Integrating Vision Into a

Computer Integrated Manufacturing System

by

Paula M. Berg

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Industrial Engineering and Operations Research

APPROVED:

H. J. Freeman, Ph.D., P.E., Chair

M. P. Deisenroth, Ph.D.

J. E. LaPorte, Ph.D.

July, 1989

Blacksburg, Virginia

Integrating Vision Into a

Computer Integrated Manufacturing System

by

Paula M. Berg

H. J. Freeman, Ph.D., P.E., Chair

Industrial Engineering and Operations Research

(ABSTRACT)

An industrial vision system is a useful and often integral part of a computer integrated manufacturing system. Successful integration of vision capabilities into a manufacturing system involves extracting from image data the information which has meaning to the task at hand, and communicating that information to the larger system.

The goal of this research was to integrate the activities of a stand-alone vision system into the operation of a manufacturing system; more specifically, the host controller and vision system were expected to work together to determine the status of pallets moving through the system.

Pallet status was based on whether the objects on the pallet were correct in shape, location, and orientation, as compared to a pallet model generated using the microcomputer-based CADKEY CAD program. **Cadd.c**, a C language program developed for this research, extracts object area, perimeter, centroid, and principal angle from the CADKEY model for comparison to counterparts generated by the vision system. This off-line approach to supplying known parameters to the vision system was chosen over the traditional "teach by showing" method to take advantage of existing CAD data and to avoid disruption of the production system.

The actual comparison of model and image data was performed by a program written in VPL, the resident language of the GE Optomation II Vision System. The comparison program relies on another short VPL program to obtain a pixel/inch ratio which equates the disparate units of the two systems.

Model parameters are passed to the vision system via hardware and software links developed as part of this research. Three C language programs enable the host computer to communicate commands and parameters, and receive program results from the vision system.

Preliminary testing of the system revealed that the object location and surface texture, lighting conditions, and pallet background all affected the image parameter calculations and hence the comparison process.

It was clear from further testing, that the principal angle was a poor measurement of orientation, because the vision system had no default value for handling symmetric objects. That problem aside, the system was found to be feasible for a number of pallet configurations.

Acknowledgements

I extend thanks to my committee members for their patience and guidance. Dr. M. P. Deisenroth suggested the topic and helped me to develop the methodology. Dr. H. J. Freeman lent her encouragement and support throughout my entire graduate program. Her editing and advice shaped this document considerably. Dr. J. E. LaPorte answered all my CADKEY related questions and always had time to exchange tales of computer induced woe. A special thanks also goes out to Dr. LaPorte and his colleagues in the Department of Technical Education for allowing me to use the equipment in their offices and labs.

I am very grateful to Dr. C. E. Nunnally for his expertise in the area of serial communications and for introducing me to Mark Potter, who not only diagnosed a sickly Uart, but helped me find and convert code for the vision system communications programs.

Many thanks also to my friend Ko Kurokawa for his advice on Script and on formatting this document.

But most of all, I wish to thank my husband Joel, whose quick mind, quick wit, and unwavering support helped me to see this project through.

Table of Contents

List of Illustrations

List of Tables

Chapter 1: Introduction

A recent article in Daedalus magazine relates a story of how Marvin Minsky, a founding father of Artificial Intelligence, supposedly assigned a summer project to a graduate student. The student was told to connect a camera to a computer and to make the computer describe what it sees. That was twenty years ago and the problem of machine vision has expanded into an entire research industry (Hurlbert,1988).

Although progress has been slower than anticipated, and a general-purpose vision system with the ability to identify arbitrary objects, in realtime, with arbitrary background and lighting conditions will probably not be available for many decades, there have been some areas of limited success (Hindin,1985). Numerous applications of machine vision have been developed and put to use in the automotive, electronic, pharmaceutical, and food processing industries. Inspection tasks are the most common application; a vision system scans items on a production line and detects anomalies in shape, size, surface design, or color. Other industrial uses of machine vision are recognition tasks, where the system identifies given components; metrology, which involves extracting dimensions of a given object; and tracking, where the vision system follows the movement of an object to gain location information needed for eventual placement or assembly (West, 1988).

Many of the advantages afforded by machine vision are clear. In inspection tasks, fatigue and distraction limit the performance of human operators to about 85% accuracy, while an automated vision system can achieve close to 100% accuracy and operate around the clock (Wilkie, 1986). But the real gains in productivity and product quality are not from having a vision system perform any one of these tasks alone, but from integrating the vision system into a production process and using the information so obtained to control or modify the process.

Problem Statement

An industrial vision system consists of two basic parts, a camera or imaging device which constructs an image of an object, and a processor which digitizes and then extracts information from the image in the form of a symbolic description. This information alone is not particularly useful, but when it is compared to a standard measurement of a part (or product), a human operator or a manufacturing cell controller can use the result to formulate decisions. Thus the vision system becomes part of a feedback loop which also includes components for decision making and implementation of these decisions (Figure 1).

For example, imagine hubcaps moving along a conveyor belt. A vision system is programmed to acquire an image of a hubcap, digitize the image, and extract the largest and smallest diameters. If these dimensions are not equal (within a certain tolerance), or if they are substantially different from a given standard, a human operator reading this information from a monitor may decide the hubcap is defective and remove it from the conveyor. Similarly, the information could actuate a mechanical diverter which sweeps the hubcap into a reject bin. After counting a certain number of "rejects," the vision system might send a signal to a cell controller which stops the process until adjustments can be made to alleviate the defects.

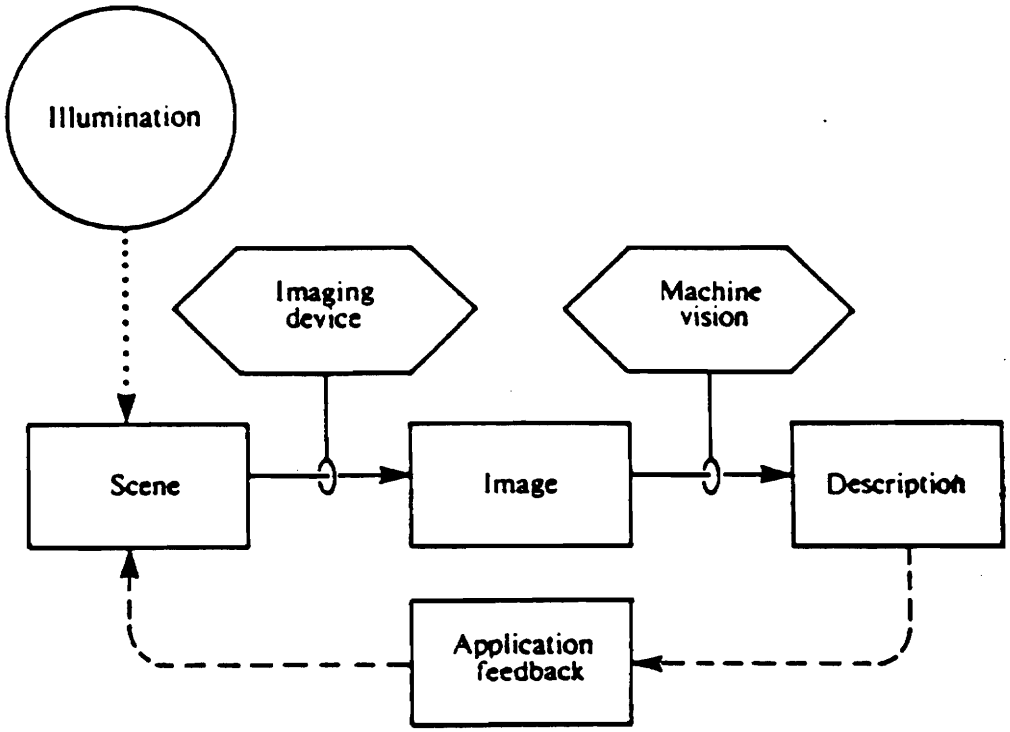


Figure 1. A Machine Vision System: (Horn, 1986)

As this example points out, for the vision system to work effectively within a process, it must 1) be able to extract and manipulate information quickly that is meaningful to the system, and 2) be able to communicate that information or decisions based on that information to the larger system.

As to the first requirement, most industrial vision systems on the market today rely heavily on hardware to process image information quickly, but such systems also need information about known or standard objects in order to make comparisons. One way for a system to obtain this information is the "teach by showing" method. This method makes use of an actual object: an operator presents many views of the object to the camera and the parameters resulting from processing the image are stored for later comparisons. Every time a part is changed or added to the store of "known" objects, an operator must go through this process. Not only is this method disruptive to the production process, but as mentioned by Sridaran (1985), it can be very time consuming, and must rely on a finished part which may not be readily available. Because system hardware and software generally differ between vision systems, once the information is processed by a system, it is in a form which cannot be easily passed to another system. Even among systems of the same make and model, a slight addition to the hardware of one system, or a difference in the objects "taught" to one system, could prevent the systems from being used interchangeably.

Another method for acquiring information for a vision system calls for off-line processing of object information, usually as features of a part's geometry, which are then sent to the vision system as known parameters. The required part geometry is often already available on a Computer Aided Design (CAD) system and, "as many organizations are introducing CAD technology into their work programs, it is becoming increasingly important that information that is already in computer format is used effectively and not regenerated" (Besant, 1986, p.318). This method also avoids other pitfalls of the "teach by showing" method, because it relies on a design rather than a finished part. When an engineer makes changes to the CAD model, there is no further work required and no disruption to the production system in order to update the parameters for the vision system. In addition, engineers do not have to work with the vision system on the shop floor, but can create a design at any off-site location and store it to be used later with a number of different vision systems.

Unfortunately, a description of an object from a CAD model bears little resemblance to that same object as described by a vision system. A vision system produces "measurements" in pixels; the physical meaning of such measurements depends on the camera lens and distance to the object. CAD information is stored in actual units of measurement, i.e., inches or centimeters. If and when the two systems communicate, their object descriptions need be to based on a common unit for comparison purposes.

The choice of descriptive parameters is problematic in itself, since there are many possible views and levels of detail one can use to describe an object. Horn (1986) suggests that this issue may be sidestepped by considering the task for which the description is intended and by choosing the description which allows the system to complete this task. Consider one of the four common vision tasks mentioned earlier in this paper: recognizing parts as they move along a conveyor. One part could be placed on the conveyor in a variety of orientations. To avoid misidentifying the part, the vision system could be familiar with every possible orientation of the part, but it is more practical to choose a descriptive parameter which is not position dependent (e.g. area). Different levels of detail on the part may call for a vision system which can handle additional levels of gray: a binary (black and white system) might not be able to delineate the details. One does not want a description that is overly detailed, however, because there is a definite trade-off in processing time and/or expense as the vision system is expected to extract a larger range of parameters or to process additional gray levels.

As stated in the second requirement listed above, even the most carefully chosen descriptive parameters will be useless unless they can be communicated to the larger system. This exchange may be as simple as displaying values from an object description on a monitor for a human operator, or as complex as a hardware/software link to a manufacturing cell controller.

Firms that have grouped together various pieces of automated-manufacturing and part-transportation equipment to form a manufacturing system may be disappointed to find that most vision systems are sold as stand-alone equipment. Although standard hardware interfaces are built

into these vision systems, the software which allows them to communicate with microcomputers and other equipment is optional and in many cases nonexistent (SME, 1984). A standard communications protocol has yet to be developed, which makes it impractical for vision system manufacturers to develop communications software for immediate use. Firms that want to integrate vision into their existing manufacturing systems have the option of developing the software themselves or contracting the vision system vendor to do the work for them. The final decision depends upon available time, personnel, and money.

Regardless of the option chosen, the same care used to select the descriptive parameters needs to be used in designing the communications interface. As part of a manufacturing system, a vision system and perhaps several other pieces of equipment are controlled by a microcomputer. If there is no human intervention in the system, the cell controller must make decisions normally made by a human operator, such as when to start and stop an operation, how to handle system errors, and where to send output. The controller needs very specific instructions to handle all of these situations. Without this software, the vision system will not be "told" what tasks to perform or when to perform them. If the vision system is not able to perform its job in conjunction with the rest of the cell, it is not truly integrated.

The issue faced in this research is just how to go about integrating vision capability into a manufacturing work cell. The next section takes a closer look at this goal and the steps needed to achieve it.

Research Objectives

The Computer Integrated Manufacturing (CIM) Laboratory of Virginia Tech's Industrial Engineering and Operations Research Department offers a unique opportunity to study a specific ex-

ample of integrating a vision system into a CIM environment. Currently, the lab consists of an automatic storage and retrieval system (AS/RS), a conveyor system, an industrial vision system, two three-axis CNC milling machines, two industrial robots, a programmable controller, and a network of personal computers. When completed, a host computer will oversee the activities of these machines as they transport, machine, and assemble wax parts to create miniature robots and milling machines.

A host computer requests materials from the AS/RS as needed. These materials, which are stored and transported on pallets, exist in three forms: raw (ready to be machined); machined (ready to be assembled); and assembled. A robot arm takes raw materials from "bulk" pallets and transports them to "working" pallets. A completed working pallet contains a "kit," or all of the materials needed to create one unit of a product. The kit then moves to a machining cell. After the parts are machined, the robot arm places them back on the same pallet on which they arrived, but in a slightly different configuration. The kit is then sent to the assembly cell, assembled, and returned to storage.

To accommodate this process, ten different pallet configurations are used, each with a given number of items in predetermined locations. Appendix A contains illustrations of the ten pallet configurations. The robot arm assumes parts are in these locations and moves accordingly.

The task of the vision system is to monitor pallets as they leave the storage system and enter the processing system at the request of the host computer. Pallets which the vision system deems "correct," when compared to a model on an off-line CAD system, continue to a robot to be processed; incorrect pallets are sent back to storage for manual processing. The objective of this research was to set up the mechanism by which the vision system identifies pallets and communicates their status to a host controller. Specifically, meeting this objective entailed:

1. defining parameters that adequately describe the objects on a pallet and their locations,

2. creating a program to extract these descriptive parameters from CADKEY pallet configuration drawings,

3. creating a vision system program to extract the corresponding descriptive parameters from pallet image data,

4. developing a method to equate the disparate measurement units of the CAD and vision systems so that a common unit can be used to compare the parameters extracted from the two systems, and

5. developing both hardware and software links to enable the vision system and host controller to communicate program results, control statements, and error conditions.

This library of programs allows the host controller to integrate the activities of the vision system into the operation of the CIM cell.

Chapter 2: Literature Review

There is a vast amount of literature dealing with the subject of machine vision and the closely related fields of image processing, pattern recognition, and scene analysis. Horn (1986) differentiated between the three related areas by explaining that image processing transforms an existing image to enhance it in some way, while pattern recognition is used to classify objects according to how well they match a given description, and scene analysis transforms simple descriptions into a form which is more suitable for a particular task. The task most closely allied to the needs of this research is pattern recognition: the vision system in the CIM cell compares pallet image parameters and CADKEY model data to determine whether the pallet is "classified" correctly.

Pattern Recognition

West (1988) described three methods used to recognize parts: template matching, global feature method, and statistical correlation. Template matching entails comparing an image of an object with a standard image stored in a vision system or host computer memory. To ensure that the images are aligned, their first and second moments of inertia are computed and used to locate and

orient the object image. After the image is aligned on the template, an algorithm counts any unmatched pixels. All templates are checked in this manner and the template receiving the lowest score (below a preset threshold) is deemed a match.

The global feature method first developed by SRI International (1979) depends on extracting features such as area and perimeter from an image and measuring the difference between the extracted value and stored model values. The object is classified according to its "nearest neighbor."

West (1988) also discussed a method for pattern recognition developed at the University of Karlsruhe. This method uses a sequence of arcs which pass through the centroid of a binary image. The number of pixels that lie on each of 256 arcs define the object. When the 256 numbers are matched with the numbers for known parts, the known part with the highest correlation constitutes a match.

Aleksander, Stonham, and Wilkie (1983) mentioned a method of discriminant functions in which a numerical weight value is associated with each pixel. The total of the intensity value of each pixel multiplied by its weight determines whether the image falls into a class.

For this research, the global feature method of pattern recognition was most appropriate, because the specific vision system being used for this thesis extracts geometric properties from images. The other methods match parts on a pixel by pixel basis -- information which cannot be generated as easily from CAD data.

Feature Selection

All of the methods for pattern recognition rely on one or more features which are extracted from an image as a basis for comparison. Winston (1984) suggested that, for a binary image, the most

direct approach for image identification uses a two-part system with a feature extractor and evaluator. Although the actual choice of features depends on speed and discrimination considerations, he lists the following possibilities:

- total object image area
- hole area
- number of holes
- perimeter
- minimum distance from the centroid to an edge
- maximum distance from the centroid to an edge
- average distance from the centroid to an edge
- length of major axis of an ellipse with equal moment of inertia
- length of minor axis of an ellipse with equal moment of inertia
- total area minus hole area
- ratio of hole area to total area
- ratio of the perimeter squared to area

Pot, Coiffet, and Rines (1983) studied the behavior of features of geometric objects. They proposed that a useful feature must help distinguish one object from another, be fast to extract from image data, and be stable with respect to the variations of images of the same object. Features they chose for their experiments included moment of inertia, moment product, area, and maximum edge distance from the center of inertia to the edge of the image.

A hierarchy of features was described by Gaglio, Morasso, and Tagliasco (1983). They extracted edges from an image, which in turn could be combined to form lines. Lines make up figures, which combine with other figures to create an object description. The object is then classified according to the figures which comprise it.

Feature selection, according to Andrews (1972, p.17) should be "the optimal retention of a minimum number of variables while maintaining and/or maximizing probability of correct classification." Such criteria are hard to quantify and evaluate, therefore features are often selected using an easier but suboptimal method. Because costs (in programming and computer time) often play a role, it is desirable to reduce the number of descriptive parameters while trying to maintain their discriminatory power for recognition purposes.

Petkovic and Hinkle (1987) proposed a method for visual inspection of disk heads. A list of defects was extracted from an object image and compared to a given set of specifications. Instead of writing a procedural-language program to effect this comparison, each specification was seen as a single rule in a rule-based system. If the part being inspected had defects which violated a specification, the specification was printed to a file for that part and saved for later data analysis and process control.

Horn (1986) addressed the subject of binary images and their geometric properties. Properties that he felt most useful for directing an interaction of a mechanical manipulator with a part are the part's area, position, and orientation. The usual practice for determining "position" is extracting the centroid of the object. Orientation (rotation) of an object can be defined by calculating the axis of minimum second moment. These same features are used to define object position and orientation in West's discussion of template matching (West, 1988).

Horn's selection of centroid, area, and axis of minimum second moment was promising for the system studied. These features are easy to compute, and should be available as functions on many commercial vision systems. In addition, these features embody the information needed to compare the model pallets: object, object location, and object orientation.

