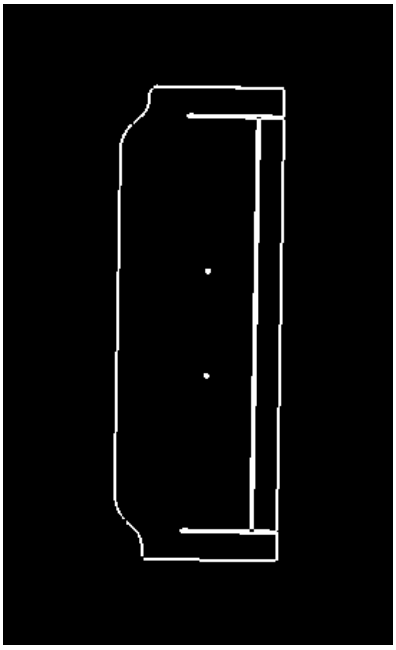
(b)

Figure 6.13 (a) The input image of the drawer front; and (b) Input parameters for the algorithm

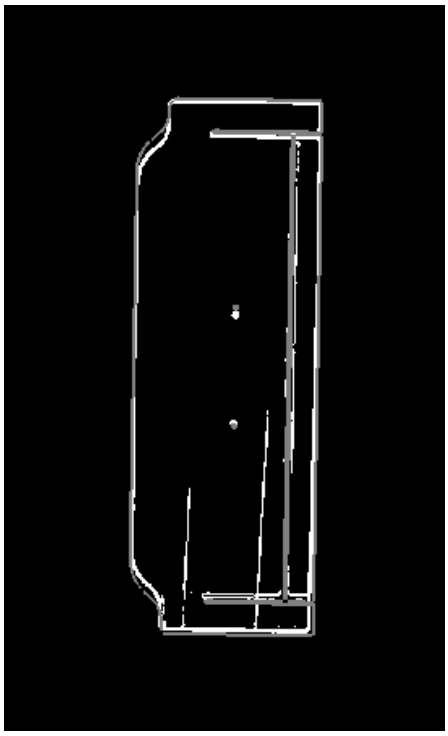
The resultant image after the application of edge extractors in Figure 6.13 (a) is shown in Figure 6.14 (a). This image acts as the input image for the template-matching algorithm. The selected template is shown in Figure 6.14 (b), while Figure 6.14 (c) shows the match between the template and input image.



(a)



(b)



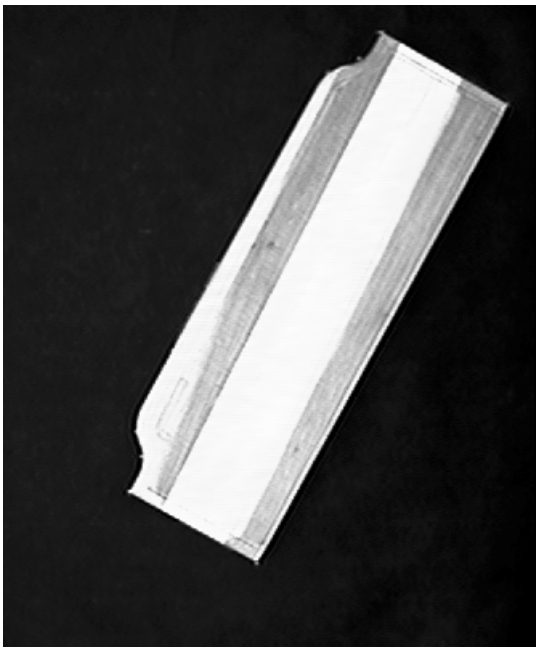
(c)

Figure 6.14 (a) Resultant image upon edge extraction on Figure 6.13 (a); (b) The selected template image; and (c) Image illustrating the match between the input and template

