Multiple Object Recognition on the Assembly Line

Mithun George Jacob*, V. G. Sridhar**
School of Mechanical and Building Sciences
Vellore Institute of Technology
** III-Year, B.Tech Mechanical Engineering*
Email: mithunjacob_oohay@yahoo.com
*** Selection-Grade Lecturer, Dept. of Mechanical Engineering*
Email: vgsridhar@vit.ac.in

A system is presented for the visual object recognition of objects on an assembly line. It is capable of identifying individual objects from a set of objects with the help of an image database.
A combination of algorithms is presented to segment the image into regions containing one component each and using the database to rapidly match the component with the database using data stored in the contour chain. Classification is executed by comparing the feature vector of the presented component with a decision boundary obtained by linear regression analysis of the database. Experimental results for a sample set of real-world objects demonstrate the robustness and accuracy of this approach.

1. INTRODUCTION

Object recognition on the assembly line is a well explored field (Geisselmann, 1986) and various investigations have been made in the application of image processing techniques to assist and direct robots on the assembly line (Loughlin, 1987). Identifying individual objects on the assembly line is simple compared to identifying an object from a collection of objects and various contour-based approaches have been used to do so.

This paper explores a system by which a set of components coming down the assembly line can be identified with the help of an image-acquisition device placed vertically above the assembly line. It involves the use of a segmentation algorithm which can be used to identify individual objects in the image rapidly and efficiently. The system architecture is described below with the aid of a block diagram (Figure 1).

2. SYSTEM ARCHITECTURE

2.1 Scene Constraints

The components used to test the algorithms developed were collected from a workshop and were metallic and unpolished. Therefore, a white/black sheet was used as the background to simulate a conveyor belt. The only source of lighting in the system was a 20W incandescent lamp placed vertically above the system at a distance of 60 cm.

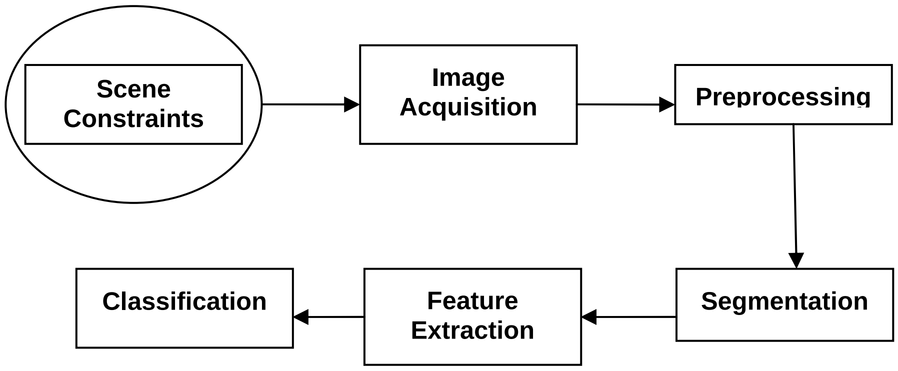

Figure 1 – System Architecture

2.2 Image Acquisition

A Creative® Video Blaster® Web Cam was used as the image capturing device and for increased speed, images of 352x288 pixels were captured during the testing phase. The algorithm is capable of analyzing 640x480 pixel images at reasonable speeds. The source settings for the camera were set to the following parameters.

Table 1 – Image Acquisition Device Parameters

Property	Min	Value	Max
Brightness	0	255	255
Contrast	0	1	7
Exposure	0	154	154
Saturation	0	0	255
Sharpness	0	0	7
Backlight	0	1	1

2.3 Image Preprocessing

Intensity quantization is used to binarize the image. The image is described as a function $I\ (o_i)$ that represents the intensity of pixel o_i. A global thresholding technique is used to preprocess the image where the threshold values are calculated by line-scanning the entire image to obtain the lowest and highest intensity values (w_0 and w_1, resp.). This has proved to be satisfactory as the image histogram obtained is bimodal thus making the selection of the threshold value fairly straightforward.