Vision System Integration

Several examples of integrating vision with other mechanical devices exist in the literature. Vanderbrug, Wilt, and Davis (1983) described how a vision system verified correctness of keycaps used in the assembly of keys on a terminal keyboard. The vision system assembled the keycaps into magazines which were in turn placed on a keyboard base by a Cartesian style robot.

A more complete system description is given by Asfahl (1985) of a system set up to solve Rubik's cube. The system consisted of a host controller, robot arm, and vision system. A video camera under the control of a Zenith Z-89 microcomputer determined the color of each of nine facelets on all six sides of a Rubik's cube. The host computer then knew the starting position of the cube and ran a series of programs which commanded the robot arm to manipulate the cube according to a published solution algorithm.

Another integrated system was discussed by Page and Pugh (1983). They described a system which depended on visual analysis of complex components for the purpose of subsequent mechanical manipulation. The supervisory computer stored a digitized version of a binary image of an object which was "seen" by an optical system. Each component was then represented by features extracted from this image such as area, centroid, and perimeter. The host computer ran a program which decided where a mechanical gripper should grasp the object and controlled the movement of the gripper accordingly.

Unfortunately, none of these examples included program listings of the actual commands or processes that the host computer used to control these mechanical devices. This omission is not totally unexpected, because control programs are very system dependent, and rarely applicable to other situations. General Motors is trying to change all this with the development of Manufacturing Automation Protocol (MAP). Richardson (1988) discussed how MAP can be used for production control communications from the sending of scheduling and machine status information to the

actual downloading of programs into a machine controller. The emphasis in CIM is also upon communication linkages, but between all manufacturing operations and support functions (Asfahl,1985). Many companies have made an effort to standardize the data bases for process planning, production control, material handling, shop floor control, and quality control. This standardized information can be communicated and used system wide for control of the manufacturing process.

On the subject of vision system integration, the literature has shown only specific system examples, and no evidence of a standard method or design for integration. The examples themselves attest to the diversity and system dependence of communications protocols. It is evident that the communications link for the system under study can not be bought "off the shelf" and must be tailored specifically to the system hardware. The next chapter describes the system hardware and the specific methodology needed to integrate this hardware into the activities of a CIM system.

Chapter 3: Methodology

Systems Description

The General Electric (GE) Optomation II Vision System consists of eight custom circuit boards which convert a grey scale image as seen by a solid state camera into a binary image (Figure 2). The user initializes the system by setting a threshold brightness level; all pixels with a brightness level below the threshold are processed as black, all other pixels are considered white. The initialization routine also sets an image area (part) as either black or white and deals with noise according to the needs of the system. In this case, noise is a shadow or smudge that the vision system views erroneously as a bona fide object. By setting a noise level in the initialization routine, the vision system will discount objects with areas smaller than that level.

The first three circuit boards in the GE Vision System generate, store, and display the 244 x 240 pixel thresholded image. The output monitor actually shows four times this number of pixels, because each pixel is actually broken down into four "quarter" pixels, so that the vision system can deal with objects which cover only a fraction of a pixel. The next five boards generate a feature

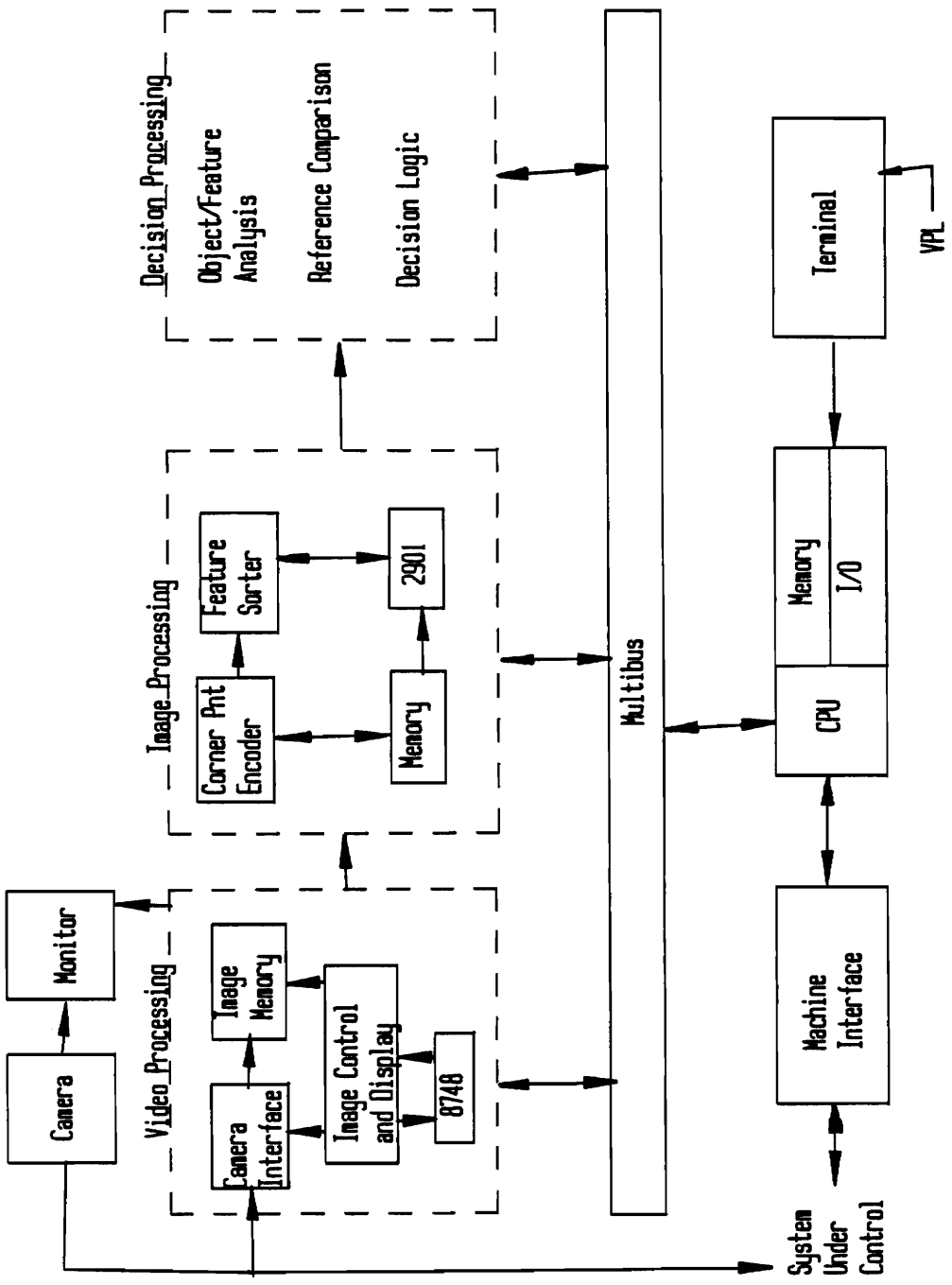


Figure 2. GE Optomation II Block Diagram: (PN2305 Optomation II VPL Language Reference Manual)

data base for the scene and perform extended commands and user-written functions. All of the boards are connected by an Intel Multibus and controlled by an Intel 8086 microprocessor.

A programmer interface panel has connections for a camera, a cassette tape storage system, a video input monitor, a program output monitor, and two standard RS-232C serial interfaces. Either interface can be used to connect to a terminal for creating and running programs. The remaining interface is intended for host computer communications using an asynchronous communications protocol. Data are sent to the vision system controller at 9600 bits per second as eight-bit ASCII characters with no parity and one stop bit.

Testing and program development were performed on an IBM AT Personal Computer using MS DOS, but the programs will eventually reside on an AT&T 6300 PC. The AT&T 6300 controls both the vision and materials handling system; it is one of three microcomputers which control machines at the cell level in the Computer Integrated Manufacturing Laboratory. These three controllers are connected via the AT&T STARLAN network to an AT&T 32B/400 mini-computer system controller.

Program Development

The objective of this research was to set up a method by which a GE Optomation II Vision System could communicate pallet status to an AT&T 6300 host computer. Four programs were developed to reach this goal: a CAD model parameter extraction program, a communications program, a vision system comparison program, and a vision system calibration program (Figure 3). The first step in this process was to use the microcomputer-based CADKEY CAD program to create a scale drawing of each of ten pallets which move through the manufacturing cell. Because the vision system is binary and only "sees" one view of the pallet, the objects could be represented by a two

dimensional silhouette drawing. Each pallet has a characteristic configuration of either bulk, machined, or assembled materials as shown in Appendix A.

The CADKEY models of the pallets are stored in a file as data primitives. A "C" language program extracts descriptive parameters from the CADKEY data primitives and then stores the parameters in a file which is accessible to the AT&T host controller through the STARLAN network. These parameters identify a pallet by the number of objects, their physical features, and their locations on the pallet.

When a pallet enters the system, the host controller initiates a C language program which returns the status (i.e., correct or incorrect) of the pallet. This program emulates the keystrokes made by a human operator by sending commands, character by character, to the vision system controller. These commands, written in Visual Programming Language (VPL), the resident language of the Optomation II system, activate the vision system to view the pallet and extract the same descriptive parameters as stored for the CADKEY model. The host controller program also sends the CADKEY parameters for the corresponding pallet to the vision system.

A VPL calibration program converts inch measurements into pixels. This program does not need to run every time the vision system is in use, but rather only when the camera lens or location is changed. The host controller requests the vision system to run this program and the resulting pixel per inch ratio is stored as a global variable.

At this point, the host controller requests the vision system to run the "comparison" program to compare the pallet image parameters to the CADKEY model parameters feature by feature. If the corresponding parameters are equal (within a certain tolerance), the vision system informs the host that the pallet is correct. Tolerances are included in the comparison program because the vision system is only accurate to one-fourth of a pixel and therefore the image parameters may be slightly larger or smaller than the more accurate CADKEY calculations.

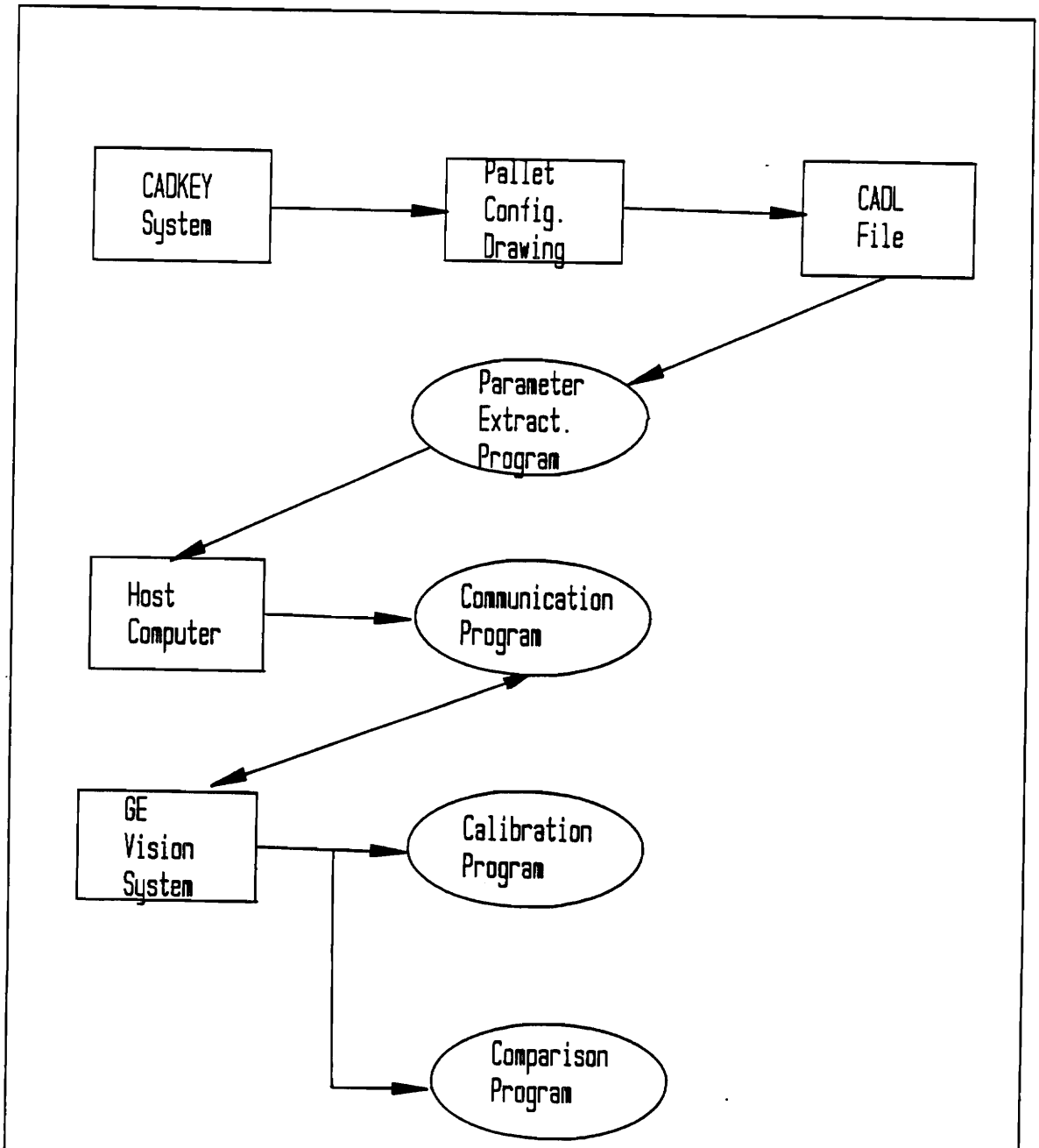


Figure 3. Program Schematic: Results of the parameter extraction program are stored on the STALAN network to be used in the comparison program. Vision system commands and program results are sent to the host controller via the communications program.

In the following three chapters, each of the aforementioned programs and their uses are described in more detail.

Chapter 4: The Parameter Extraction Phase

Pallet Configuration Models

Figure 4 shows a representative CADKEY pallet drawing for a robot working pallet. Pallet configuration drawings for all ten pallets are in Appendix A. Note that the drawing depicts objects on the pallet as they will be seen on the conveyor by the vision system camera. A phantom line delineates the outline of the pallet, while a dashed line represents a hole in a part (if any). A small rectangle in the upper left-hand corner of the pallet is not an actual object, but a "registration" object which is painted onto every pallet. The designer does not need to draw this object on the pallet, because all of the parameters for this object are constant and already known. The significance of this object is explained more fully in relation to the parameter extraction program, which is covered in the next section.

Because the vision system assumes that the origin is in the upper left-hand corner of the screen, and that the y-value increases when moving downward, while CADKEY generally views the y-axis as increasing in the upward direction, the designer must mirror the objects about the center line of the pallet. This ensures that the y-axis on the CADKEY system is aligned with the y-axis of the GE

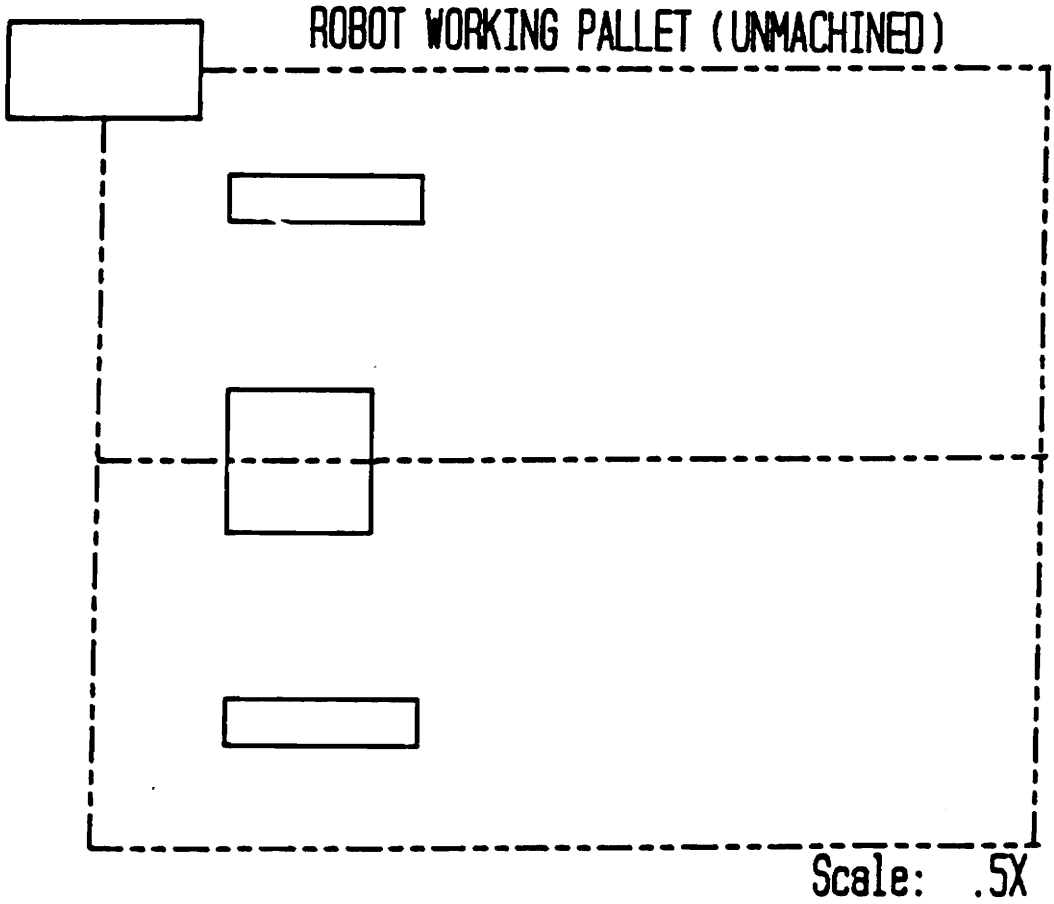


Figure 4. CADKEY Pallet Drawing: These are unmachined parts for a miniature robot.

Vision System. Then each object on the pallet must be grouped and the entire drawing saved as a CADL file. Also, because the GE Vision System considers holes as separate entities with their own areas and properties, holes should be grouped separately from the objects that enclose them, i.e., data about a pallet holding two objects, each of which contains a hole, should be saved as four individual groups.

A CADL file stores pallet configuration drawings as a series of data primitives consisting of arcs, lines, and circles. An example CADL file for Figure 4 is contained in Appendix B.

The Parameter Extraction Program

The next step in determining pallet status is to extract descriptive parameters from the "known" CADKEY model so that they may be compared to the pallet image as seen by the vision system. Remember that a "correct" pallet proceeds to a robot which expects the objects to be in a predetermined configuration. It is not enough to verify that individual objects are correct: their location and orientation must also be correct.