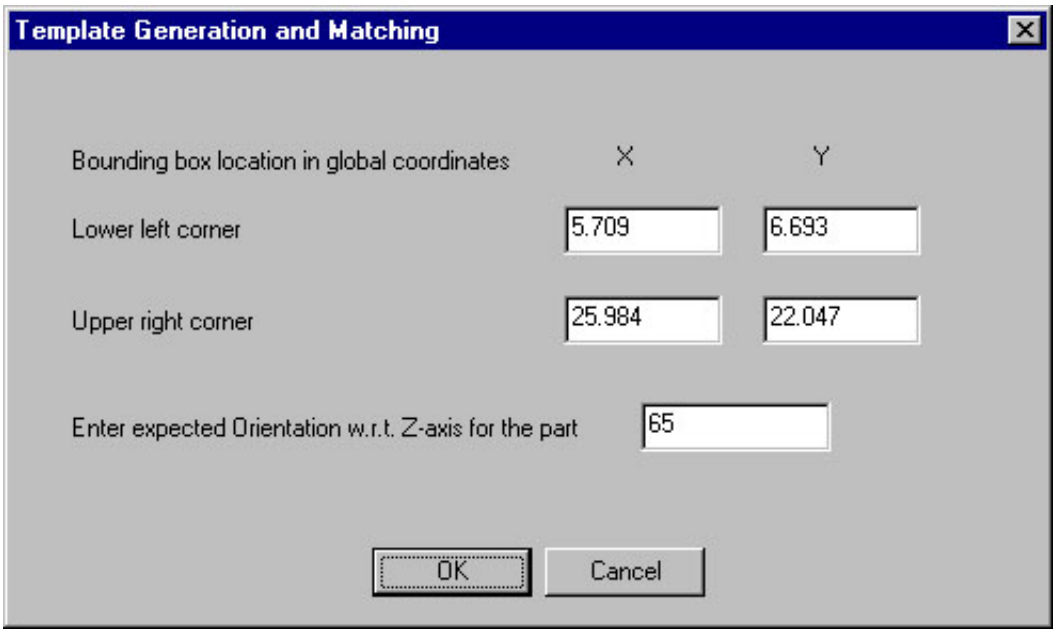
The orientation of the object was computed accurately as 89° with respect to the Z-axis. The orientations about the X and Y-axes were 180° and 0° respectively. The degree of match was found to be 35.44 percent. The low value could be attributed again to the texture resulting from wood grain that is present in the image resulting from performing edge extraction on the input image. The location of the center was computed to be (15.207, 15.620). The value obtained from manual measurement was (15.236, 15.650). The error in this case was 0.19 percent along the X-axis. The error along the Y-axis was 0.192 percent. The value for the orientation matches exactly with the measured value. This value can now be used to update the CNC part program appropriately.

Since the nesting software is used to maximize the utilization of parts placed in the bed of router, most rectangular parts are oriented at either 0 or 90° with respect to the Z-axis. In order to demonstrate the applicability of the approach to arbitrary orientations, an image

of the CD case was captured and used as input for the algorithm. The following figures show the application of the algorithm to this arbitrary orientation.



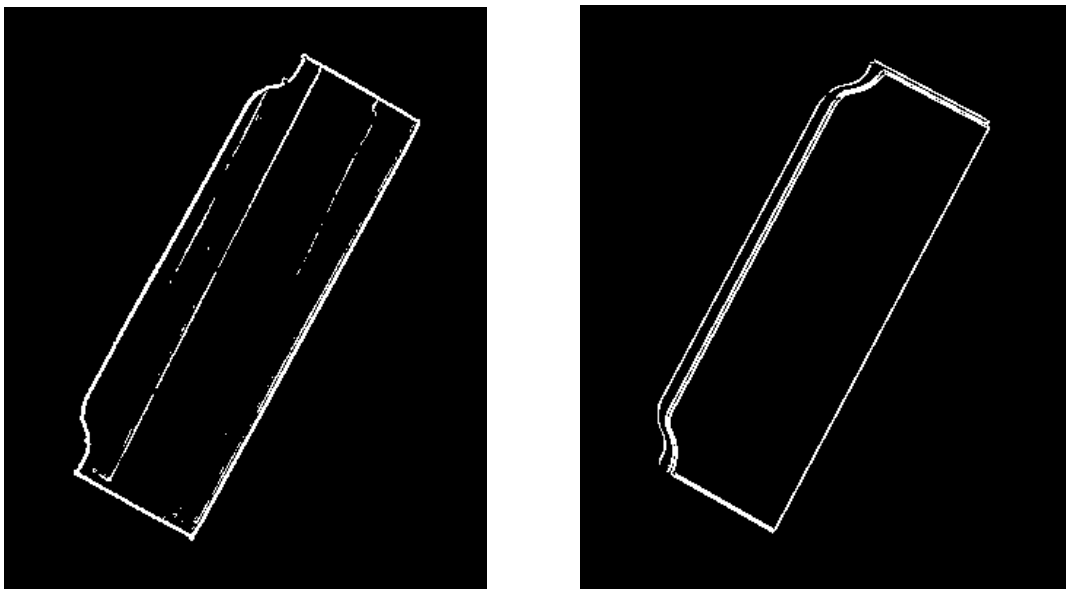
(a)