Using the threshold values of w_0 and w_1 for the lowest and highest intensity values, the image is binarized with the following threshold function.

$$IF\ \ D(I(o_i),w_1) < D(I(o_i),w_0)\ \ I(o_i) = 1$$

$$IF\ \ D(I(o_i),w_1) > D(I(o_i),w_0)\ \ I(o_i) = 0$$

The function *D (a, b)* returns the distance between *I* (o_a) & *I* (o_b) using the RGB values of a pixel o_i , represented by R_i, G_i or B_i. *D (a, b)* is defined as follows:

$$D(a,b) = \sqrt{(R_a - R_b)^2 + (G_a - G_b)^2 + (B_a - B_b)^2}$$

The function *D(a,b)* is repeatedly used in this paper to ascertain the degree of closeness between the intensities of two pixels *a, and b.*

2.4 Segmentation

Segmentation is executed with a combination of algorithms which assumes that the robot is equipped with an image database consisting of a variety of images in different orientations. The object recognition strategy proposed here is a two-stage process.

2.4.1 Contour Extraction

The image is line-scanned horizontally and vertically and points of intensity difference are sought and marked by assigning a value w_c to it. The contour is extracted with the following function:

$$IF \quad D(\,I\,(o_{i-1}\,), w_0\,) \neq D(I\,(o_i\,), w_0\,)\,) \;\; THEN \;\; I\,(o_i\,) = w_c$$

Since the image is scanned horizontally and then vertically to obtain the contour points, the points will not be in the appropriate sequence required to extract further information from the image.

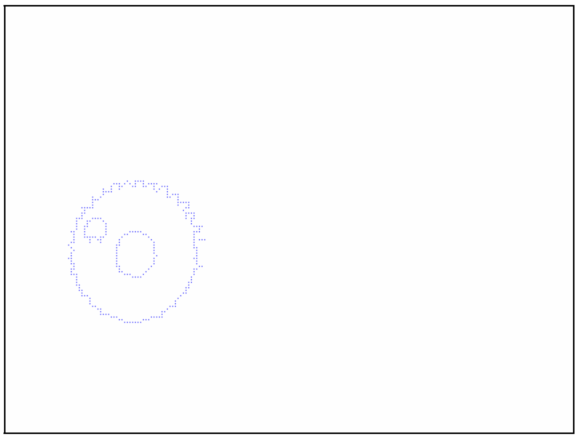
Figure 2 – Extracted Contour

Therefore, these contour points have to be sorted. Please note that if there are multiple components in the image, the obtained contour chain will contain the contour points of all the components present in the image.

The contour chain is sorted by choosing an arbitrary point on the contour chain, say *Contour[i]* and finding a point closest to it, say *Contour[x]*. Then, *Contour[x]* will be swapped with *Contour [i+1]*. This is repeated for *Contour [i+1]* until every point on the contour chain has been sorted.

2.4.2 Vertex Extraction

The contour is then sampled at every *n* points by examining points in *I* $(o_{i,\,i+n})$ and if there exists a radical difference in the gradient (determined by the *G (a, b)* function), between 2 samples, the range of *2n* points is re-examined with $G(I(o_i), I(o_{i+1}))$ to ascertain the exact point of gradient change denoted by w_v. The function *S* uses the coordinates of each pixel defined as ($I(o_a)_x, I(o_a)_y$) and is defined below.

$$G(I(o_a), I(o_b)) = \frac{I(o_a)_y - I(o_b)_y}{I(o_a)_x - I(ob)_x}$$

$$IF\ \ G(I(o_{i-1}), I(o_i)) \equiv G(I(o_i), I(o_{i+1}))\ THEN\ \ I(o_i) = I(o_i)$$

$$ELSE\ \ I(o_i) = w_v$$

During image processing, the black & white pixels in the image will be retained to facilitate feature extraction described below. The aforementioned pixels have been removed in Figure 2 to present a clear view of the obtained contour. During actual image processing white pixels denote the area occupied by the body, black pixels denote the background, and blue denotes a contour point and red, a vertex.

2.2 Feature Extraction

Feature extraction is implemented using the contour chain to obtain the minimum-enclosing rectangle of the component. The algorithm used to extract the minimum enclosing rectangle followed by the area, perimeter and number of vertices of the object is given below in pseudo-code.

