Introduction to Machine Vision

Azriel Rosenfeld

ABSTRACT: This paper reviews the basic steps in computer-based recognition of patterns in image data, with emphasis on industrial machine vision.

Introduction

Computers have been used for nearly 30 years for numerous applications involving the automatic or semiautomatic recognition of patterns in image or signal data. Major areas of application include character recognition (document reading), speech recognition, medical applications such as blood counting and x-ray analyses, interpretation of remote sensor imagery, target recognition in reconnaissance data, and many others. A particular application area with growing interest is machine vision for robotics and industrial quality control.

This paper reviews the basic steps involved in such computer-based recognition processes, with emphasis on visual recognition. For concreteness, our examples will be drawn from the domain of industrial machine vision, including such tasks as flaw detection and parts inspection.

The basic components of a machine vision system are shown schematically in Fig. 1. The *sensor,* most commonly a television camera, acquires an image of the object that is to be recognized or inspected. The *digitizer* converts this image into an array of numbers, representing the brightness values of the image at a grid of points; the numbers in the array are called *pixels*. The pixel array is input to the *processor,* a general-purpose or custom-built computer that analyzes the data and makes the necessary decisions (identifying the object, detecting flaws); how this is done is the main subject of this paper. The processor may also have some degree of control over the sensor, the object, or the environment; it may, for example, be able to focus or adjust the gain on the sensor, slow down the movement of the object past the sensor, or vary the illumination.

There is an endless variety of possible vision tasks, but they all tend to involve a common set of basic steps. To detect a flaw on a surface, we must be able to distinguish the flaw from the rest of the surface. The process of discriminating part of an image from its background is called *segmentation*. We may also need to determine its size and shape, so that (for example) if it is too small, we can ignore it. To inspect a complex pattern, we may need to compare it with a reference pattern in order to detect discrepancies. In the following subsections, we discuss the basic processes of segmentation, shape analysis, and matching.

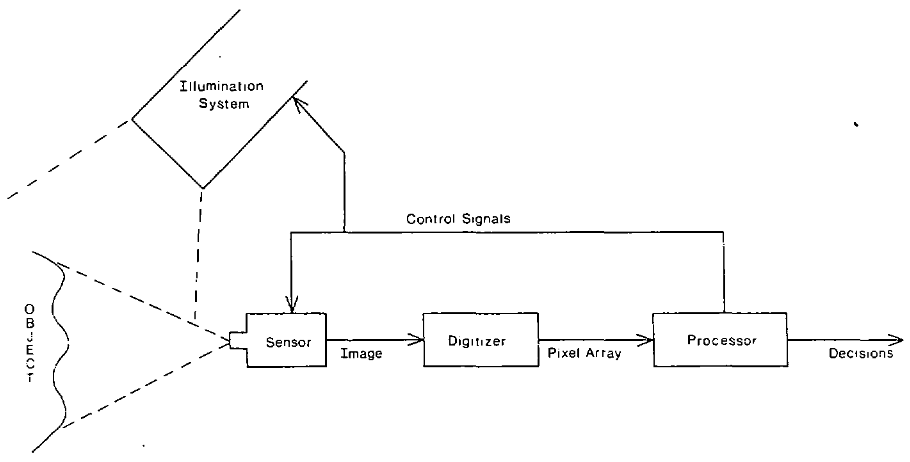

Fig. 1. Components of a machine vision system.

Segmentation

The image comes into the processor as an array of pixels, representing *samples* of the image brightness at a grid of points. The size of the array is limited by the speed of the processor; if it is n by n, the processor must handle n^2 brightness values. A typical size in practical machine vision systems is $n = 256$. On the other hand, the grid of brightness samples must be finer than the smallest features of interest in the image; if it is too coarse, it may miss a feature completely. This limits the field of view that can be processed at one time by the system; the field cannot be more than 100 to 200 times as wide as ‘he smallest feature of interest.