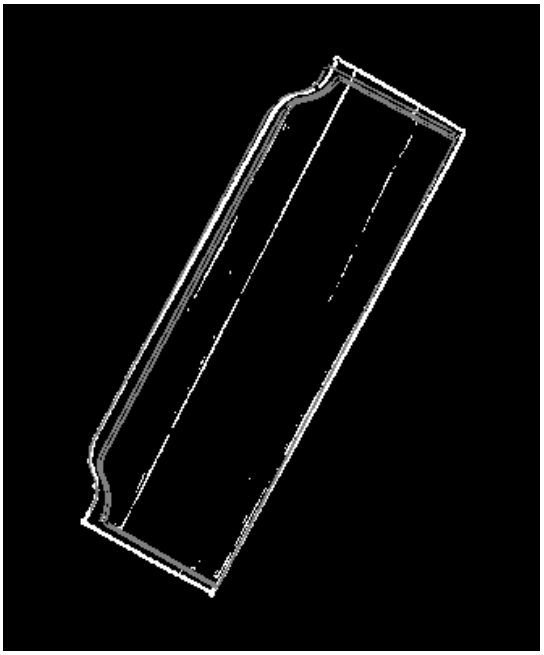
(b)

Figure 6.15 (a) The input image; and (b) The input parameters

The input image and the input parameters are shown in Figure 6.15. The results of the application of the algorithm are shown Figure 6.16.



(a) (b)



(c)

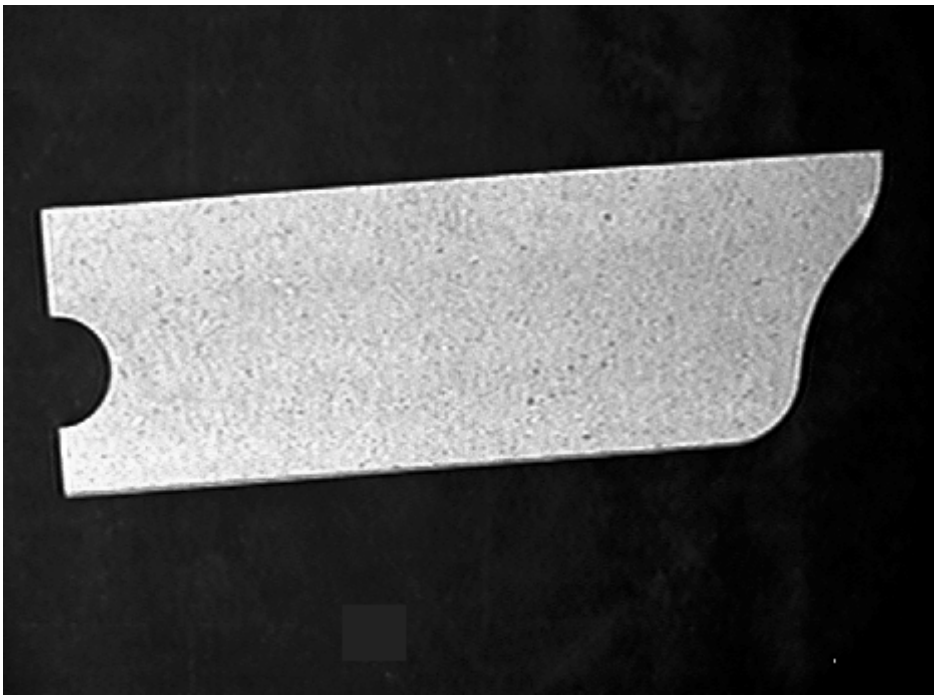
Figure 6.16 (a) Input image for template matching; (b) The selected template; and (c) Image illustrating the match between the input and template