Input: Contour of image with vertices

Algorithm:

```
1.   i=0  ;  i<Perimeter-1  ; i=i+1
2.   if(Contour[i].y<Limits.y1) Limits.y1=Contour[i].y
     else if(Contour[i].y>Limits.y2) Limits.y2=Contour[i].y;
3.   if(Contour[i].x> Limits.x2) Limits.x2=Contour[i].x;
     else if (Contour[i].x< Limits.x1) Limits.x1=Contour[i].x;
4.   if( absolute(Dist(i,i+1)-Dist(i,i-1)) > MinJump  &&  Limits.Area >
     MinArea)
     Store (Limits);
     Reset ();
     Goto 1;
```

Output: Minimum Enclosing Rectangle(s) of the component(s) present in the image

The algorithm uses the following functions and variables:

- **i=0; i< a ; i=i+1:** describes a loop where *i* has a starting value of 0, and is incremented after every run and ends once it is not less than *a*.
- **Contour[i]:** represents a point on the contour. **Contour[i].x & Contour[i].y** denotes its coordinates. **Perimeter** denotes the length of the entire contour chain. Please note that if there are multiple components present in the image, their contours will also be included in **Contour.**
- **Perimeter:** Number of contour points in **Contour.**
- **Limits:** represents the minimum enclosing rectangle. **Limits.x1, Limits.y1** represents the top-left point of the rectangle and **Limits.x2, Limits.y2** represents the bottom-right point. **Limits.Area** denotes the area of the rectangle obtained.

- **Dist (a, b):** returns the distance between two contour points **Contour[a]** & **Contour[b]** using Cartesian geometry.
- **MinJump:** denotes the minimum distance between 2 points on the contour chain. If the distance between 2 contour points exceeds this value, this implies that the focus has shifted to the contour chain of another component.
- **MinArea:** denotes the minimum area a component should possess to qualify as a component. This value is arbitrarily decided if the image database has not been built, otherwise the area of the smallest object in the database multiplied by a probability factor (generally around 0.8) is used. This feature has been implemented to discard false regions actually created by shadows or other phenomena.
- **Store(Limits):** stores the coordinates of the minimum closing rectangle
- **Reset ():** resets all values to begin acquisition of the minimum closing rectangle of the next component.

Once the minimum enclosing rectangle(s) of the component(s) have been obtained, the area, perimeter and number of vertices of the component(s) enclosed is fairly straightforward. The white pixels are counted to determine the area of the component; the blue & red pixels, to determine the perimeter of the component; and the red pixels, to determine the number of vertices present in the component. Therefore the details of each component present in the acquired image is identified and stored for comparison with the existing image database.

2.5 Classification

The algorithms mentioned above are used to create an image database which will be used during classification. The above algorithms are used to segment the image into different regions containing a component each and thus obtain the parameters of each component in the image. These parameters will be used to identify the component by comparing it with the image database. The decision making process is defined with an algorithm in pseudo-code given below.

Input: Image Database & acquired Image to be processed
Algorithm:

```
1.  i=0; i<Body.count(); ; i=i+1
2.  MatchArea(i);
3.  if(MatchedArea.count() == 1) DisplayImageDetails(); goto 1;
4.  MatchPerimeter(MatchedArea)
5.  if(MatchedPerimeter.count() == 1) DisplayImageDetail(); goto 1;
6.  MatchVertices(MatchedPerimeter);
7.  if(MatchedVertices.count() == 1) DisplayImageDetail(); goto 1;
8.  Unresolved(); goto 1;
```

Output: Identity and position of the components present in the acquired image.

The algorithm uses the following functions and variables:

- **Body.count():** returns the total number of components retrieved from feature extraction

- **MatchArea (i):** matches the area of the component **Body[i]** with all the images in the database and stores the images in the database which most closely match the component being examined in **MatchedArea.** The number of such images is stored in **MatchedArea.count ().**
- **DisplayImageDetails ():** displays all the relevant image details.
- **MatchPerimeter (MatchedArea):** matches the perimeter of all the components in **MatchedArea** with the images in the database and stores the images in the database which most closely match the component being examined in **MatchedPerimeter.**
- **MatchVertices (MatchedPerimeter):** performs a similar operation but the number of vertices in **MatchedPerimeter** is compared with the images in the database and the results are stored in **MatchedVertices**.