As suggested by Horn (1986), an individual object can be identified by its area and perimeter, the object location by its centroid, and the object orientation by its axis of minimum second moment or principal axis. Because the pallet cannot be precisely aligned under the camera, an object or registration point with known centroid and principal axis serves as a reference point.

The parameter extraction program **cadd.c** converts a CADKEY pallet configuration drawing which is in CADL format to an output file which contains the area, perimeter, centroid location, and principal axis of each object on the model pallet. Appendix C contains a complete program listing.

Cadd.c is an executable file which requires two arguments: an input file name (the CADL file name) and an output file name. The output file has information needed by the vision system to run the comparison program; thus an output file needs to be created and stored on the STARLAN network for each pallet that the vision system is expected to recognize.

The main program in **cadd.c** consists of three functions: **read_data(), group_params()**, and **data_out()**, each of which deserves further explanation.

read_data( )

This function reads a line or data primitive from a CADKEY CADL file. Each data primitive describes one arc, line, or circle which in turn is part of an entire object. The format of a data primitive is constant but dependent on the specific element -- for example, a description of a line is formatted as:

LINE x1,y1,z1,x2,y2,z2,color,level,linetype,groupnum, subgr,pen

Information from each data primitive is assigned to its proper place in a two dimensional array of structures where one dimension is the group (or object) number, to which the data primitive belongs, and the remaining dimension is the number of the primitive within the group.

A function, **sort_data()**, within **read_data** scans the endpoints of each line or arc (circles do not undergo this scrutiny) and locates the left-most point in the entire group (or object). This step is important, because it determines the starting point for chaining the elements within the group in a counterclockwise direction. Alignment in a constant direction, which essentially sets the counterclockwise direction as positive, is required when working with cross products as in the group_params function.

group_params( )

Function **group_params** computes the area, perimeter, centroid, and principal axis of each object on a CADKEY pallet drawing. Lines, arcs, and circles all require different treatment as detailed in the following sections.

Area:

Any given object can be broken into triangles, circles, and partial circles. By creating vectors from the origin to the endpoints of each line segment, cross products can be used to find the area contributed by each element while moving in a counterclockwise manner about the contour of the part (Figure 5).

A linear element contributes the area of a triangle:

$$A = \frac{1}{2}\left|(\vec{r}_1 \times \vec{r}_2)\right|$$

where

$$\vec{r}_1 = (x_1)\hat{i} + (y_1)\hat{j}$$

$$\vec{r}_2 = (x_2)\hat{i} + (y_2)\hat{j}$$

and therefore

$$A = \frac{1}{2}(x_1 y_2 - x_2 y_1)$$

where (x_1, y_1) and (x_2, y_2) are endpoints of the element.

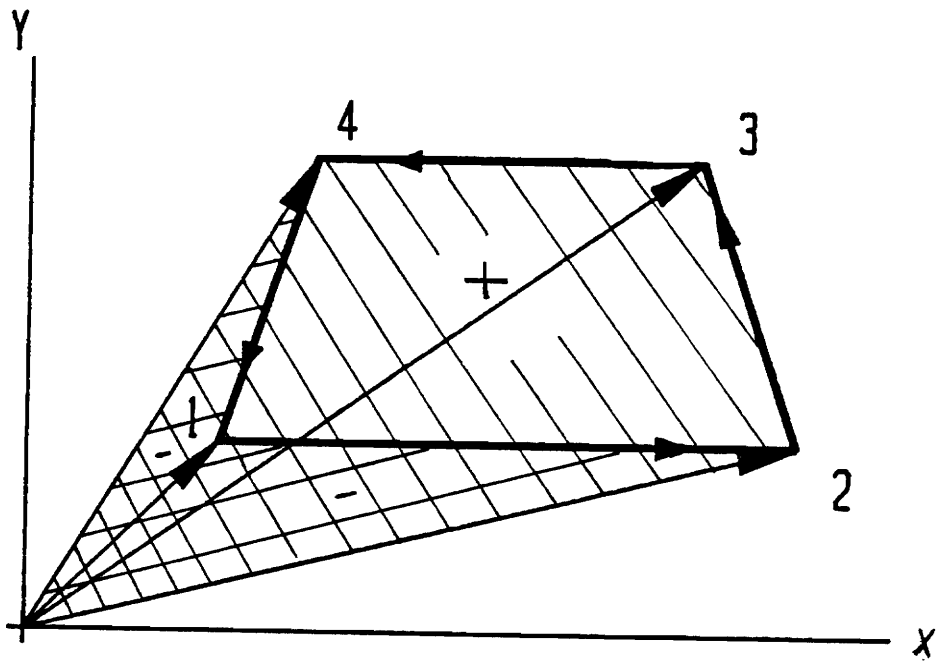


Figure 5. Finding the Area of an Object Using Cross Products: Positive and negative areas cancel each other, remaining area is positive.

The area of a circle is straightforward and given by:

$$A = \pi r^2 .$$

As seen in Figure 6, adding arcs to the object complicates the area calculation, but only to the extent of finding the area of two triangles and a pie-shaped wedge. The pie-shaped wedge has an area given by:

$$A = \alpha r^2$$

where α is one-half the difference between the starting and ending angle of the arc. Note that CADKEY assumes arcs run in a counter-clockwise direction; therefore the starting angle is always smaller than the ending angle (Figure 7).

To determine whether the wedge should be added or subtracted from the total area, we must first create a bisecting vector, $\vec{a}$, and cross it into a vector, $\vec{b}$, which runs from the center of curvature of the arc toward the endpoint of the arc (Figure 8). If the cross product is positive, the wedge is convex in relation to the contour of the object and its area should be added to the total area; if negative, the wedge is concave in relation to the contour of the object and its area should be subtracted from the total area (Figure 8).

Total area is the sum of both negative and positive areas for all triangles and wedges. Any holes in the part (represented by dashed lines in the CADKEY drawing) are assigned negative areas. In keeping with the GE Vision System convention, the holes are treated as separate entities and are not subtracted from the area of the enclosing object.

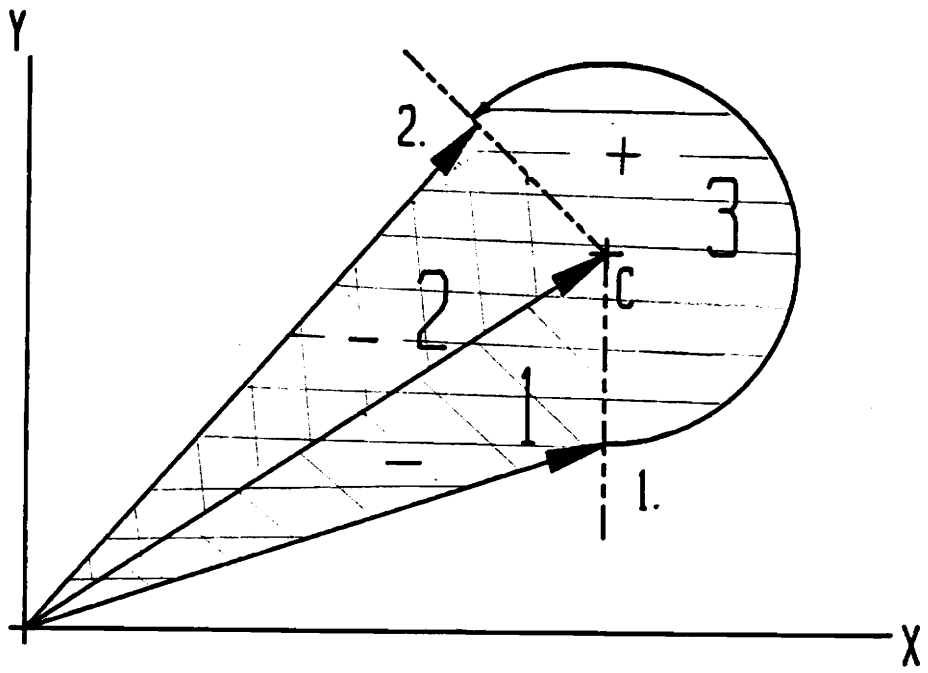


Figure 6. Finding the Area of a Circular Element: Negative areas 1 and 2 cancel all but the partial circle of area 3.

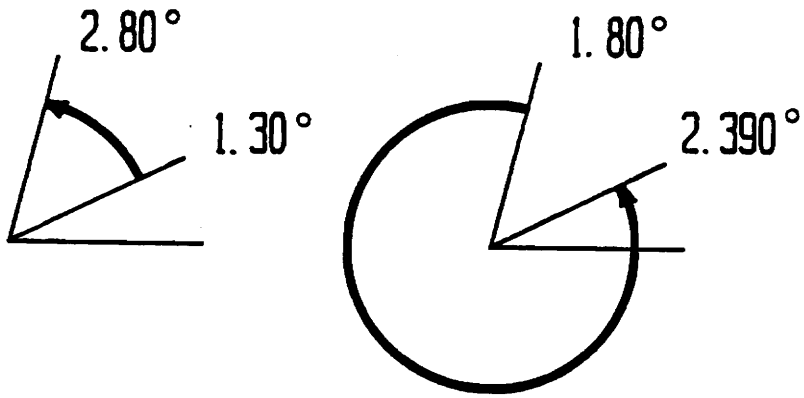


Figure 7. CADKEY Convention for Defining Beginning and Ending Arc Angles

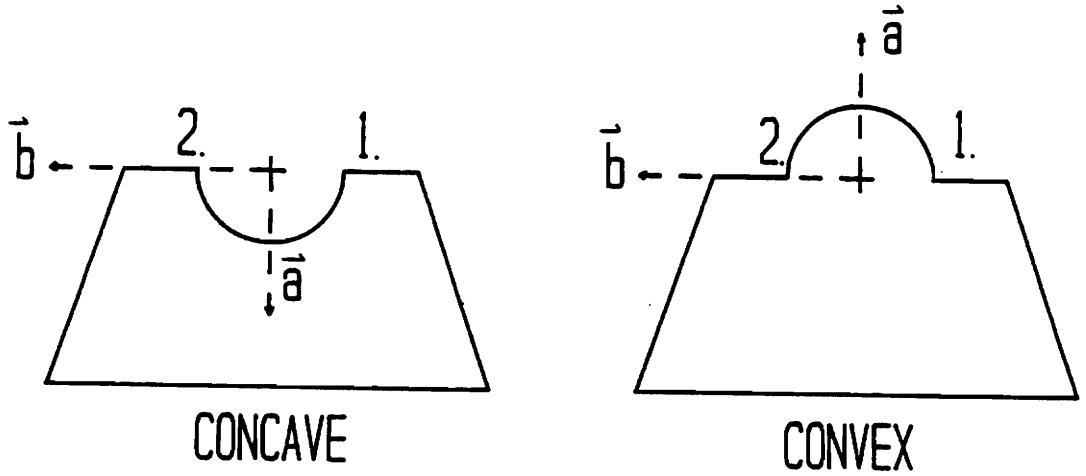


Figure 8. Determining Arc Area: Bisecting vector $\vec{a}$ is crossed into $\vec{b}$

Perimeter:

The perimeter is found by summing element lengths around the contour of the object. For a line,

$$l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

where (x_1, y_1) and (x_2, y_2) are endpoints of the line.

For an arc,

$$l = 2\alpha r$$

where α is one-half the difference between the starting and ending angles of the arc.

The perimeter of a circle is

$$2\pi r \ .$$

Centroid:

The centroid is a measurement of the concentration of mass for a solid object. It is a natural choice for determining the location of an object, because it is unique and can be calculated for objects regardless of their shape or size.

To find the centroid, **cadd.c** views objects as the same collection of triangles, circles, and wedges used in the area calculation. For linear elements, the centroid of the triangle formed by linking the two endpoints of the line with the origin is calculated. The centroid location is given by:

$$X_c = \frac{x_1 + x_2}{3}$$

$$Y_c = \frac{y_1 + y_2}{3}$$

where (x_1, y_1), (x_2, y_2) are the endpoints of the linear element.

The centroid for a circular element (arc) is a "weighted average" of the centroids of two triangles and one wedge-shaped area (Figure 9):

$$X_c = \frac{\sum_{i=1}^{3} X_i A_i}{A}$$

$$Y_c = \frac{\sum_{i=1}^{3} Y_i A_i}{A}$$

Here X_i and Y_i are centroid locations of the three individual areas.

For the wedge-shaped element:

$$X_c = X + \left(\frac{(2)(r) \sin \alpha}{3\alpha} \right)(\cos \theta)$$

$$Y_c = Y + \left(\frac{(2)(r) \sin \alpha}{3\alpha} \right)(\sin \theta)$$

where (X,Y) is the center of curvature for the arc, and α, θ are shown in Figure 9.

The last element type, a circle, has a centroid coincident with its center point.

When the centroids for each of the individual elements are known, the overall value is found by the formulas:

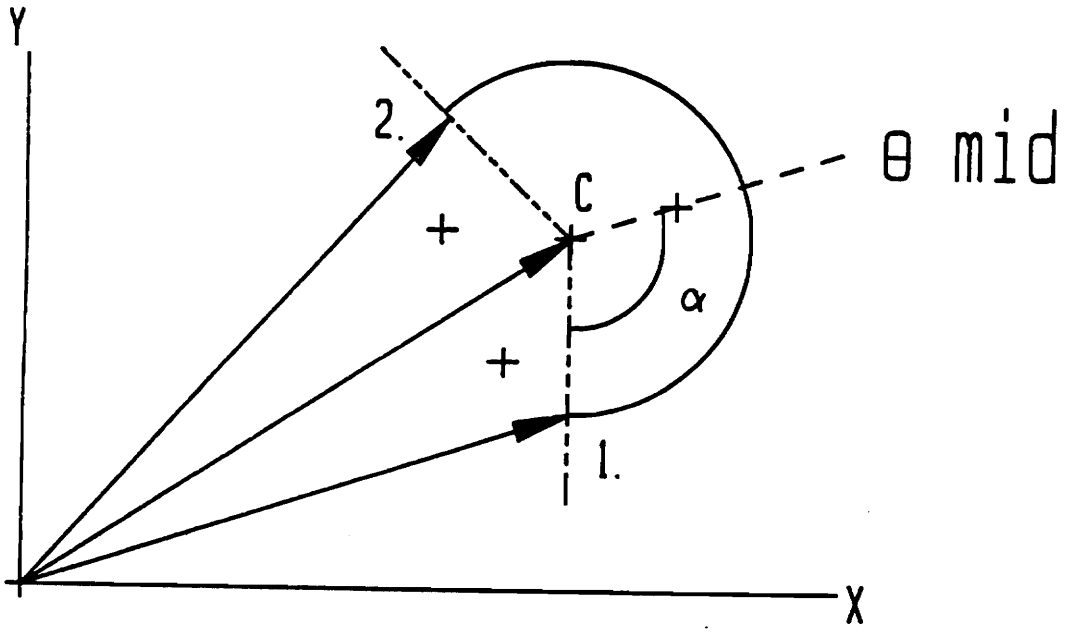


Figure 9. Finding the Centroid of a Partial Circle: Crosses mark the centroid locations of the three individual areas.

$$X_c = \frac{\sum_{i=1}^{\#Elements} (X_{ci})(ElementArea)}{Total\ Area}$$

$$Y_c = \frac{\sum_{i=1}^{\#Elements} (Y_{ci})(ElementArea)}{Total\ Area} \quad .$$

Principal Angle:

The principal angle is a measurement of the angle between the horizontal axis and the maximum principal body axis of an object. Finding the principal angle requires first finding the second moments of inertia, I_{xx}, I_{yy}, and I_{xy}, for each object on the pallet. Once these values are known, the principal angle, β, is found as shown in Figure 10:

$$\beta = \frac{1}{2} \tan^{-1}\left(\frac{I_{xy}}{I_{xx} - \frac{I_{xx} + I_{yy}}{2}} \right) = \frac{1}{2} \tan^{-1}\left(\frac{2I_{xy}}{I_{xx} - I_{yy}} \right) \quad .$$

For a circle, I_{xx}, I_{yy}, and I_{xy} are simply:

$$I_{xx} = \frac{\pi}{4} r^4$$

$$I_{yy} = \frac{\pi}{4} r^4$$

$$I_{xy} = 0 \quad .$$

For a line, look at the triangle formed by connecting vectors from the origin to the endpoints of the line. Such a triangle can take two forms: one in which the upper vector has a larger X value, and the second in which the lower vector takes on a larger X value (Figure 11). A four-sided polygon

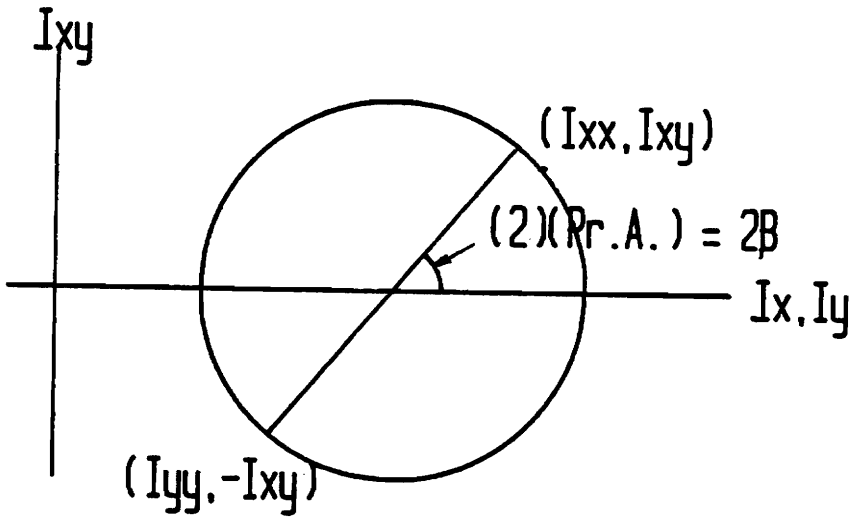


Figure 10. Mohr's Circle for Second Moments of Areas

can be used to generalize both cases (Figure 12). Note that, for the first case, the slopes of Y_1 and Y_2 are equal; for the second case, the slopes of Y_3 and Y_4 are equal. Starting with the general formulas:

$$I_{xx} = \int_A Y^2 dA$$

$$I_{yy} = \int_A X^2 dA$$

$$I_{xy} = \int_A XY dA$$

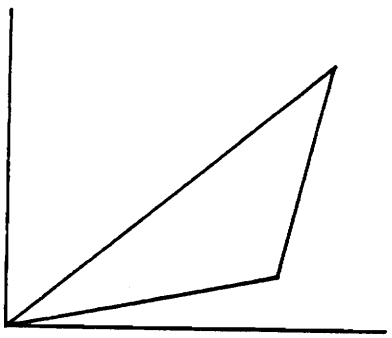
and carrying out the operations described in Appendix D, the following are obtained:

$$I_{xx} = \frac{1}{12}\{(A^3 - B^3)x_{mid}^4 + (D^3 - F^3)(x_{tip}^4 - x_{mid}^4) + 4(CD^2 - EF^2)(x_{tip}^3 - x_{mid}^3) + 6(C^2D - E^2F)(x_{tip}^2 - x_{mid}^2) + 4(C^3 - E^3)(x_{tip} - x_{mid})\}$$

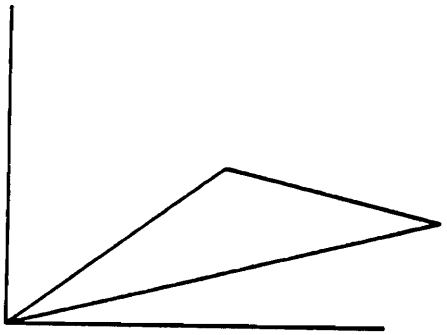
$$I_{yy} = \frac{1}{12}\{3(A - B)x_{mid}^4 + 3(D - F)(x_{tip}^4 - x_{mid}^4) + 4(C - E)(x_{tip}^3 - x_{mid}^3)\}$$

$$I_{xy} = \frac{1}{24}\{3(A^2 - B^2)x_{mid}^4 + 3(D^2 - F^2)(x_{tip}^4 - x_{mid}^4) + 8(CD - EF)(x_{tip}^3 - x_{mid}^3) + 6(C^2 - E^2)(x_{tip}^2 - x_{mid}^2)\}$$

The variable definitions and derivation of these formulas are in Appendix D.