The orientation of the object was determined as 62° with respect to Z-axis, and 180° each with the X and Y-axes. Manual measurement of this angle yielded the result as 61.5°. The error in measurement was thus 0.8 percent. The error in this case can be attributed to the fact that the algorithm searches for part orientation only at 1° increments. Since the actual value was between the 61 and 62°, the algorithm produced a slight error. The location of the center was determined to be (15.847, 14.370). The actual value from manual measurement was (15.847, 14.370). The degree of match was 23.21 percent. The low value is again attributed to a certain extent to the wood grains. The fact that some features on the part do not create large brightness variations along the edges also demeans the match percentage. However, the pose and body center location values corresponding to the maximum match are in fact, quite close to the actual values. This is despite the pronounced wood grains present in the input image, which lends credibility to the approach's efficacy. The solid model of the part was approximated from the physical part, and that has resulted in a model that varies to a certain extent from the actual part. Even under these conditions, the approach seems to work correctly.

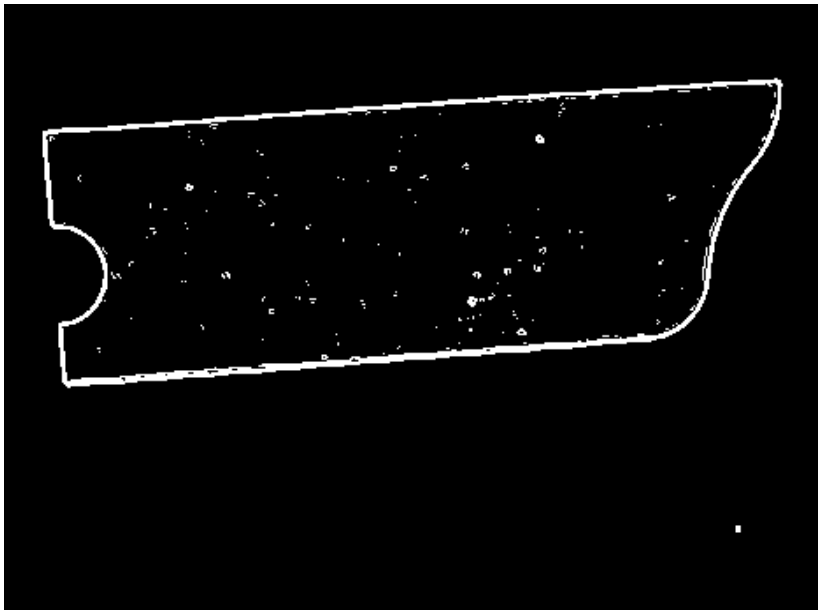
It is worth noting that during testing, the images captured with the setup had a relatively low resolution. On an average, the resolution obtained with the current experimental setup is about 16 pixels/inch, or 0.06 inch/pixel (approximately $1/16^{th}$ of an inch), which is quite low. Under these conditions, the measured errors in the actual and computed values for the center are almost equal to the resolution of the camera. A vision setup that can accord a better resolution will yield more accurate results.

6.3 Limitations

The approach mentioned above does have some limitations. The presence of noise in the input image can lead to erroneous results. The noise could lead to the location of the edges in the image away from the center of the image after it is sized appropriately with the cropping algorithm. If the table or bed of the router on which parts are placed is not clean, it could cause noise in the captured image. The image in Figure 6.17 (a) and Figure 6.17 (b) illustrates this potential condition.