The matching of a component using the parametrical details like area, perimeter or number of vertices is explained in the next section where some of the experimental results obtained are displayed.

3. EXPERIMENTAL RESULTS

To evaluate the proposed system, 4 components were chosen from a workshop (MS Flat with a hole, gear, disc and a threaded cylinder). 3 training samples were taken of each image in different orientations and positions.

The images were processed with the algorithms described below and the decision boundary for each component in the terms of area, perimeter and number of vertices were obtained through linear regression analysis. Therefore, the image database is complete and ready to be used for object recognition.

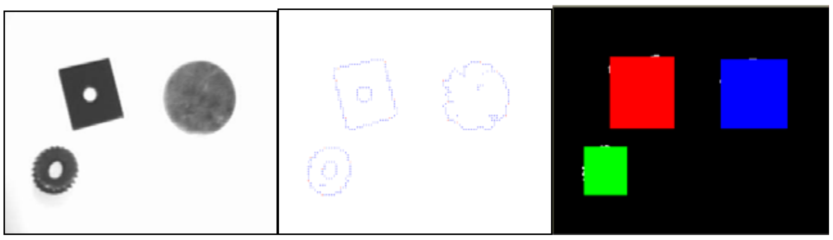

Figure 3 – a) Testing Image (b) Extracted Contour (c) Segmented Image

The captured image (Figure 3a) was analyzed and segmented successfully as shown in Figure 3.b. The data extracted from the testing image is given below.

Table 2 – Component Parameters

Component No	Area(pixels)	Perimeter(pixels)	Vertices(pixels)
1.Flat (Red)	4945	120	6
2.Gear (Green)	2180	88	7
3.Disc (Blue)	5351	114	8

Then the area for each component is compared in the graphs presented below. The square points represent the Area/Perimeter/Vertex count of each component (Red/Green/Blue) in the same order and the linear regression line (black) is obtained from the image database.

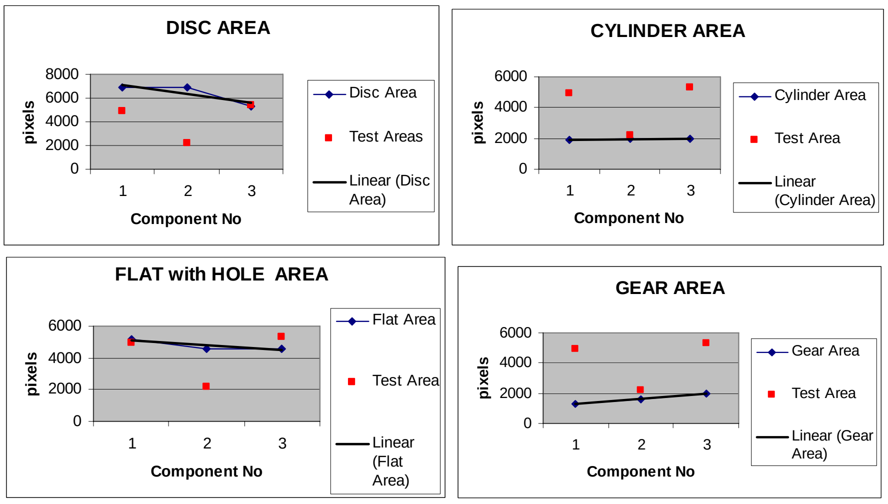

Figure 4 – Area Matching

From Figure 4, it is clearly obvious that *Component 1* is the MS Flat bar, since it closely matches the regression line. But for *Component 2*, we have close matches with the Cylinder and the Gear and for *Component 3*; we have close matches with the MS Flat Bar and the Disc. Therefore, perimeter-matching is required.