Case 1



Case 2

Figure 11. Two Cases for a Triangle

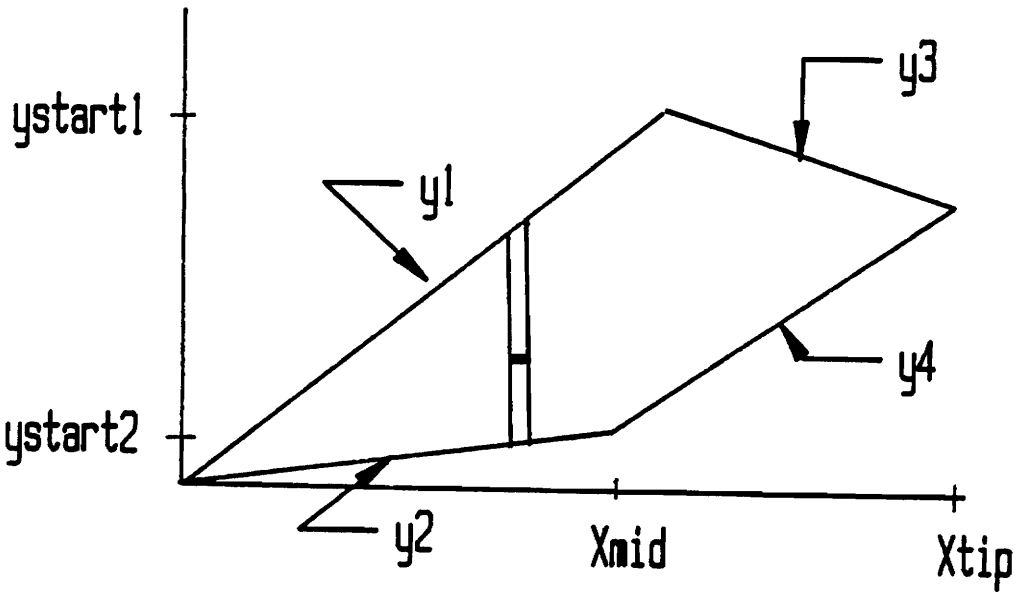


Figure 12. Generalizing a Triangle With a Four-Sided Polygon

An element-of-type arc is again treated as two triangular areas and a wedge-shaped area. The second moments of the wedge-shaped area are most easily found at the center of curvature and then translated to the centroid and the origin by the parallel axis theorem.

Starting with the variables shown in Figure 13 and the general formulas:

$$I_{xx} = \int_A \tilde{Y}^2 dA$$

$$I_{yy} = \int_A \tilde{X}^2 dA$$

$$I_{xy} = \int_A \tilde{X}\tilde{Y} dA$$

where $(\tilde{X}, \tilde{Y}) = (\rho \cos \theta_{mid}, \rho \sin \theta_{mid})$, ρ is the radial distance to dA, and α is one-half the distance between the beginning and ending angles of the arc:

$$I_{xx} = \frac{r^4}{8}[2\alpha - \sin(2\alpha)\cos(2\theta_{mid})] + \alpha r^2 \left[y_c^2 - \frac{4r^2 \sin^2\alpha \sin^2\theta_{mid}}{9\alpha^2} \right]$$

$$I_{yy} = \frac{r^4}{8}[2\alpha + \sin(2\alpha)\cos(2\theta_{mid})] + \alpha r^2 \left[x_c^2 - \frac{4r^2 \sin^2\alpha \cos^2\theta_{mid}}{9\alpha^2} \right]$$

$$I_{xy} = \frac{-r^4}{8}[\sin(2\theta_{mid})\sin(2\alpha)] + \alpha r^2 \left[x_c y_c - \frac{4r^2 \sin^2\alpha}{9\alpha^2} \cos\theta_{mid} \sin\theta_{mid} \right] .$$

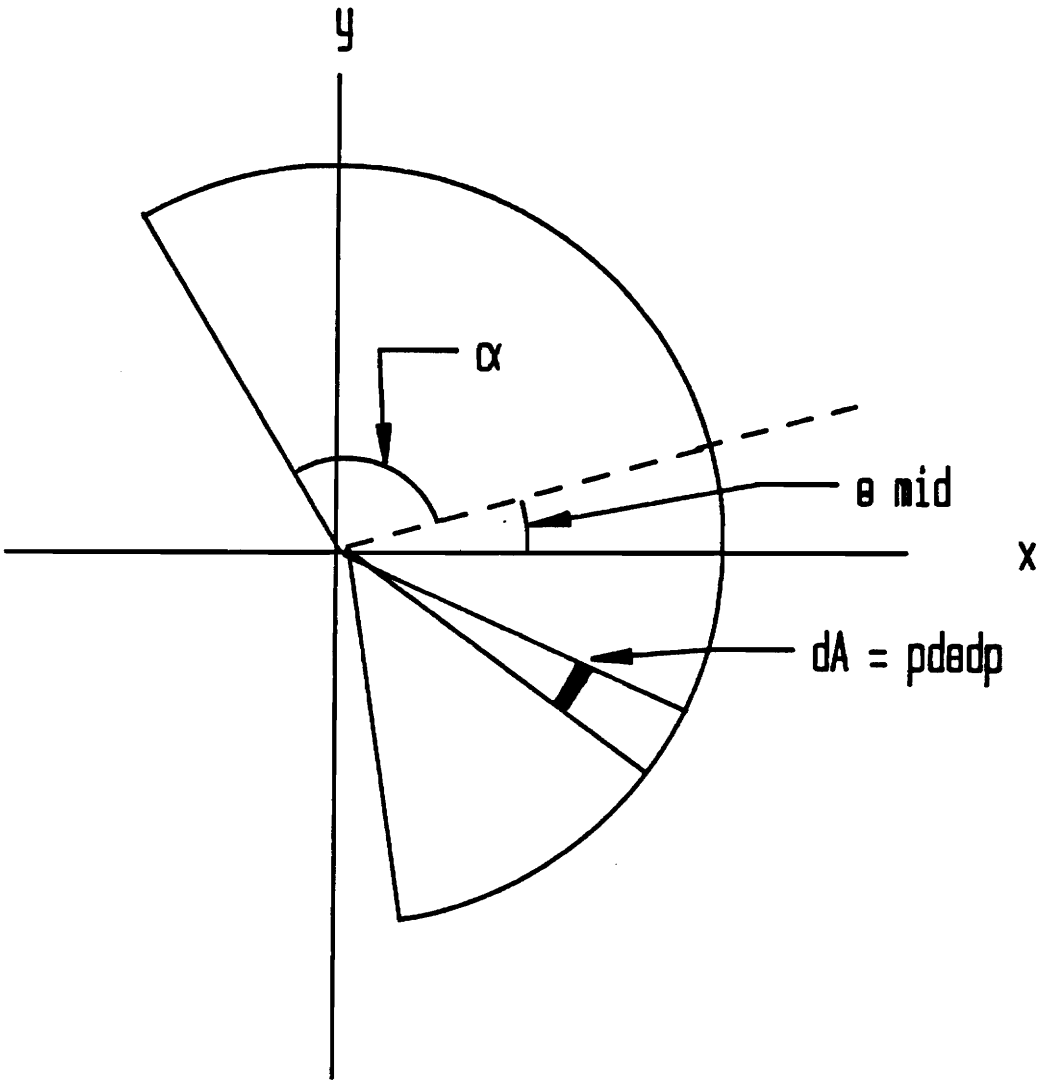


Figure 13. Finding Moments of Inertia for a Partial Circle

The derivation of these formulas is contained in Appendix E.

Knowing the second moments for the individual elements, it is easy to find the totals:

$$I_{xx} = \sum_{i=1}^{n} I_{xx}$$

$$I_{yy} = \sum_{i=1}^{n} I_{yy}$$

$$I_{xy} = \sum_{i=1}^{n} I_{xy} \quad .$$

Because the GE Vision System calculates second moments about the centroid of the object, and not about the origin as calculated here, the second moments must be translated to the centroid using the parallel axis theorem. This theorem states that "the moment of inertia of an area with respect to a noncentroidal axis is equal to the moment of inertia of the area with respect to the parallel centroidal axis plus the product of the area and square of the distance between the axes" (Cheng, 1985, pp. 262), i.e.

$$I_{X0} = I_{Xc} + Ay_c^2$$

$$I_{Y0} = I_{Yc} + Ax_c^2 \quad .$$

Here x_c and y_c are the distances between the origin and centroid along the x and y axes, respectively.

Now that all of the model parameters have been calculated, they must be formatted so that they may be used by the vision system. The function **data_out( )**, explained in the next section, performs this task.

data_out( )

The last function in **cadd.c** creates a file containing the descriptive pallet parameters in a format which can be used by the GE Vision System. If **cadd.c** were to be used with another vision system, this function could easily be modified to accommodate a new format.

As shown in the example below, a **cadd.c** output file for Figure 4 has VPL statements which "fit" into the COMPARISON@ program. The first numbered statement defines the number of objects on the pallet drawing, while the second numbered statement sends an array of five descriptive parameters for each of those objects:

```
30 NOITEMS = 3 < CR >
100 DATA(P(2,1),100,200,125,500,1800,100,200,675,500,1800, 225,175,400,600,900)
```

Because the vision system only accepts integer input, the parameters are multiplied by one hundred and truncated. This operation maintains precision of the area, centroid location, and perimeter to one-hundredth of an inch. The sole exception to this operation is for the principal angle: to adopt the same clockwise convention followed by the GE Vision System, this parameter is first subtracted from 180 degrees and then multiplied by ten.

Two additional functions, **print_groups()** and **print_props()**, display information about the elements that make up each object, and repeat the descriptive parameters to the monitor. These functions have no bearing on the output file but, because their format is easy to read, they are useful for checking the contents of the output file.

At this point, the pallet model information is calculated and made available for the GE Vision System to use. Chapter 5 describes the next step in the integration process: comparing the model information to the pallet image obtained by the vision system.

Chapter 5: Vision System Programs

The third and fourth objectives of this research were to create vision system programs to extract descriptive parameters from the actual pallet image and to compare them to those parameters extracted by the **cadd.c** program. Two VPL programs were developed to perform these vision system tasks: the COMPARISON@ program, which directs the vision system camera to view the pallet and calculate object parameters, and the CALIB@ program, which provides the pixel/inch ratio to convert **cadd.c** parameters to their equivalent pixel units. Both programs are listed in Appendix F.

COMPARISON@

COMPARISON@ starts by initializing the GE Vision System to view as black any area with a threshold brightness level lower than 90 (on a scale from 1 to 255, the brightest). The vision system considers any object with a low brightness level to be a part. Choosing the threshold level is a matter of trial and error: the right level returns a solid image to the output monitor.

The VPL language has a number of built-in functions or "feature extraction commands" which automatically generate information about the image seen by the GE Vision System. The COMPARISON@ program calls one such function, QTY.ITEMS, to find the number of objects in the viewing field.

To find QTY.ITEMS, the GE Vision System scans the input scene from left to right and from top to bottom. When the system encounters a pixel with a low brightness level, it starts "building" an object. Continuing to scan, the vision system reaches a point where there are no longer any low level pixels connected to the object. If this completed object is larger than the the preset noise level, the vision system assigns it an item number.

With this scanning convention, the vision system always views the registration object in the upper left-hand corner of a pallet as item number one. This fact is important, because the centroid location and principal angle of item number one are reference point zero. Of course, the registration object is irrelevant when one is finding an object's area and perimeter: these parameters remain constant regardless of location.

Two lines in the COMPARISON@ program contain data from a **cadd.c** output file: line 30 defines the number of items on the model pallet, and line 100 defines an array of area, $x_{centroid}$, $y_{centroid}$, and principal angle for each of those items. COMPARISON@ receives this information via the communications program which discussed in Chapter 6.

The first comparison in the program is between the number of objects on the model pallet and the number of objects viewed by the vision system (Figure 14). Obviously, the pallet is not correct if these numbers differ. If they are equal, the COMPARISON@ program begins to compare attributes of the individual objects.

If an object on the pallet checks favorably against all five model parameters, the program increments a counter. The pallet is correct if the count equals the number of items on the actual pallet.

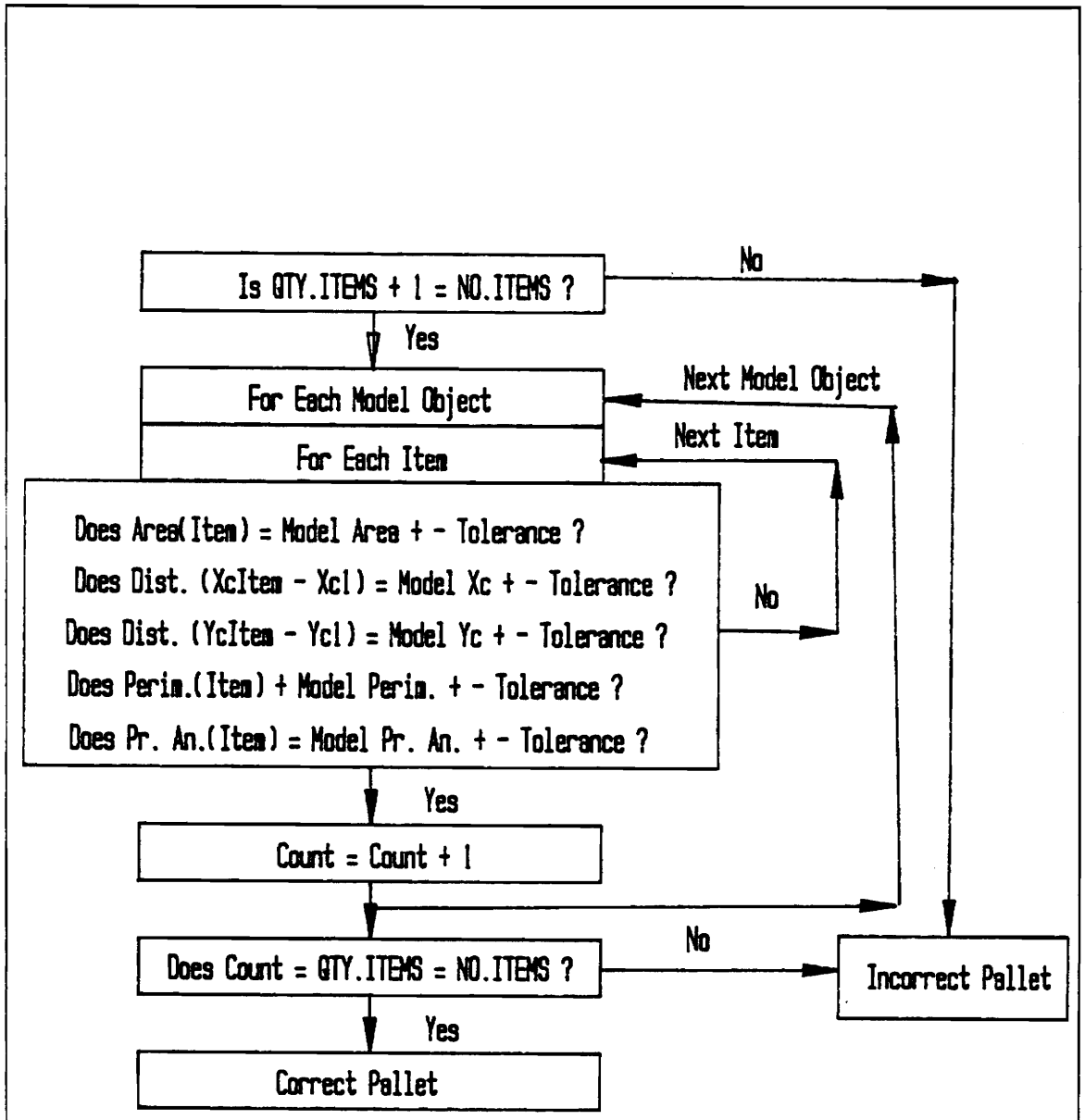


Figure 14. Flow Chart for the COMPARISON@ Program

The comparison process is not quite as simple as comparing two numbers: both the vision system parameters and the model parameters need to be transformed to equivalent units before they can be compared. In the case of the vision system parameters, this transformation entails subtracting X and Y centroid values and principal angle of the registration object from the corresponding values of each object the vision system views. These relative values now conform to CADKEY model values which have a zero reference point.

Cadd.c defines the model parameters in units of inches, inches squared, or in the case of principal angle, in degrees times ten. The principal angle is the only parameter which is defined in the same manner by both systems and thus can be compared directly to its vision system counterpart. The other model parameters are multiplied by a pixel/inch ratio to convert them to pixel units. The CALIB@ program described in the next section provides this ratio for the COMPARISON@ program.

One problem in dealing with the vision system is that it may return a range of values for a parameter for a given part. This effect is mostly due to the location and orientation of an object. The vision system may or may not include partially covered pixels in its calculations; that is why this effect is especially apparent on diagonal object borders.

The COMPARISON@ program handles this problem by designating a range of "correct" values which a parameter may take on. To determine the range, objects with known parameters were tested. The test objects, a 1" diameter circle, a 2" diameter circle, a 1" by 2" rectangle and a 1" square, were placed under the vision system camera at various locations and orientations. Thirty values were obtained for each object for area, perimeter, and length. To test the principal angle, an entire sheet of objects with equal orientations (Figure 15) was placed under the camera and rotated incrementally through 180 degrees. This operation was repeated thirty times. The largest and smallest principal angles were recorded for each trial.