In order to detect an object or a feature (a flaw, for example), we must be able to distinguish it from the surface on which it lies. In other words, we need some means of deciding which pixels belong to the feature and which do not. Two basic methods of doing this are *thresholding* and *edge detection*. These methods will be briefly discussed in the following paragraphs.

Thresholding

In many situations, the objects or features of interest are always darker (or lighter) than their background. We can ensure that this is true by controlling the background or the lighting. For example, if we want to detect dark parts, we can put them on a light-colored conveyor belt. If we want to detect scratches on a flat surface, we can light the surface from its edge; this will cause the scratches to appear very dark because of shadow effects.

The process of distinguishing dark pixels from light ones is called *thresholding*. We compare the value z of each pixel with a "threshold" t; if $z \geq t$, we call the pixel light; if $z \leq t$, we call it dark. (z is called the *gray level* of the pixel; typically z is an integer in the range 0 to 255, i.e., an 8-bit integer.) After thresholding, we can represent the value of a pixel by a single bit—e.g., by 1 if it is dark, and by 0 if it is light. Figures 2 and 3 show an image before and after thresholding.

It is sometimes possible to set the threshold once and for all, so that in the given application, every image is thresholded using the same t. This is a safe approach if the illumination can be carefully controlled, the brightness response of the system is stable, and the contrast between the dark and light pixels is high. A fixed threshold can be applied to the data before it is input to the processor, using a simple comparator circuit; thus, the processor need only deal with 1-bit, or *binary*, data.

In other situations, it is impossible to use a single, fixed threshold. The illumination or

An early version of this paper was presented at the 1984 IEEE Conference on Decision and Control, Las Vegas, Nevada, December 12-14, 1984. Azriel Rosenfeld is Director of the Center for Automation Research, University of Maryland, College Park, MD 20742.

0272-1708/85/0800-0014/$01.00 © 1985 IEEE

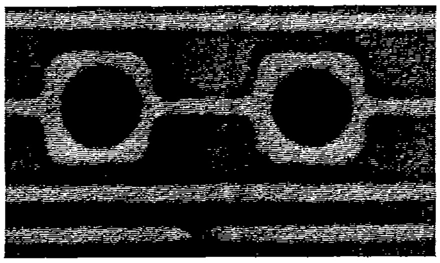
Fig. 2. Image (circuit board).

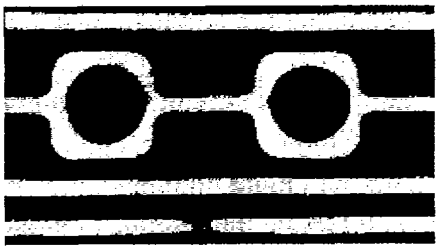
Fig. 3. Image after thresholding.

the reflectivity of the objects may vary from image to image, so that a given threshold may be too high in one case and too low in another. Worse yet, the illumination may vary even across a single image, making the background on one side of the image darker than the features on the other side. In all of these cases, however, the features are darker than their immediate surroundings. If we could automatically set a different threshold for each individual image or for each part of an image, we could extract all the features correctly.

To choose a threshold automatically, we begin by counting how often each gray level occurs in the given image (or piece of image). These counts should be high in certain gray level ranges, corresponding to light background pixels and dark object pixels, and lower in intermediate ranges. Thus, we can pick a threshold automatically for a given image by looking for ranges of high values ("peaks") and then choosing a threshold in the "valley" between them — say, at its lowest point.

Edge Detection

In some cases, it isn't necessary to extract a feature from its background, but only to find the border between them, since the desired information can be determined by examining the shape of the border.

Border pixels can often be identified by the fact that the *rate of change* of the gray level at such pixels is high. We may define this rate of change as follows: let z be the gray level of the given pixel P, and let z_1, z_2, z_3, z_4 be the gray levels of the pixels adjacent to it on its left, right, top, and bottom; these pixels are called its *neighbors*. Then we define the rate of change at pixel P as $\max(0, z_1 - z, z_2 - z, z_3 - z, z_4 - z)$ — in other words, the greatest of the positive differences between the gray levels of P and its four neighbors. Evidently, if P is interior to a homogeneous region of the image, the differences are all small and so is their max; but if P is a low-valued pixel and is adjacent to high-valued pixels, at least one of the differences will be large.