(a)



(b)

Figure 6.17 (a) Image with presence of noise; and (b) Resultant image after edge extraction and cropping.

The noise illustrated in Figure 6.14 can be generated by the presence of extraneous, light colored particles on the bed of the router. In most cases, proper image preprocessing that eliminates small isolated "dots" in the image will reduce problems caused by this type of noise. Without such preprocessing, the computed center of the image generated after edge extraction and cropping will be incorrect. For the untouched image shown in Figure 6.17(b), the algorithm concludes that the initial orientation was either incorrectly specified, or that the solid model of a different part was selected. The maximum match was merely 3.85 percent in the range between ± 5° of horizontal. The actual orientation of the object in the image was 4° with respect to the Z-axis, and the noise was added manually. This could be the biggest limitation of the approach. Moreover, the location of the noise too can have implications on whether the algorithm will work. If the noise is quite close to the part boundary, the shift in the part location in the image after edge extraction will not be substantial, and the approach will still work reasonably well. In these cases, so-called "edge thinning" algorithms may also be applied.

For parts that need to be machined with multiple setups, the solid models of the intermediate states of the part are needed if the geometry of the part will be modified after the setup. If the geomteries are vastly different, the degree of match after the first setup and the solid model of the finished part is likely to be very low. This is not seen as an undue burden, as most CAD solid modelers allow individual features to be suppressed with relative ease. In this case, the features machined following each setup can be suppressed, and the resulting solid model can be exported in the STL file format. The side of the part that will be aligned along the direction of viewing with respect to camera must be mapped along the Z-axis of the STL file.

7. Conclusions and Recommendations

7.1 Conclusions

The universal fixture holds tremendous potential for the furniture industry in reducing lead times. The usage of these fixtures for machining parts requiring multiple setups is particularly promising. This thesis was focussed on solving the problems regarding the location and orientation of parts when they are machined with universal fixtures. A computer vision based sensing technique was identified as a possible means for addressing the issue. The first part of this thesis reviews the popular techniques used for object recognition and pose determination and details the application of some of these algorithms. The presence of textures on furniture parts because of wood grains was identified as the main reason for the failure of these approaches in the current scenario, and the implementations have been documented in chapter 4. This chapter also details the application of template matching and its selection to solve the problem at hand.

The usage of template matching for pose determination in a scenario where the number of parts to be identified with the vision system is very large poses new challenges. Chapter 5 touches upon the difficulties encountered with the selected approach. A novel way of generating the templates in conjunction with CAD modelers was hence proposed to handle these problems. The methodology and implementation of the proposed algorithm for automatic generation of view dependent templates has been given in this part of the thesis. A variation of the maximum correlation was chosen as the criteria for deciding the match between the input image and the generated template. The algorithm was tested on some simple rectangular part blanks as well as parts with intricate geometry resultant after the first machining setup. The results were found to be satisfactory and error in determining orientation was consistently negligible. The average error in determination of the location was less than 0.2 percent and can be attributed to the low-resolution obtained from the current vision setup.

The current research effort has led to the development of a technique for determination of the pose of wooden parts as they are placed on the bed of a CNC router. The application of template matching in conjunction with CAD models to circumvent the problems arising out of wooden textures is a novelty. Moreover, the approach can be applied for identification of an object, or to determine its pose as long as its solid model can be generated. Since most solid modelers are quite robust, the approach is not restricted to specific geomteries. The patterns caused by wood grains tend to confuse most popular computer vision algorithms and can prove to be a major impediment. Thus, on a more fundamental level, this effort has proposed a way for using computer vision techniques to identify objects having patterns on their surface.

7.2 Future work

The algorithms developed as part of this research effort can provide a good starting point for techniques involving visual sensors to mechanically manipulate, locate or recognize wooden parts. The ability to handle wooden parts can also be extended for applications involving parts with a texture on the surface. These textures could, for example, be a result of changes in color along a surface of the part, or because of some manufacturing activity like embossing. The algorithms can also be extended for recognition and manipulation of wooden parts that have been stacked up. Three-dimensional templates can be prepared for an individual part to get information about depth. This can then be used to handle individual parts in a stack.