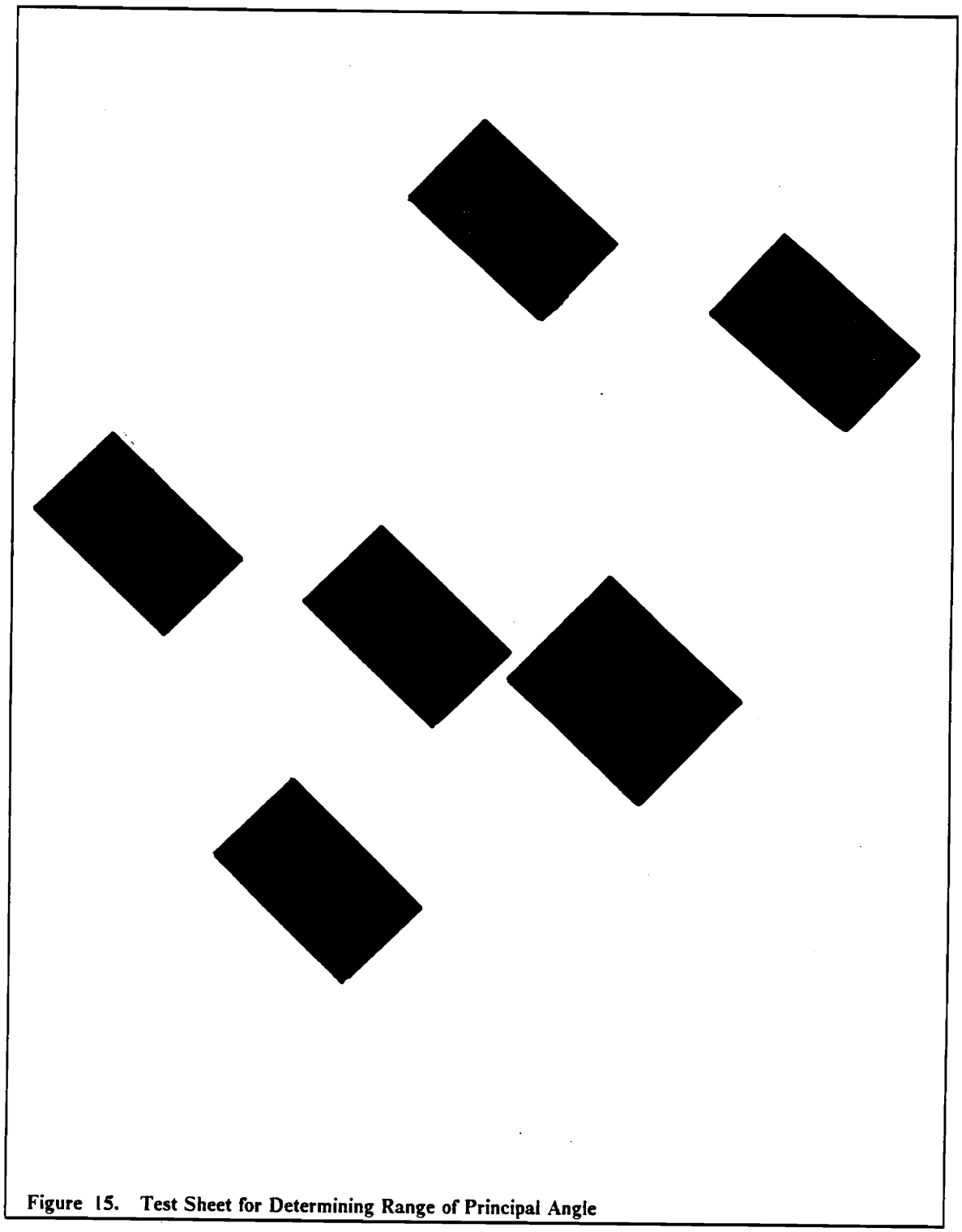


Figure 15. Test Sheet for Determining Range of Principal Angle

Table 1 lists the largest and smallest area values recorded for each test piece. The "real" value is the area of the test piece based on 5550 quarter pixels per square inch (the calibrated ratio). Dividing the real value by a "divisor" yields the upper and lower limits shown. Sometimes the real value is not included in the range, because glare or object placement causes the vision system to see the object as much smaller (or larger) than it really is. The COMPARISON@ program employs the largest and smallest divisors to set the range of values the image area may take on. For example,

$$\frac{\text{Model Object Area}}{1180} \leq \text{Image Area} \leq \frac{\text{Model Object Area}}{1040}$$

Tables 2 and 3 list the real values, limits, and divisors for perimeter and centroid (measured by a length between two points). The tolerances for principal angle are simply plus or minus the largest difference between the two recorded angles for each trial -- plus or minus 40 (4 degrees).

CALIB@

The CALIB@ Program is an essential part of the comparison process. For any given camera height and setting, the program returns a conversion factor to transform inches to pixels. This result, RATIO%, is a global variable which resides in nonvolatile memory of the vision system. As mentioned in an earlier chapter, CALIB@ should be run whenever the camera settings change.

Figure 16 illustrates a calibration pallet with a one inch diameter circle; the actual calibration pallet is twice this size. The vision system views the two inch diameter circle, calculates its width through the centroid, and divides by two. Because the vision system can only work with integers, CALIB@ actually multiplies the width value by ten to ensure a whole number result.

Table 1. Image Area Values for Four Test Parts

Object	Real Value	Upper Limit (Square Pixels)	Divisor	Lower Limit	Divisor
1" Circle	4329	4162	1040	3957	1094
1" Square	5550	5240	1059	4924	1127
2" Circle	17427	16708	1043	15756	1106
Rectangle	11100	10211	1087	9406	1180

Table 2. Image Perimeter Values for Four Test Parts

Object	Real Value	Upper Limit (Square	Divisor Pixels)	Lower Limit	Divisor
1" Circle	233	273	983	215	1080
1" Square	298	304	980	286	1121
2" Circle	467	492	949	427	1092
Rectangle	447	463	965	405	1103

Table 3. Image Centroid Values for Four Test Parts

Object	Real Value	Upper Limit (Square Pixels)	Divisor	Lower Limit	Divisor
1" Circle	74.5	74	1006	69	1065
1" Square	74.5	74	1006	69	1065
2" Circle	149	165	900	140	1057
Rectangle	149	161	920	142	1045

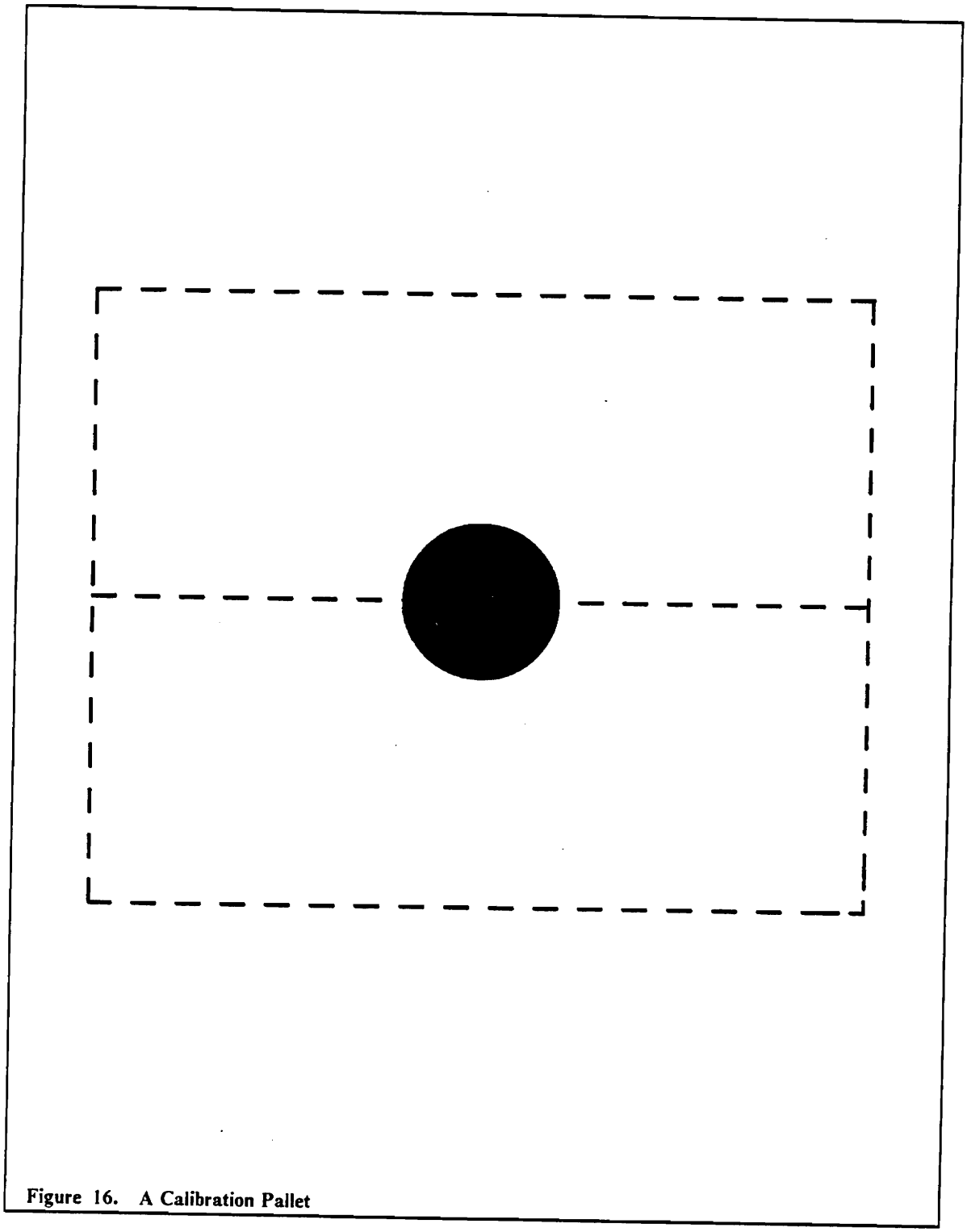


Figure 16. A Calibration Pallet

Cadd.c, COMPARISON@, and CALIB@ need to work together to determine pallet status, but they need a method to share parameter information. The next chapter discusses the missing link: a method to communicate program results between the host computer and vision system programs.

Chapter 6: Vision System Integration

The last objective of this research was to develop hardware and software links to enable the host controller to communicate with the vision system.

The hardware link is straightforward: a cable (with pins 2 and 3 reversed, 7 running straight through) runs from the host computer's RS-232C communications port (COM1) to the RS-232C serial interface of the vision system. The second communications port (COM2) of the host computer could be used in place of COM1 with a slight modification to the communications software as detailed below.

The software link is actually a set of three separate executable files: a terminal program, a program to run the vision system COMPARISON@ program, and a program to run the vision system CALIB@ program. These programs share the same functions, and differ only by the structure of their main programs.

All of the programs configure COM1 to accept and transmit data as eight-bit characters with one stop bit and no parity. The baud rate is set at 9600. Both the COM port and baud rates are parameters of a function **Setserial** which is called from the main program. The program could easily

be modified to accommodate COM2 or other baud rates by placing the new parameters into **Setserial** and recompiling.

Function **initserial** clears the input buffer (characters waiting to be read) and sends a signal to the vision system that the host controller is ready to send data. Function **SerialOut** outputs a character to COM1 after ensuring that the output register is clear and that the vision system is ready to receive the character.

A function **getccb** constantly monitors the input buffer to see if a character has arrived from the vision system. If a character arrives while another character is still waiting to be read, a software interupt prevents the character from being overwritten.

Serial.c, the terminal program, monitors the keyboard for any incoming characters. Providing that a key has been pressed, and it is not the escape character which exits the program, the character will be sent to COM1 and hence the vision system. If a character is waiting to be read, it will be sent to the monitor.

Calibcom.c runs the vision system CALIB@ program. This program sends the characters "RUN CALIB@" to the vision system character by character. The input or receiving buffer is not monitored, because the results of the calibration program are stored and used by the vision system only.

The last program, **Comparecom.c**, controls execution of the vision system COMPARISON@ program. This program requires one argument: the name of the file created by **cadd.c** which corresponds to the pallet being viewed by the vision system. With this file information, the vision system compares the model and image data.

To incorporate the **cadd.c** generated file into the COMPARISON@ program, **Comparecom.c** sends a command to "EDIT COMPARISON@", which is then followed by a stream of characters from the **cadd.c** created file. When editing is complete, **Comparecom.c** sends the command "RUN COMPARISON@." The program output from the vision system is sent to the monitor of the host

computer. The last character output by COMPARISON@ is always a "?" which terminates **Comparecom.c**.

If for any reason there is an error condition, an error message will appear on the monitor. At this point, an operator will need to evaluate the error, take corrective action, and terminate the program manually by entering a "?" from the keyboard.

Now that a link has been established between the host computer and vision system, and in effect, between the the model and image parameters, the methodology section of this thesis is complete. The final chapter presents results from testing the system of programs along with discussion of results and future enhancements.

Chapter 7: Results and Conclusions

The last four chapters have detailed a method that was developed for this research to integrate a GE Optomation II Vision System into a computer integrated manufacturing system. Chapter 4 presented a means to extract parameter values from CADKEY model pallets. Chapter 5 described the vision system programs which calculate the corresponding image parameters and enable the two sets of parameters to be compared on the same unit basis. Chapter 5 also outlined a process to set a range of variability for the image parameters based on their actual value. Chapter 6 discussed a method to control the vision system via a communications program. The last step in this research was to verify that the vision system does indeed identify objects based on CADKEY model data.

A preliminary test of the system involved placing two different correct pallets, an unmachined robot working pallet and an assembled NC working pallet, under the vision system camera. The system did not consistently label the pallets as correct: however, the two pallets did test correctly when the objects were located at the middle or right-hand side of the input scene.

Further investigation revealed that the vision system calculated much smaller parameter values for objects located at the left-hand side of the scene -- even more so for objects located in the upper left-hand side of the scene. The calibration process should have prevented this problem, but because the calibration pallet is generally placed at the center of the scene, the resulting RATIO% is

correct for objects in that area. When the calibration pallet is placed at the left-hand side of the scene, the pallets with objects at the middle and right-hand side of the scene test poorly. One solution to this problem would be to widen the range of values the image values may take on. A drawback to this solution is that the tolerances may be so large that incorrect parts could easily be labeled as correct. It would also be possible to separate the areas on the input scene which return significantly higher (or lower) parameter values, and calibrate these areas independently. The COMPARISON@ program would then be much more complex: a different set of tolerances would need to be applied to objects according to their location. A new sensor in the camera may eliminate the need for any of these solutions.

It was apparent from the preliminary testing that the color and surface of the objects and background affect image parameter values. To determine the first set of image parameter ranges, the trials were run using black tagboard objects against a matte gray background. The objects were soon replaced with matte black objects, however, because glare from the overhead lighting appeared as holes to the vision system. A white background also produced glare which changed parameter values. The effect was large enough to necessitate running new trials with the white surface used as a background for the test pallets. Future vision system operators should choose background surfaces carefully, calibrate using the same background, or control lighting conditions.

The GE Optomation II System is extremely sensitive to lighting conditions. Ambient lighting may cause glare and/or shadows which in turn may affect image parameter values. Setting the proper threshold level is of utmost importance to reduce these effects. Other alternatives are to create an enclosed lighting system which surrounds the camera and directs light at the pallet from all directions, or to use backlighting of the pallet.

In order for many of the system pallets to fit into the camera's field of view, the reference object must be located in the extreme upper left-hand corner of the scene. Because of the aforementioned problems with locating objects in this area, four new pallets were generated for testing with objects grouped at the left-hand side of the scene.

Figure 17 shows the first pallet, which contains only one part. A duplicate copy of the second object was cut out of black paper so that it could be moved freely about the true centroid location for the second object. The object could be moved approximately one-half inch about the X centroid location before the COMPARISON@ program labeled the pallet as incorrect. In the Y direction, the variability was somewhat smaller, approximately one-quarter inch about the centroid. The difference may be due to the larger Y location values; a larger model parameter produces a wider range of values which the image parameter may take on. One suggestion to alleviate this problem would be to create pallets with shallow indentations to hold the objects in place. The COMPARISON@ program would then play the role of checking for missing and extra objects and objects that are grossly misaligned rather than checking for absolute alignment.

Figure 18 illustrates three objects with unique principal angles. The COMPARISON@ program recognized the rectangular objects, but rejected the pallet on the basis of principal angle of the circle. Up to this point, all of the test pallets have contained objects with definite principal axes. A circle is radially symmetric, i.e.,

$$\frac{I_{\max}}{I_{\min}} = 1$$

and therefore the vision system has no means to define the principal axis save by default. The **cadd.c** program does default to zero in this situation. The pallet shown in Figure 19 was employed to test the vision system's handling of circular objects. In ten trials, the vision system returned seemingly random values for principal angle.