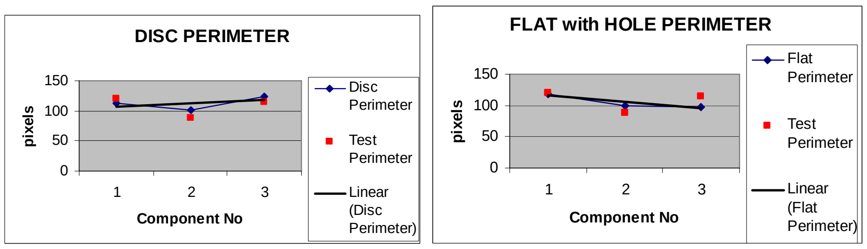

Figure 5 – Perimeter Matching of Component 3

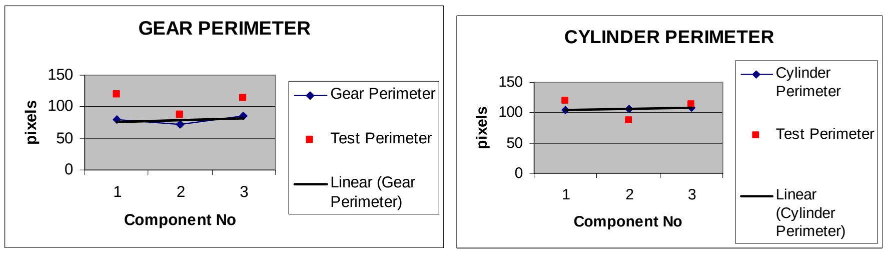

Figure 6 – Perimeter Matching of Component 2

In Fig 6, it is observed that for *Component 2* perimeter-matching has identified the component as the gear as the perimeter point lies closer to the Gear-Perimeter regression line as compared to the Cylinder-Perimeter regression line. But as shown in Figure 5, the ambiguity still remains for *Component 3* between the MS Flat Bar and the Disc which will be resolved through vertex-matching.

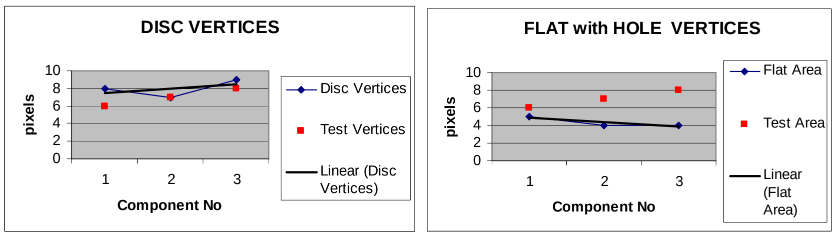

Figure 7 –Matching of Vertices

In Figure 7, the ambiguity is resolved for *Component 3* as it is clearly observed that the Vertex point is closest to the Disc vertex line.

4. CONCLUSION AND FUTURE WORK

The described system can be used to guide robots working on an assembly line through operations like sorting, pick-and-place or counting. The system has proved to be efficient in speed due to its simple, rapid contour extracting system followed by an equally simple and efficient segmentation process.

As it can process images of 352x288 pixels under 1 second using a Pentium IV (3GHz) processor, it is possible to implement this as an actual machine vision system. The algorithms were implemented with software coded in Microsoft Visual C++ 6.0. The system has proved to be reliable by obtaining a 93.75% success rate in the tests and this can be improved with the steps described below.

This binarization algorithm used will be replaced by a more efficient one like the Lloyd-Max algorithm (Lloyd, 1957). The contour extraction algorithm can be improved to obtain a more precise contour chain for metrological purposes. Further improvement to the segmentation algorithm is possible by replacing the minimum-closing rectangle with a minimum-closing polygon to increase efficiency. The segmentation algorithm can also be improved to disregard shadows and other noise causing factors more efficiently. The classification algorithm can be improved by using a probabilistic approach rather than the decision theoretic approach used in this system.

Future work will involve the improvement of the integrity of the contour chain and the speed of the algorithms used. Ongoing work has incorporated some of the steps described above including the ability to detect holes with an improved segmentation algorithm.

5. REFERENCES

1. H. Geisselmann *et al.*, ‘Vision systems in industry: application examples’, *Robotics*, **2**, pp. 19-30, Elsevier Science Publications, (North-Holland), Amsterdam, 1986.
2. C. Loughlin, ‘Automatix provides seal of success for Austin Rover’, *The Industrial Robot*, **14**(3), pp. 145-148, IFS (Publications), Bedford, 1987.
3. S.P. Lloyd. Least Squares quantization in pcm. Technical report, Bell Laboratories, 1957.