In the current implementation, the user has to select the solid model of the part presented to the vision system. An interesting extension of the work could be the identification of some metrics that populate a database so as to enable automatic selection of the solid model, given an image of the object. The parts used for pose determination in this scenario are laid out flat on the bed of the router. The algorithms however can easily be extended to search a 3D space to determine arbitrary orientations of objects. Another practical extension would be defining the maximum allowable tolerance on orientation and location of the part, and using the algorithms developed to decide if the object is within the expected limits. Figure 7.1 illustrates this approach.

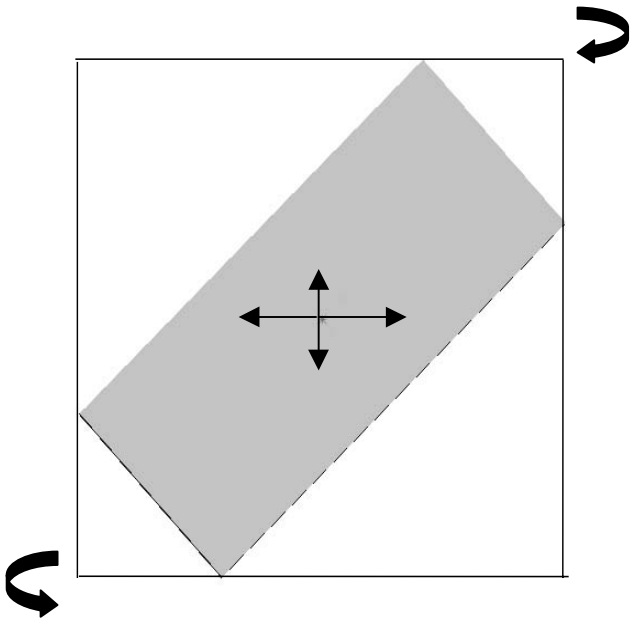


Figure 7.1 Determination of the maximum possible deviations

In Figure 7.1, the possible deviations of the part, as it is placed on the router bed before being machined are shown. The prescribed tolerances along the X and Y-axis are say $1/8^{th}$ of an inch respectively. The coordinates for the bounding box in the extreme conditions are easily determined, if the allowable tolerance in rotation about the Z-axis is specified as say ±5°. The pixels in the objects image, which have the minimum and maximum x, y coordinates can now be subjected to the constraining conditions of a displacement of $1/16^{th}$ of an inch along ±X and ± Y axes, and ± 5° rotation in the clockwise and counterclockwise directions. The values resulting from these transformations will determine the maximum allowable deviation of the bounding box for a given location and orientation. Equation 7.1 gives the transformations involved in the computation.

$$\begin{bmatrix} x & y & z \end{bmatrix}^T = S \bullet \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{Equation 7.1}$$

where S is the matrix resultant upon application of translation and is given in as:

$$S = \begin{bmatrix} x_1 & y_1 & z_1 \end{bmatrix}^T \bullet \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ **Equation 7.2**

In the equations above, x_1, y_1, z_1 are the initial coordinates of a given point, d_x, d_y, and d_z are the displacements and θ is 5° or -5°. Depending upon whether the bounding box of a part being inspected is within or outside the tolerance zone, the part program can either be updated, or the operator is asked to reorient and relocate the part.

Thus, the algorithms developed as a part of this effort have a lot of potential in reducing lead times for furniture manufacturers. The ability to specify tighter tolerances on parts being machined with the router can also lead to substantial savings on raw materials, and thus increase profitability. As furniture companies run on lean profits, they need such technologies for rapid and efficient product development and smaller factory-to-market time. The technique also can provide a suitable starting point for other similar applications.

8. References

R. C. Bolles, P. Horuad, “3DPO: A Three Dimensional Part Orientation System”. International Journal of Robotics Research., 5(3): 3-26 Fall 1986.