Results of the testing show that the integrated system is feasible for a limited number of pallet configurations. Although the system works well for pallets such as the one illustrated in Figure 20, where all objects have a definite principal axis, correct pallets containing symmetrical objects (including squares) are not generally labeled as correct. Clearly, principal angle is not an optimal choice for an object parameter, and for any future use of the system another method to verify ro-

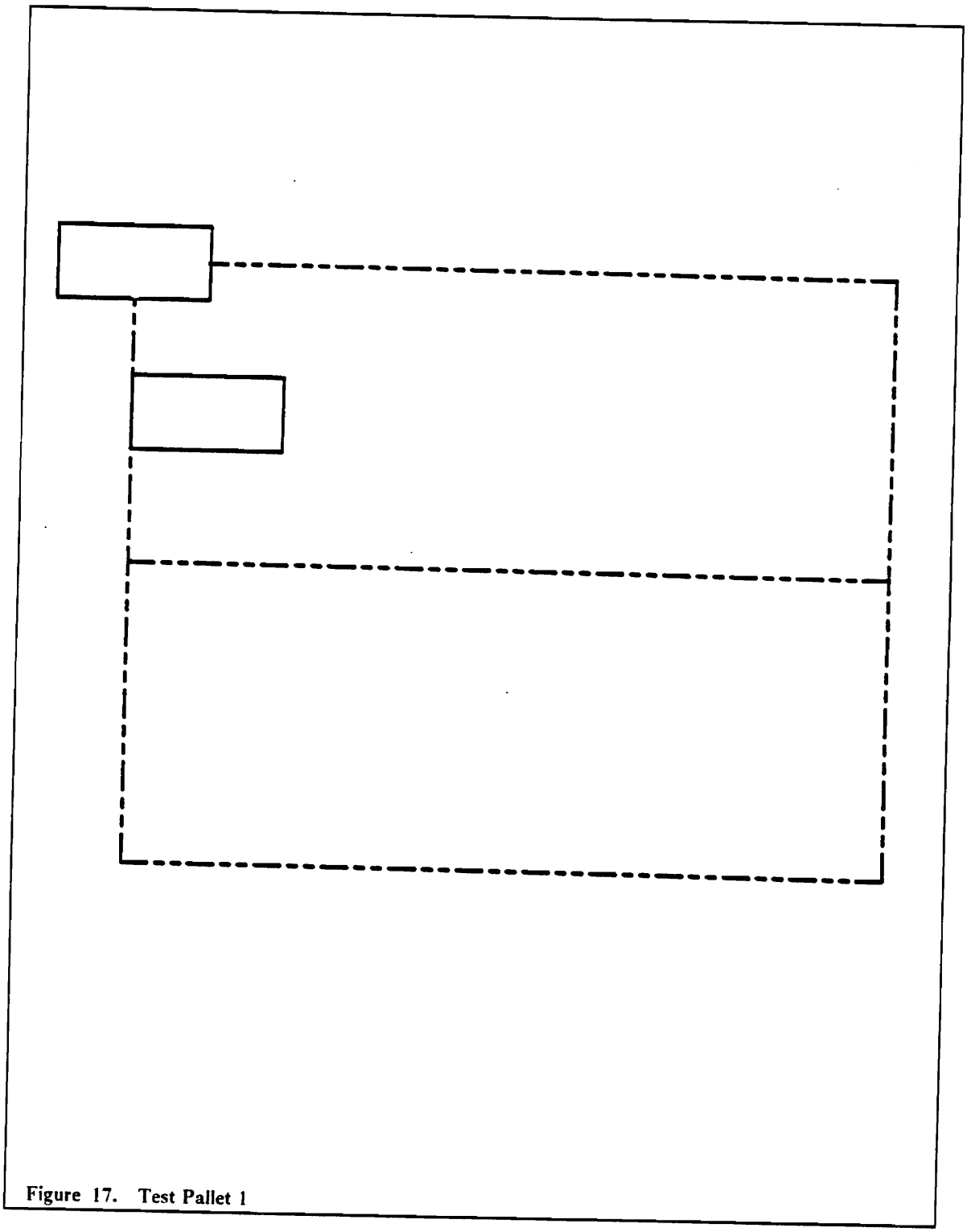


Figure 17. Test Pallet 1

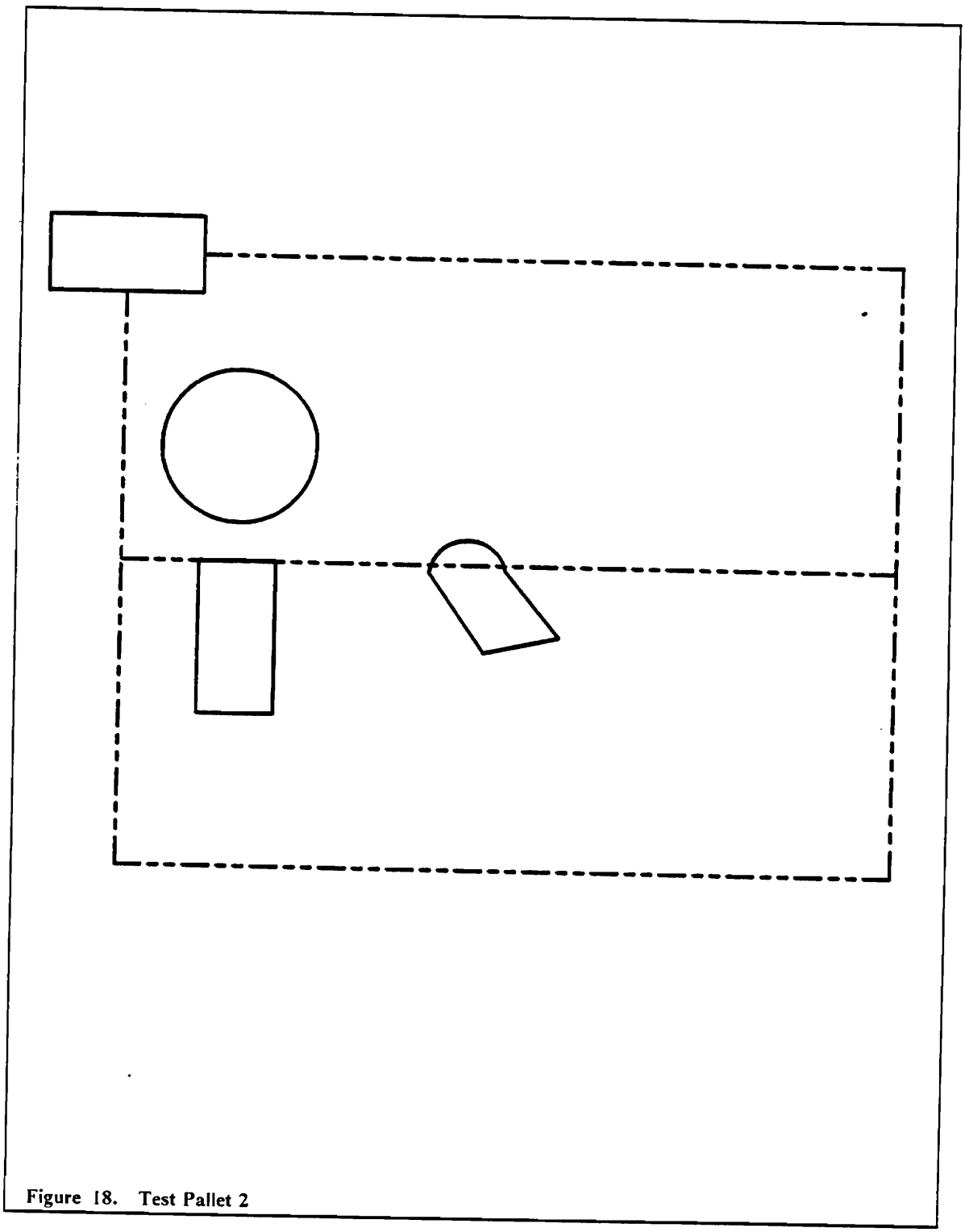


Figure 18. Test Pallet 2

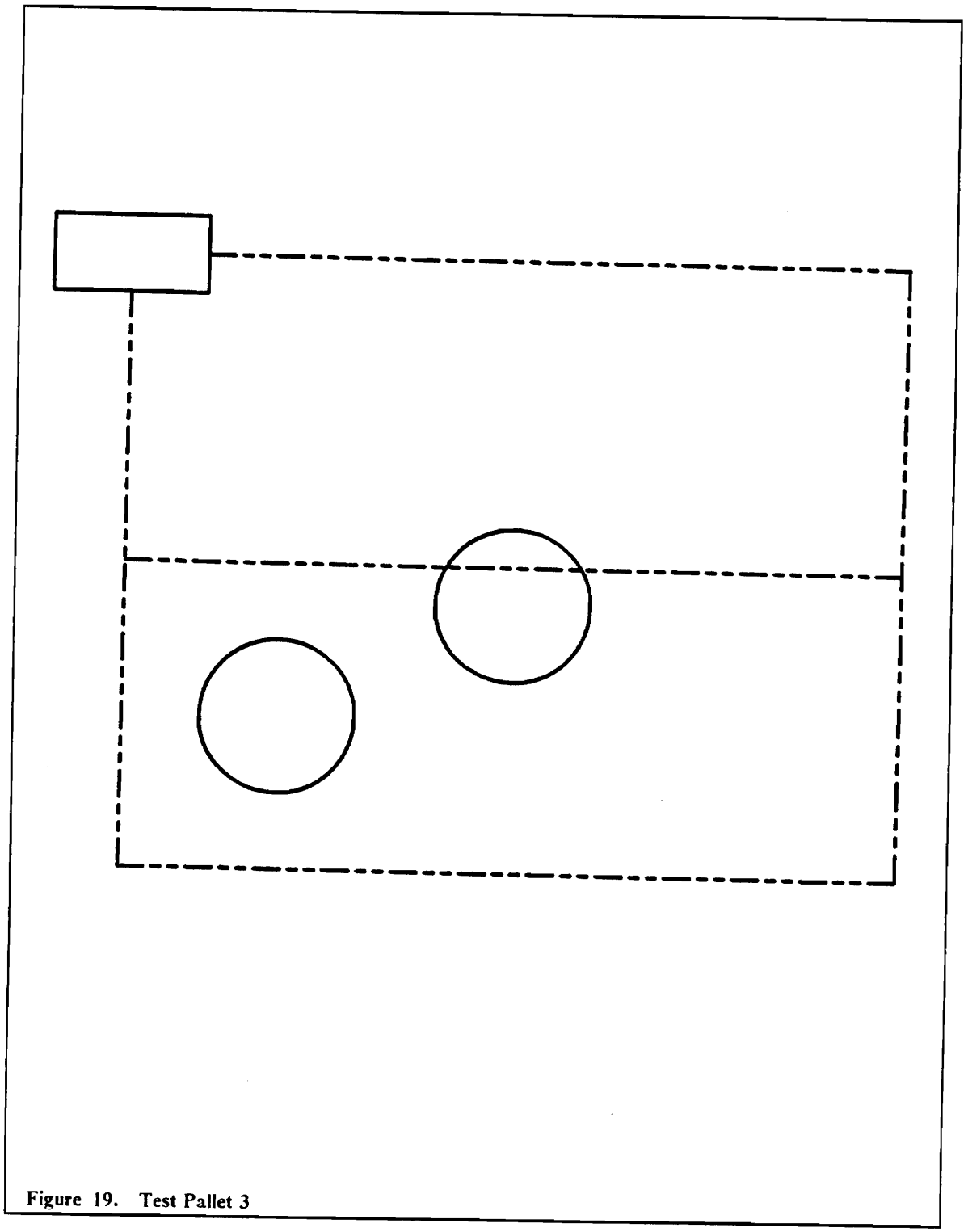


Figure 19. Test Pallet 3

tation is needed. One possibility would be to use the minimum and maximum bounding box functions, or to run tangent lines from the object to determine an intersection point.

One must also consider another limitation of the methodology: that all the testing was performed with two-dimensional binary images. Adding depth to the objects compounds the difficulty with shadow and perspective -- not all objects can be placed directly under the camera for viewing. It would be interesting to study whether grey scale capabilities would help or hinder these problems.

Future extensions to this research might include calibrating the camera at different heights, because parameter variability may not be a function of the actual value alone. A step function may be a much better alternative then the straight percentage used in this thesis.

Along the same line, it would be interesting to examine whether the tolerances set up for the image parameters affect the ability of the robot to perform its job. Will an object that passes vision system inspection cause problems for the robot if the object is slightly undersized or rotated? What are the acceptable limits?

Although somewhat constrained by the functions available on the GE Vision System, future research might center on adding more descriptive parameters to the **cadd.c** program. The present system devotes only one parameter to verifying object location, one to verifying object orientation, and two to verifying the form of the object itself.

Presently the **cadd.c** progrm deals with objects composed of lines and circular arcs. A data primitive for ellipses is available through the CADKEY program, and it would be a challenging task to include this element type in the **cadd.c** program. Another enhancement to **cadd.c** would be the capability to update CADL files for objects that have been removed from the pallet.

In conclusion, this research has presented a methodology for integrating vision capabilities into a computer integrated manufacturing system. Although the goal of integration has been achieved, limitations in the methodology and hardware exist which restrict the pallet configurations that the

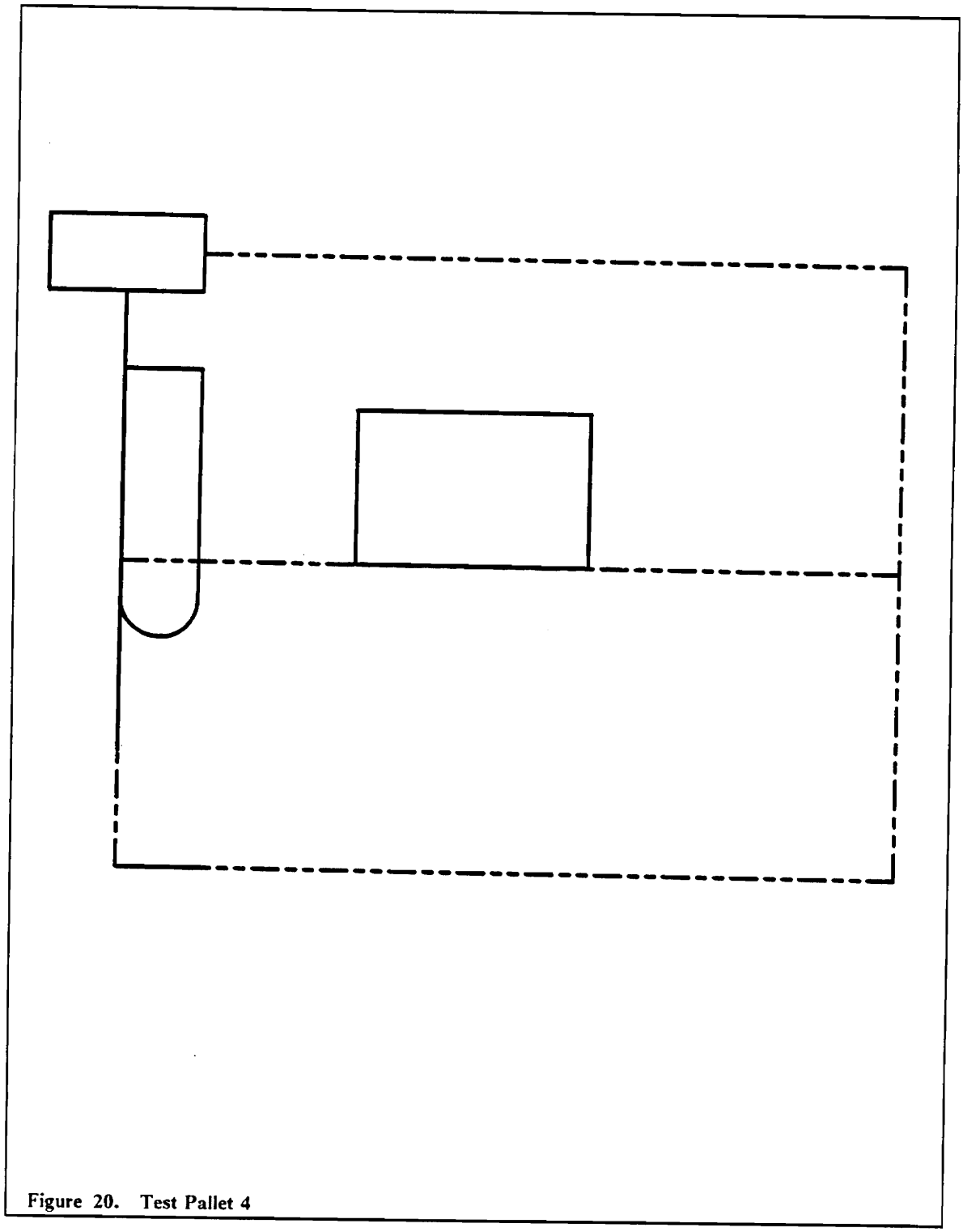


Figure 20. Test Pallet 4

system can identify. This work has shown that, in spite of differences in communications protocol and control languages, it is possible to combine equipment from different manufacturers into a cohesive system.

References

Aleksander, I., T.J. Stoneham, and B.A. Wilkie, Computer Vision Systems for Industry: Comparisions, in *Artificial Vision for Robots*, ed. I. Aleksander, Kogan Page: London, 1983.

Andrews, H.C., *Introduction to Mathematical Techniques in Pattern Recognition*, John Wiley and Sons, Inc.: New York, 1972.

Asfahl, C.R., *Robots and Manufacturing Automation*, John Wiley and Sons, Inc.: New York, 1985.

Barkakati, N., *The Waite Group's Microsoft C Bible*, Howard W. Sams and Company: Indianapolis, Indiana, 1988.

Besant, C.B., and C.W.K. Lui, *Computer-Aided Design and Manufacture*, 3rd ed.,Ellis Horwood Limited: Chichester, England, 1986.

Cheng, Fa-Hwa, *Statics and Strength of Materials*, Macmillan Publishing Company: New York, 1985.

Chin, R.T., Survey, Automated Visual Inspection: 1981 to 1987, *Computer Vision, Graphics, and Image Processing*, **41**, 1988, pp. 346-381.

Gaglio, S., R. Morasso, and V. Tagliasco, Syntactic Techniques in Scene Analysis, in *Artificial Vision for Robots*, ed. I. Aleksander, Kogan Page: London, 1983.

Gleason, G. J., and G. J. Agin, A Modular Vision System for Sensor-Controlled Manipulation and Inspection, *SRI International Technical Note 178*, February 1979.

Hindin, H.J., Algorithms Still Key to Computer Vision Systems, *Computer Design*, May 1985, pp. 69-74.

Horn, B.K.P., *Robot Vision*, MIT Press and McGraw-Hill Book Company: Cambridge, MA. 1986.

Hurlbert, A., and T. Poggio, Making Machines (and Artificial Intelligence) See, *Daedalus*, **V117**, Winter 1988, pp. 213-239.

Machine Vision, 1st ed., Society of Manufacturing Engineers: Dearborn Michigan, 1984.

Novini, A., Fundamentals of Machine Vision Lighting, *Materials Evaluation*, **45**, November 1987, pp. 1224-1229.

Page, C.J., and A. Pugh, Visually Interactive Gripping of Engineering Parts from Random Orientation, in *Artificial Vision for Robots*, ed. I. Aleksander, Kogan Page: London, 1983.

Petkovic, D., and E.B. Hinkle, A Rule-Based System for Verifying Engineering Specifications in Industrial Visual Inspection Applications, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **Vol. PAM1 -9**, No. 2, March 1987, pp. 206-311.

Pot, J., P. Coiffet, and P. Rines, Comparison of Five Methods for the Recognition of Industrial Parts, in *Artificial Vision for Robots*, ed. I. Aleksander, Kogan Page: London, 1983.

Richardson, D., Implementing MAP for Factory Control, *Manufacturing Engineering*, January 1988.

Sridaran, S., Off-line Robot Vision System Programming Using a Computer-Aided Design System, *MS Thesis, Virginia Polytechnic Institute and State University*, October, 1985.

Vanderbrug, G., D. Wilt, and J. Davis, Robotic Assembly of Keycaps to Keyboard Arrays, *Proceedings, 13th International Symposium on Industrial Robots and Robots 7*, Society of Manufacturing Engineers: Dearborn, Michigan, 1983.

West, D.C., Machine Vision In Practice, *IIE Transactions*, March 1988, pp. 393-399.