The process of measuring the rate of change of the gray level at each pixel, so as to identify border pixels, is called *edge detection*. Differences of gray levels can be combined in a number of different ways to measure rate of change, but the results are usually quite similar. We can easily distinguish between the border and nonborder pixels; a fixed threshold applied to the rate of change values can usually be used to do this. It should be pointed out that the process of edge detection is insensitive to overall or gradual variations in illumination, since high differences still remain high; thus edge detection can be used even in situations where thresholding is difficult. Figure 4 shows an image after edge detection.

Shape Analysis

Now that we have extracted the features or borders from an image, the next step is usually to determine something about their sizes or shapes. In this section, we describe some of the basic methods that are used to obtain size and shape information.

Topology

A thresholded image may contain many regions of above- (or below-) threshold pixels; we usually want to examine these regions one at a time so as to measure their sizes or shapes. In order to do this, we must first define exactly what we mean by a "region."

Suppose we are given a binary image, and we want to define the regions in it composed of pixels having the value 1 ("1's," for short). Let P and Q be any two 1's, and suppose there exists a sequence of 1's $P = P_0$, $P_1, \ldots, P_n = Q$, such that P_1 is a neighbor of P_0, P_2 is a neighbor of $P_1, \ldots$, and P_n is a neighbor of P_{n-1}. If such a sequence exists, we say that P and Q belong to the same region. The regions defined in this way are called the *connected components* of 1's in the image.

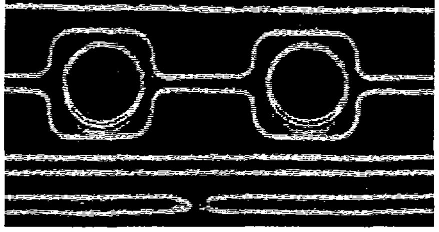
Fig. 4. Image after edge detection.

There exist simple algorithms that, given a binary image, will assign labels to the 1's in such a way that two 1's get the same label if and only if they belong to the same connected component. Figure 5 shows labeled regions in part of the thresholded image.

Once we have assigned unique labels to the regions of 1's, it becomes easy to examine the regions one at a time. We can now easily distinguish between small regions and large ones; if a label occurs only a few times in the image, it must belong to a small region. If the regions of interest are not too small, we can distinguish them from the "noise regions" by applying a fixed threshold to the sizes.

Borders

The shape of a border can be analyzed by examining the turns that the border makes as we move along it. To define more precisely what we mean by moving along a border, think of the pixels as small squares; then a pixel on the border of a region has at least one side along which it meets a square lying outside the region. We can move along the border by following these sides, always keeping the region on our left. Figure 6 shows part of the border of a labeled region.

When we follow a border in this way, we are always moving either horizontally or vertically, and we can only turn by 90 degrees at a time. This makes it hard to distinguish actual sharp turns in the border from turns that are only due to the discrete natures of the pixels. In order to tell them apart, we must first smooth out the tiny turns in the border by fitting straight lines to successive pieces of the border.

Skeletons

Border curvature provides useful information about the shape of a region, but some types of shape information are hard to derive by inspecting the border. For example, we cannot easily detect a narrow "waist" in a region by examining its border; an indentation in the border does not give rise to a "waist" unless it comes near some other part of the border.

Shape features such as waists can be detected by constructing a "skeleton" version of the region called its *medial axis*. This construction involves first computing the shortest distance (in pixel units) from each pixel in the region to the border of the region. Simple

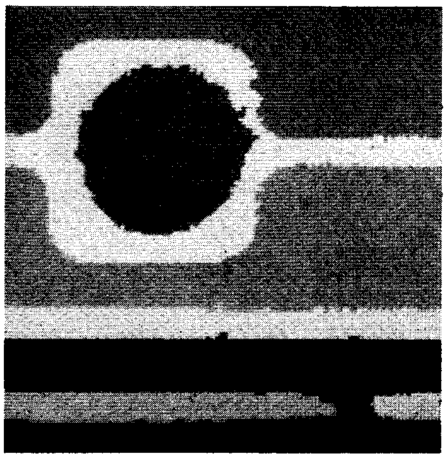
Fig. 5. Labeled regions in a section of the thresholded image.