W. N. Martin, J. K. Aggarwal, “Volumetric Descriptions of Objects from multiple views”. IEEE Transactions on Pattern Analysis and Machine Intelligence. PAMI-5(2):150-158. March ’86.

P. J. Flynn, A.K. Jain, “CAD-Based Computer Vision: From CAD Models to Relational Graphs”. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 13, No. 2: 114-132. 1991.

J.H.M. Byne, J.A.D.W. Anderson, “A CAD-based computer vision system”. Image and Vision Computing 16: 533-539. 1998.

W. P. Cheung, C. K. Lee, K. C. Li, “Surface Reconstruction by integrating Stereo Vision with Shading”. International Conference on Industrial Electronics, Control and Instrumentation. Volume 2: 968 –971. 1994.

H Najjari, S. J. Steiner , “3D CAD based Object Recognition for a Flexible Assembly Cell”. SPIE Proceedings on Intelligent Robots and Computer Vision XV: Algorithms, Techniques, Active Vision and Materials Handling. Vol 2904: 167-177. 1996.

C. Hansen, T. C. Henderson, “CAGD-Based Computer vision”. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 11, no. 11: 1191-1193. 1989.

Ales Ude, “Automatic CAD-based Construction of Hybrid Models for 3D Object Recognition and Localization”. Proceedings of the IEEE International Conference on Intelligent Engineering Systems. 561-566. 1997.

David Vernon, “Machine Vision”. Prentice Hall International (UK) Ltd., 1991.

Dana Ballard, Christopher Brown, “Computer Vision”. Prentice Hall Inc. 1982.

http://www.cs.rochester.edu/u/brown/172/assts/UFind.html

http://www.unchainedgeometry.com/jbloom/dissertation/chptr4-nonman.pdf

S. Rock, M. Wozny, “Generating topological information from a ‘bucket of facets’”. Solid Freeform Fabrications Symposium Proceeding, University of Texas at Austin. 251-259, 1992.

Denis Cormier; Kittinan Unnanon; Ezat Sanii, “Specifying non-uniform cusp heights as a potential aid for adaptive slicing”. Rapid Prototyping Journal, Vol 6, no 3: 204 – 212, 2000.

Foley, van Dam, Feiner, Hughes, “Computer Graphics: Principles and Practice”. Second edition, Addison-Wesley Publishing Company Inc, 1991.

R. O. Duda, P. E. Hart, “Use of the Hough transformation to detect lines and curves in pictures”. Communications of the ACM, Vol. 15, No. 1: 11-15, January 1972.

http://www.cs.ucdavis.edu/~ma/ECS175_S00/Notes/0411_a.pdf

M. Lynch, “Parametric programming for computer numeric control machine tools and touch probes”, SME publications.

9. Appendix

9.1 STL File Format

The software developed accepts the geometry of the solid model in the form of an STL file. In the STL file, tessellations are mapped onto the surface of the part. The solid model is approximated by a number of triangular facets, which are stored in a particular way in a binary STL file. The triangular facets are defined by, their vertices and normal vectors. The vertices are defined by the x, y and z coordinates. The normal vectors, i, j, k, determine the surface orientation. Table 9.1 shows the way in which data is stored in an STL file.

Bytes	Data type	Description
80	ASCII	Header. No significant data.
4	Unsigned Long Integer	Number of facets in file
4	Float	X component of normal
4	Float	Y component of normal
4	Float	Z component of normal
4	Float	X for vertex 1
4	Float	Y for vertex 1
4	Float	Z for vertex 1
4	Float	X for vertex 2
4	Float	Y for vertex 2
4	Float	Z for vertex 2
4	Float	X for vertex 3
4	Float	Y for vertex 3
4	Float	Z for vertex 3
2	Unsigned Integer	Attribute byte count

Table 9.1 The STL file format

The STL file starts with the “Header” and the information on the number of facets in the file. The Header takes 80 bytes and doesn’t have any data significance other than providing the title for the file. The information on the number of facets takes 4 bytes. The facet is represented by the facet normal (i, j, k) and the co-ordinates of the vertices (x, y, z) for each facet.