Wilkie, F., Industry Opens its Eyes to Machine Vision, *Process Engineering*, January 1986, pp. 41-43.

Winston, D.H., *Artificial Intelligence*, 2nd ed., Addision-Wesley Publishing Company, Inc.: Reading, MA, 1984.

Wright, P.K., and D.A. Bourne, *Manufacturing Intelligence*, Addison-Wesley Publishing Company, Inc.: New York, 1988.

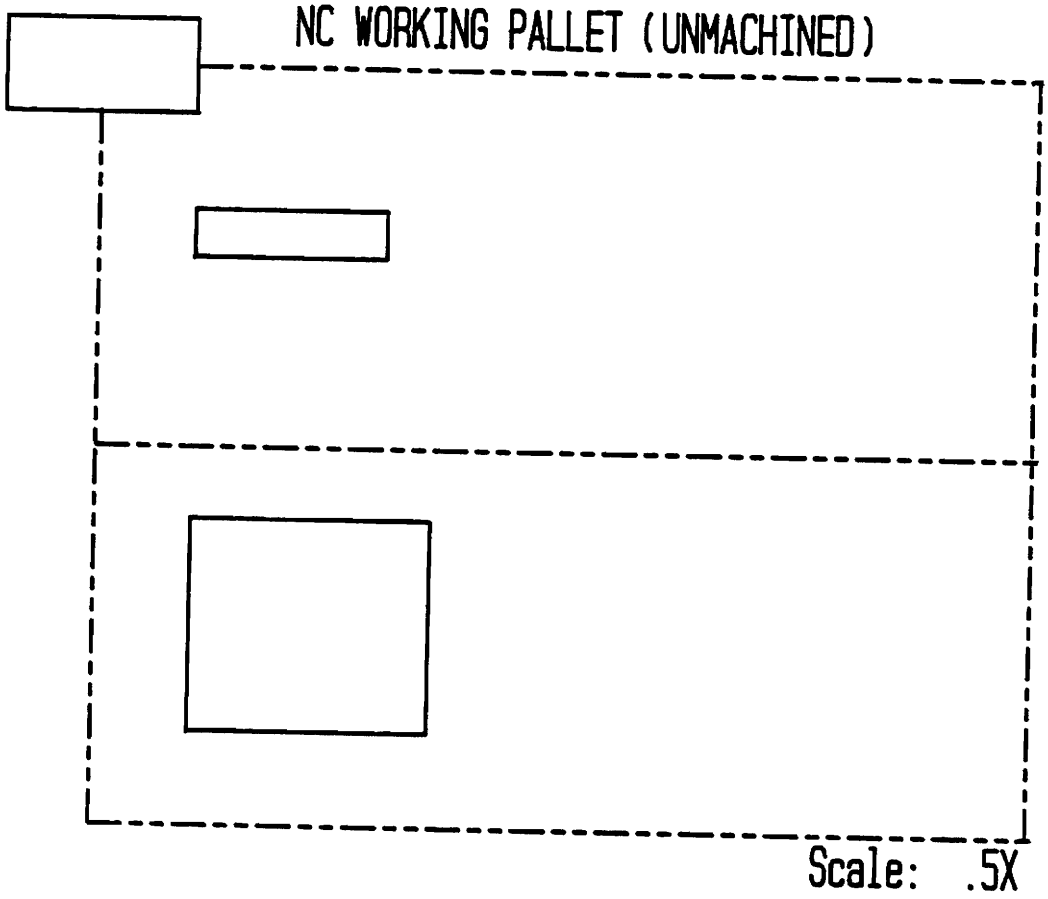
CADKEY User Reference Guide, Micro Control Systems Inc., Version 1.4-E, September 1986.

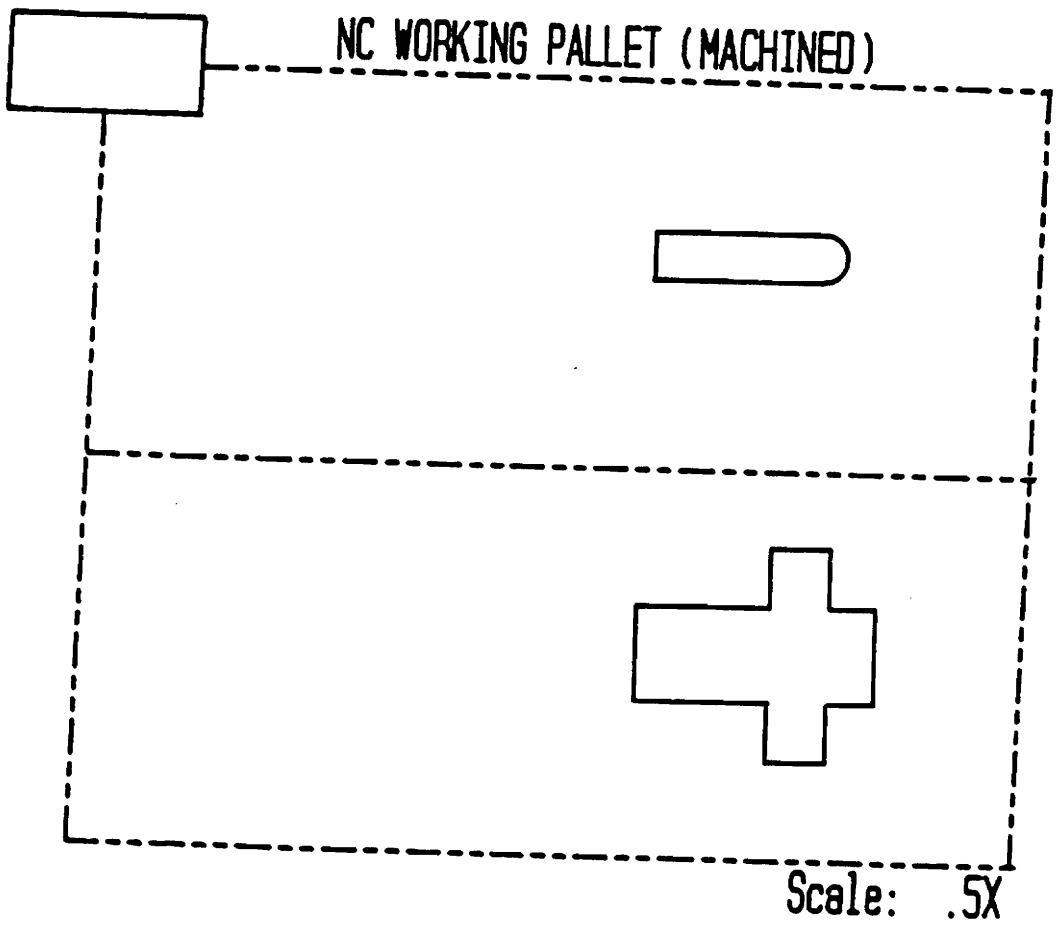
PN2305 Optomation II VPL Language Reference Manual, General Electric Company, February 1984.

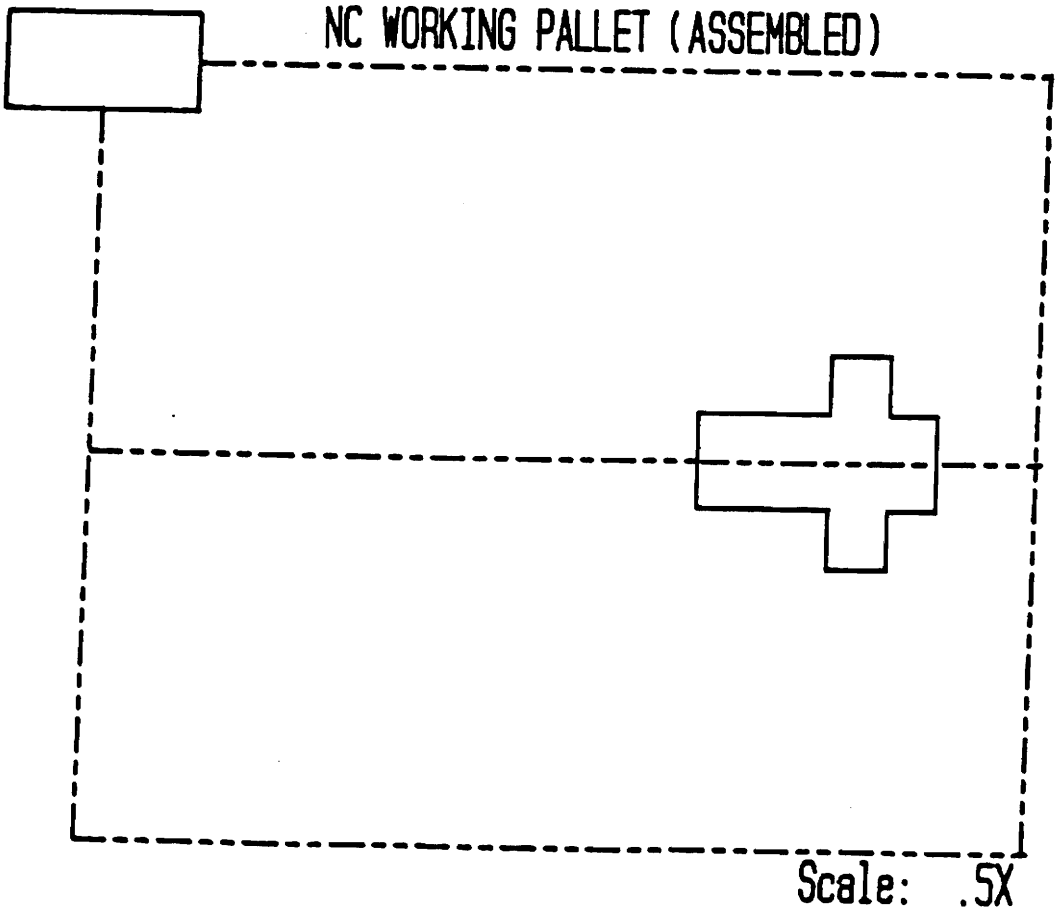
PN2305 Factory Vision Processor VPL Installation Manual, General Electric, March 1984.

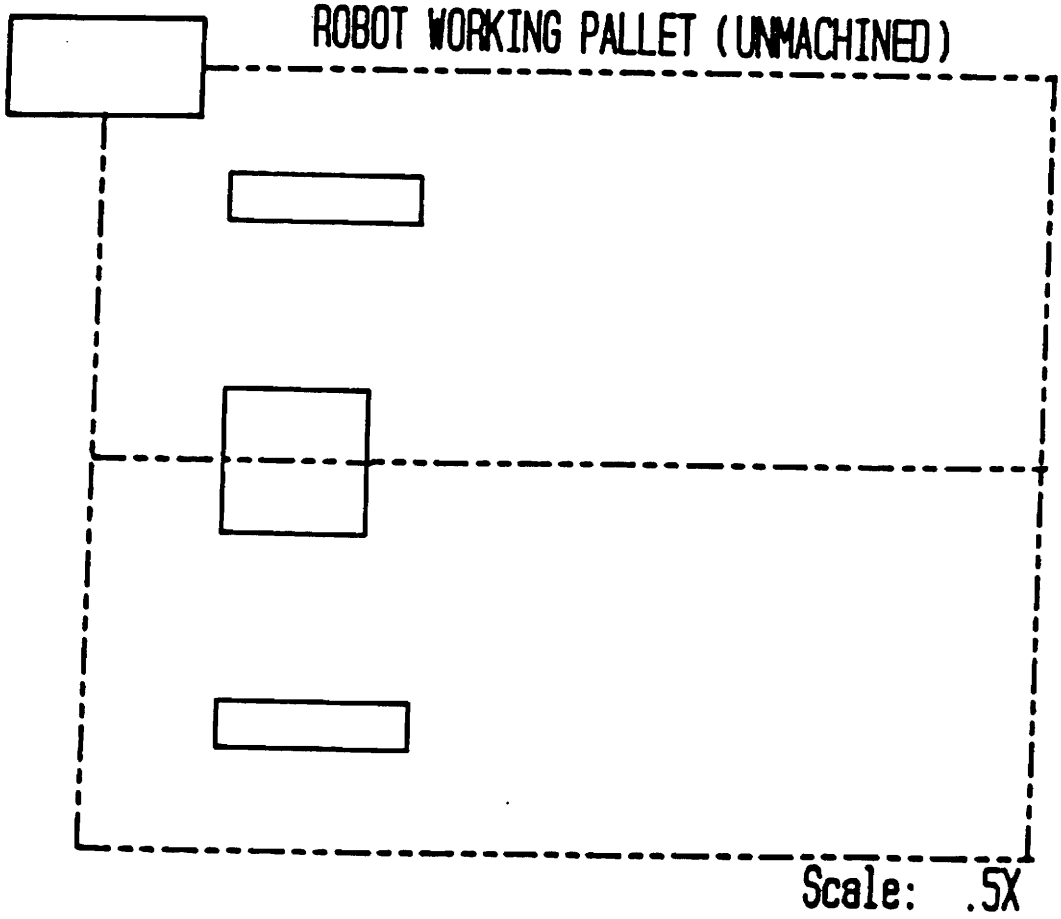
Appendix A. Pallet Configuration Drawings

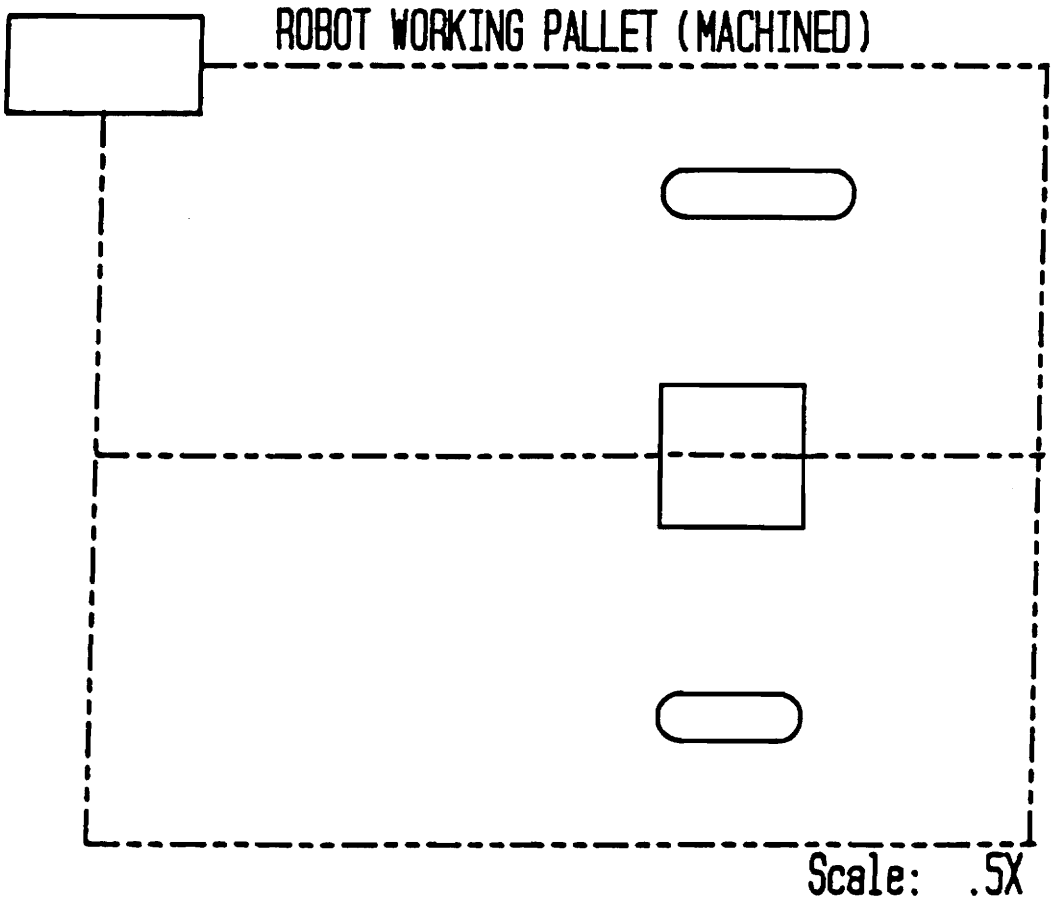
1. Blank Pallet, NC or Robot
2. NC Working Pallet, Unmachined
3. NC Working Pallet, Machined
4. NC Working Pallet, Assembled
5. Robot Working Pallet, Unmachined
6. Robot Working Pallet, Machined
7. Robot Working Pallet, Assembled
8. Bulk NC Base Pallet
9. Bulk Robot Base Pallet
10. Bulk Link Pallet