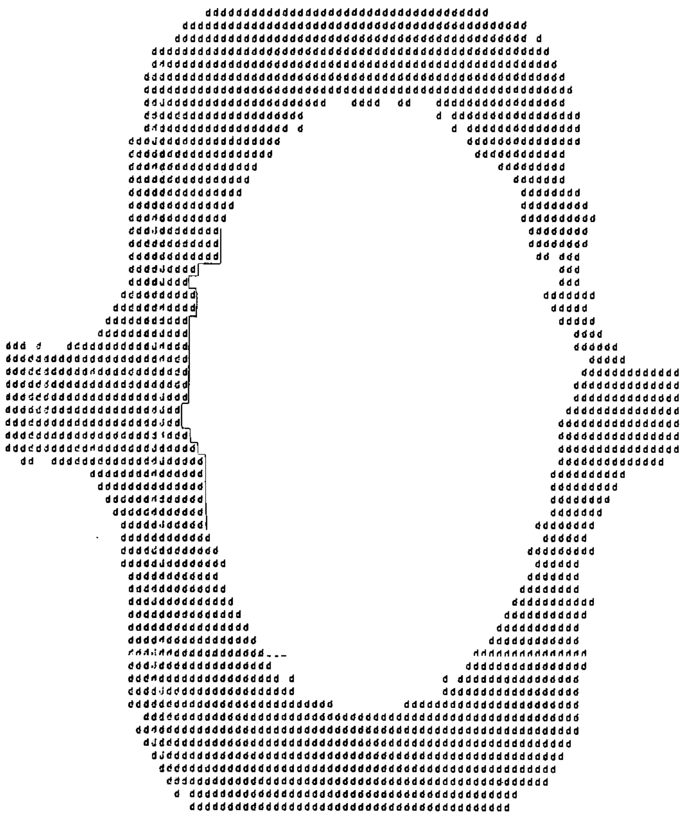
Fig. 6. Part of the border of a labeled region (on the left side of the hole).

algorithms exist that can compute all of these distances very efficiently.

Inspection of the medial axis can tell us whether a shape is wide or narrow, and, in particular, it allows us to detect narrow waists. The distance value at a given point of the medial axis is equal to half the width of the shape at that point; thus, a low value implies that the shape is narrow there.

Detection of narrow parts of a shape is an important technique in the inspection of printed circuit boards. The connectors on the board should have certain standard widths; a piece of connector that is too narrow may be defective. Width measurement based on the medial axis can be used to detect such danger spots.

Model Matching

Many inspection tasks involve comparing the given object with a reference standard and verifying that there are no discrepancies. One way of doing this is known as *template matching*. An image of the object is compared with a reference image, pixel by pixel, and the pixelwise differences are added up; if there is a discrepancy, it will give rise to a region where the differences are high. However, if the observed image and the reference are slightly out of registration, there will be many differences along the borders between light and dark regions in the image, where a slight misalignment can lead to dark pixels being compared with light pixels and vice versa.

A more flexible approach to comparing an image with a standard is to measure a set of properties of the image and to compare the measured values with the corresponding expected values. An example of this *property matching* or *feature matching* approach is the use of width measurements to detect flaws in printed circuits, discussed above. In our work, using this method, the expected width values were relatively high; when we found narrow ones, it was an indication of possible defects.

Some 3-D Vision Concepts

The examples given up to now have all involved essentially two-dimensional tasks. The simple techniques that can be used in such tasks all break down, at least to some extent, when applied to three-dimensional problems, such as recognizing the parts shown in Fig. 7. This image cannot be thresholded in any simple way, because illumination cannot be easily controlled, and the parts cast shadows on one another. Edge detection would also give confusing results, since there is no easy way to distinguish the edges of objects from the edges of shadows. Even if we somehow managed to segment the image, the shapes of the resulting regions (or borders) would not give us reliable information, because we do not know how the objects are oriented; as an object rotates in three dimensions, the shape of its silhouette can change drastically. This, in turn, makes it difficult to compare the image with a model; for a given object, we would need many different models corresponding to its many different possible orientations, each of which gives rise to a differently shaped image.

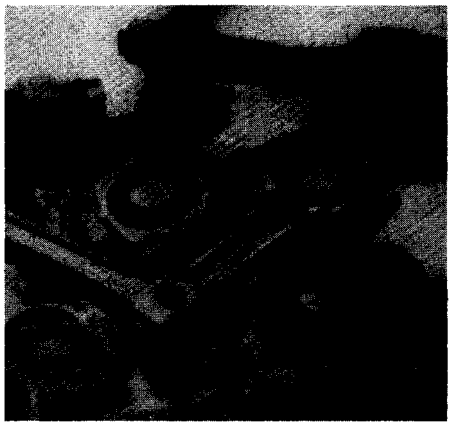
Fig. 7. A 3-D problem: recognizing parts.

3-D vision becomes much easier if we can determine the "topography" of the surfaces that are visible in the image. Each point P in the image represents light received by the sensor from some point P' on the surface of some object in the scene. What we would like to know is how far away P' is from the sensor. This does not tell us everything about the scene; objects can still (partially) hide one another, and we can only see one side of an object. Thus, even if we know all the distances to the visible points, our knowledge about the scene is only " 2 & 1/2–

dimensional," not truly three-dimensional. Nevertheless, this information provides a very good basis for recognizing the objects that are present in the scene.

Inferring 2 & 1/2 D Information

The ideal way to obtain 2 & 1/2 D information about a scene is to measure it directly. Sensors have been developed that can directly measure the distances to all the (visible) points in their field of view. The output of such a sensor is a "range image," i.e., an array of values that represent ranges rather than brightnesses. Describing how such sensors work is beyond the scope of this paper, but we can briefly sketch two approaches. In the first, we illuminate the scene one point at a time with a coherent beam of light and measure the phase shift of the reflected light; from this shift, the range to the point can be computed. In the second approach, we project a pattern of light, such as a grid, onto the scene and analyze how the pattern is distorted when it falls on the surfaces in the scene; the shapes of these surfaces can be deduced from the distortions.

Another way of directly computing 2 & 1/2 D information is to use two images of the scene taken from different viewpoints. If we can identify corresponding scene points in the two images, we can compute the ranges of these points by triangulation. The main difficulty with this *stereomapping* approach is that it is not always easy to find pairs of corresponding points. On a smooth, featureless surface, all points look alike; thus our best bet is to try to match up pairs of distinctive points, such as points that lie on edges. However, such a point may not be visible on both images, since from one viewpoint, it may be hidden by another object, making it impossible to find a correct match for it. So automatic stereomapping is not a trivial task; but it can be quite effective in many situations.

If we have only a single image, we can still make useful inferences from it about the 2 & 1/2 D structure of the scene by making various reasonable assumptions about the surfaces that are present. For example, if we assume that the surfaces reflect light diffusely, we can relate the changes in brightness around a given image point to the surface curvature at the corresponding scene point; this approach is known as "shape from shading" because it infers information about 2 & 1/2 D surface shape from the shading (= brightness variations) in the image. This idea is used in an interesting technique called *photometric stereo,* in which two or more images are taken of the scene from the *same* position; but for each image, we change the direction of illumination. The shading in each image imposes constraints on the orientations of the surfaces in the scene, and by intersecting these constraints, we can determine the orientations uniquely.

Another basis for making reasonable 2 & 1/2 D inferences about the scene is to make simple assumptions about the shapes of regions or surface features. For example, if a surface has regular surface markings, we can infer its orientation from the way the pattern of markings is distorted by perspective; this idea is called "shape from texture." Or, if an object is known to have rectangular surfaces, we can infer its orientation from the way the shapes of these rectangles are distorted in the image; this class of ideas is known as "shape from shape" (i.e., inference of 2 & 1/2 D surface orientation or shape from the 2-D shapes of regions in the image).

Processing 2 & 1/2 D Information

If we know the 2 & 1/2 D shapes of the surfaces in the scene, we can use shape description or matching techniques to inspect the surfaces, or to identify the objects to which the surfaces belong. For example, we can detect bumps or dents in a surface if they give rise to high surface curvature, just as we detected wiggles in a border by the presence of high border curvature. We can recognize objects by comparing properties of the observed shape with the expected property values for the objects. For example, we can characterize a polyhedral object by the shapes of its faces and the angles at which the faces meet.

Another approach to object recognition, embodied in Stanford University's ACRONYM system, is based on inferring 3-D object shapes from the shapes of regions in the image. A given object can only give rise to a restricted set of region shapes in the image, and it can only give rise to particular shapes if it is oriented in particular ways. If we find several regions in the image that can all be accounted for consistently by the presence of a certain object in a certain orientation, then we can reasonably infer that that object is, in fact, present.

Concluding Remarks

This paper has illustrated progress in machine vision systems using examples from the domain of industrial applications. Recognition systems of various degrees of sophistication have been developed by a very large number of firms. Many of the basic 2-D vision techniques have been incorporated into such systems. The major limitation on these systems is computation time; if a task must be performed within a given time, there is a limit on the complexity of the algorithms that can be used. However, microprocessors are steadily getting faster and more powerful, so that this limit will become less and less significant over the coming years.

Two-and-one-half-dimensional vision techniques have not yet come very far out of the laboratory, but during the next few years, we will begin to see applications of these techniques to practical problems. Such applications will require even more computer power, and so will be time-consuming at first; but this, too, will improve steadily.

Machine vision has not yet achieved the power and flexibility of human vision, but it is getting better all the time and can already perform very well in various restricted environments. As the cost of computer power continues to drop, the range of practical applications of machine vision will continue to expand.

References

[1] D. H. Ballard and C. M. Brown, *Computer Vision,* Englewood Cliffs, NJ: Prentice-Hall, 1982.

[2] J. M. Brady (ed.), *Computer Vision,* Amsterdam: North-Holland, 1981.

[3] M. P. Ekstrom (ed.), *Digital Image Processing Techniques,* NY: Academic Press, 1984.

[4] O. D. Faugeras (ed.), *Fundamentals in Computer Vision—An Advanced Course,* Cambridge, UK: Cambridge University Press, 1983.

[5] M. D. Levine, *Vision in Man and Machine,* NY: McGraw-Hill, 1985.

[6] D. Marr, *Vision—A Computational Investigation into the Human Representation and Processing of Visual Information,* San Francisco, CA: Freeman, 1982.

[7] R. Nevatia, *Machine Perception,* Englewood Cliffs, NJ: Prentice-Hall, 1982.

[8] T. Pavlidis, *Algorithms for Graphics and Image Processing,* Rockville, MD: Computer Science Press, 1982.

[9] A. Rosenfeld and A. C. Kak, *Digital Picture Processing,* 2nd ed., NY: Academic Press, 1982.

Azriel Rosenfeld received the Ph.D. in mathematics from Columbia University in 1957. In 1964, after ten years in the defense electronics industry, he joined the University of Maryland, where he is currently Research Professor of Computer Science and Director of the Center for Automation Research. Dr. Rosenfeld is an Editor of the journal *Computer Graphics and Image Processing,* an Associate Editor of several other journals, a Past President of the International Association for Pattern Recognition, and President of the consulting firm ImTech, Inc. He has published 18 books and over 350 papers, most of them dealing with the computer analysis of pictorial information.