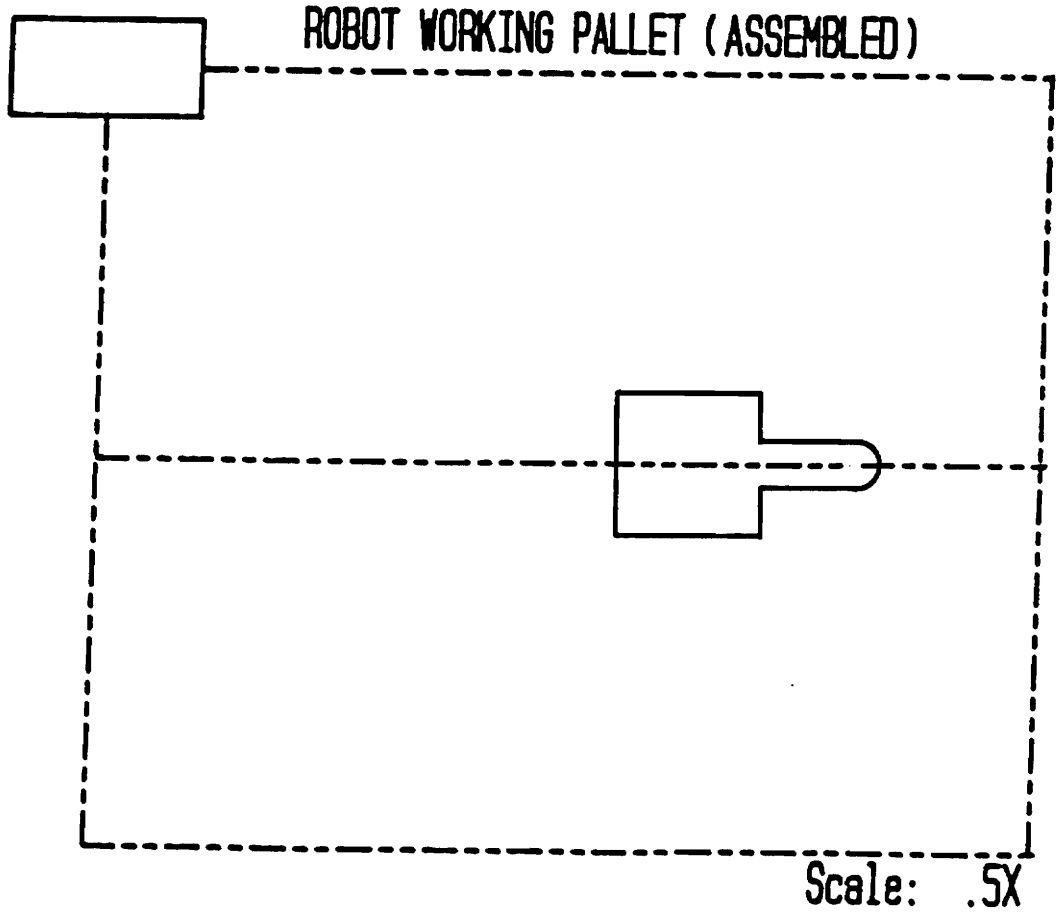

NC WORKING PALLET (UNMACHINED)
Scale: .5X

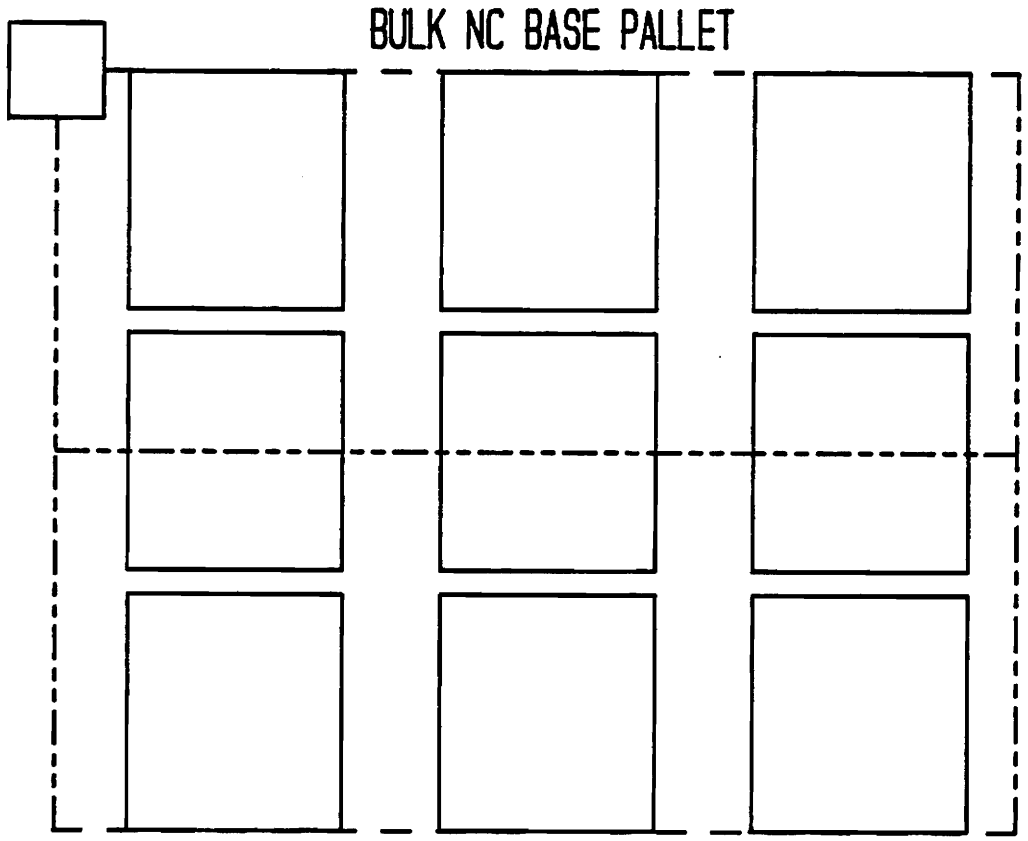

NC WORKING PALLET (MACHINED)
Scale: .5X

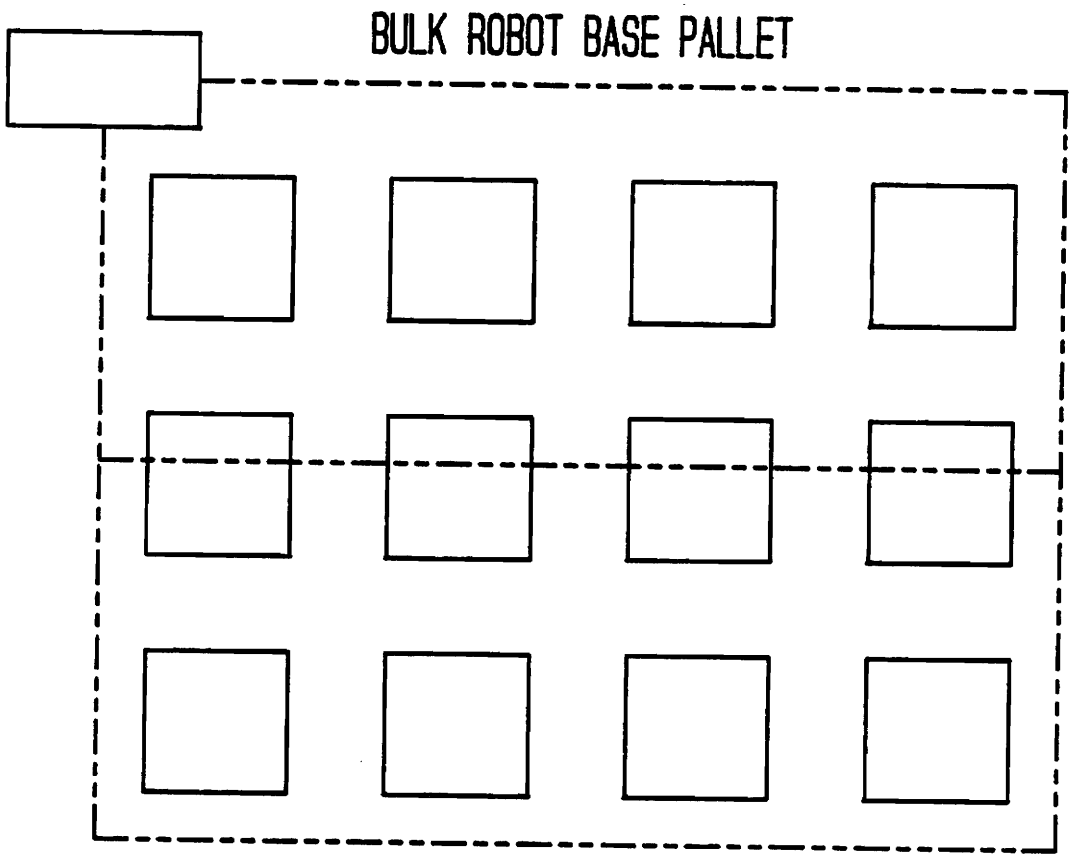

NC WORKING PALLET (ASSEMBLED)
Scale: .5X

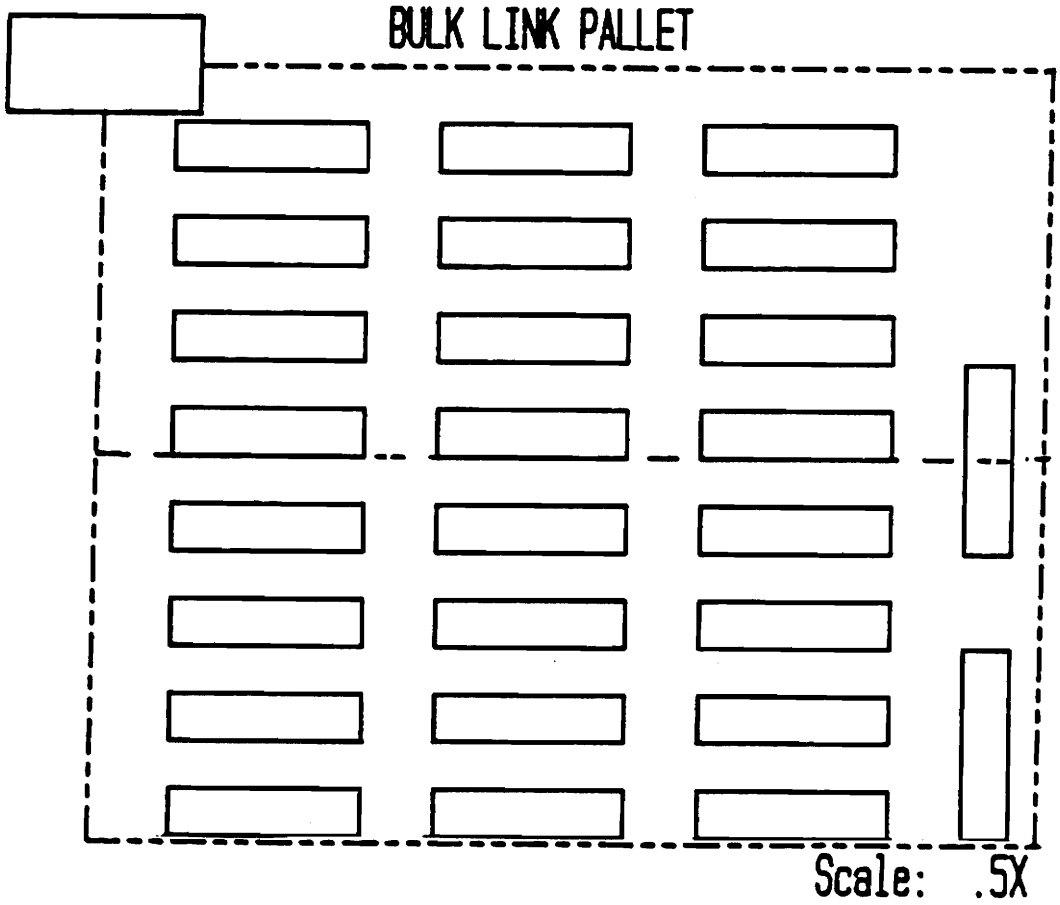

ROBOT WORKING PALLET (UNMACHINED)
Scale: .5X

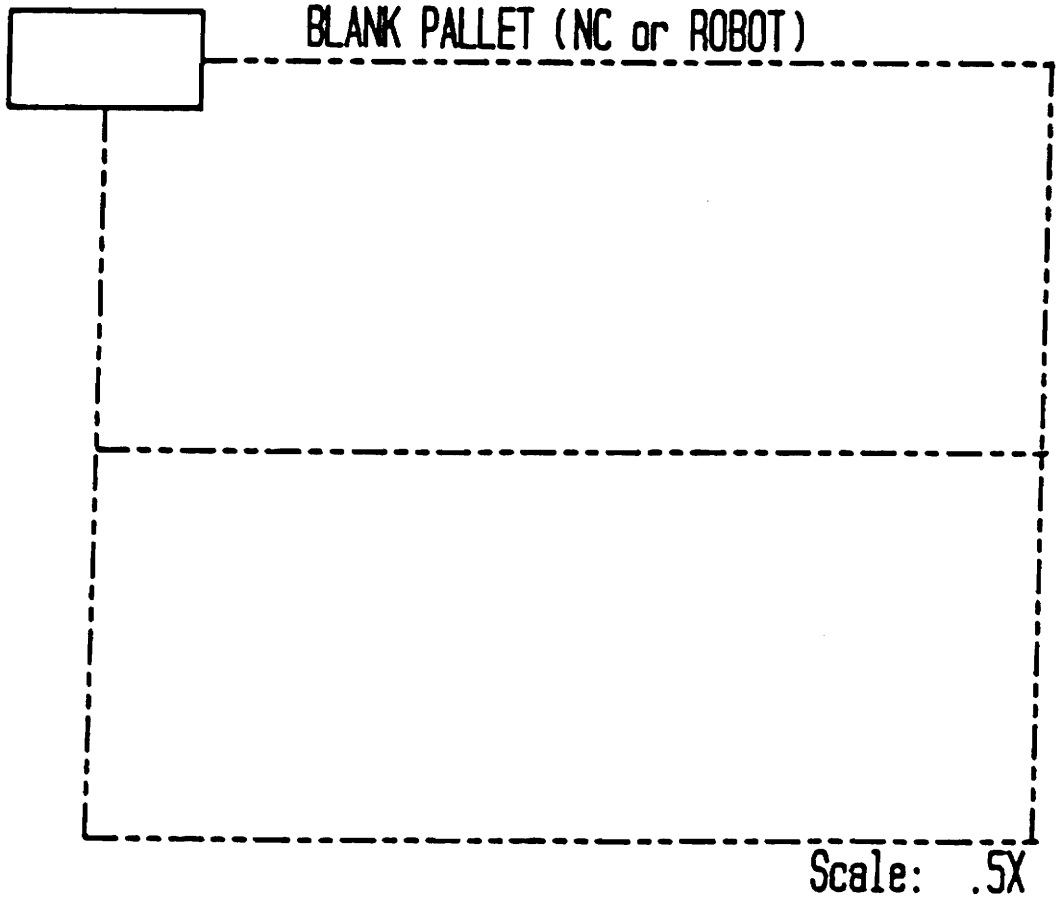

ROBOT WORKING PALLET (MACHINED)
Scale: .5X


ROBOT WORKING PALLET (ASSEMBLED)
Scale: .5X


BULK NC BASE PALLET


BULK ROBOT BASE PALLET


BULK LINK PALLET
Scale: .5X


BLANK PALLET (NC or ROBOT)
Scale: .5X

Appendix B. Example CADL Files

The following three lines show the meanings of each of the numbers in a CADL data primitive. The example CADL file below is for a Robot Working Pallet, Unmachined (Appendix A.)

LINE x1,y1,z1,x2,y2,z2,color,level,ltype,grpnum,subgrp,pen,lwdt

ARC xc,yc,zc,rad,ang1,ang2,vnum,color,lev,ltype,grpnum,subgrp,pen, lwdt

CIRCLE xc,yc,zc,rad,vnum,col,lev,ltype,grpnum,subgrp,pen,lwdt

CADL File For Robot Working Pallet, Unmachined

LINE 0.0000000, 8.0000000, 0.0000000, 10.0000000, 8.0000000,
0.0000000, 1, 1, 4, 0, 0, 1, 1
LINE 10.0000000, 8.0000000, 0.0000000, 10.0000000, 0.0000000,
0.0000000, 1, 1, 4, 0, 0, 1, 1
LINE 10.0000000, 0.0000000, 0.0000000, 1.0000000, 0.0000000,

0.0000000, 1, 1, 4, 0, 0, 1, 1

LINE 0.0000000, 0.5000000, 0.0000000, 0.0000000, 8.0000000,

0.0000000, 1, 1, 4, 0, 0, 1, 1

LINE 0.0000000, 4.0000000, 0.0000000, 10.0000000, 4.0000000,

0.0000000, 1, 1, 4, 0, 0, 1, 1

LINE 1.0000000, 1.5000000, 0.0000000, 3.0000000, 1.5000000,

0.0000000, 1, 1, 1, 1, 1, 1, 1

LINE 3.0000000, 1.5000000, 0.0000000, 3.0000000, 1.0000000,

0.0000000, 1, 1, 1, 1, 1, 1, 1

LINE 3.0000000, 1.0000000, 0.0000000, 1.0000000, 1.0000000,

0.0000000, 1, 1, 1, 1, 1, 1, 1

LINE 1.0000000, 1.0000000, 0.0000000, 1.0000000, 1.5000000,

0.0000000, 1, 1, 1, 1, 1, 1, 1

LINE 1.0000000, 7.0000000, 0.0000000, 3.0000000, 7.0000000,

0.0000000, 1, 1, 1, 2, 1, 1, 1

LINE 3.0000000, 7.0000000, 0.0000000, 3.0000000, 6.5000000,

0.0000000, 1, 1, 1, 2, 1, 1, 1

LINE 3.0000000, 6.5000000, 0.0000000, 1.0000000, 6.5000000,

0.0000000, 1, 1, 1, 2, 1, 1, 1

LINE 1.0000000, 6.5000000, 0.0000000, 1.0000000, 7.0000000,

0.0000000, 1, 1, 1, 2, 1, 1, 1

LINE 1.0000000, 4.7500000, 0.0000000, 2.5000000, 4.7500000,

0.0000000, 1, 1, 1, 3, 1, 1, 1

LINE 2.5000000, 4.7500000, 0.0000000, 2.5000000, 3.2500000,

0.0000000, 1, 1, 1, 3, 1, 1, 1

LINE 2.5000000, 3.2500000, 0.0000000, 1.0000000, 3.2500000,

0.0000000, 1, 1, 1, 3, 1, 1, 1

LINE 1.0000000, 3.2500000, 0.0000000, 1.0000000, 4.7500000,

0.0000000, 1, 1, 1, 3, 1, 1, 1

The **cadd.c** output file which is sent to the vision system COMPARISON@ program looks like this for the above CADL file:

30 NOITEMS = 3 < CR >

100 DATA(P(2,1),100,200,125,500,1800,100,200,675,500,1800, 225,175,400,600,900)

Appendix C. Cadd.c Program Listing

```
/* cadd.c */

#include<stdio.h>
#include<string.h>
#include<math.h>
#include<process.h>

#define maxelem 15              /*TOTAL NUMBER OF ELEMENTS IN OBJECT*/
#define group_limit 5          /*TOTAL NUMBER OF OBJECTS ON A PALLET*/
#define pi 3.141592
#define epsilon 0.0001

unsigned int maxgrp,groupsize[group_limit];
char   *lptr[11];
struct info { char type[7];              /*INFORMATION FROM ONE DATA*/
          float X,Y,R1,R2,R3;           /*       PRIMITIVE          */
          int line;
        } element [group_limit][maxelem];
struct results { float xbar,ybar,area,perim,pang; }
 group[group_limit];
                          /*OUTPUT INFO FOR EACH OBJECT*/
                          /*         ON PALLET         */
```

```
main(argc,argv)

  int argc;
  char *argv[];
{
  if (argc != 3)
    { puts("need 2 command line arguments"); return(1); }
  read_data(argv[1]);
  group_params();
  data_out(argv[2]);
} /* MAIN */

read_data(in_name)

  char    in_name[8];
{
  char    instring[100],next_type[7];
  int     i,j,grp,ele,next_grp;
  float   lr2,lr3;
  FILE    *in;

  if ((in = fopen(in_name,"rt")) == 0) {
    strcpy(instring,"can't open file ");
    strcat(instring,in_name);
    puts(instring);
    return(1); }

  grp = -1;                              /*INITIALIZE GROUP COUNTER*/

  while (fgets(instring,95,in)) {

    lptr[0] = strtok(instring," ,\t\n");     /* ASSIGN POINTERS TO */
    for (i = 1; i < 12;i++)                  /*      SUBSTRINGS     */
      lptr[i] = strtok(NULL," ,\t\n");       /*                     */
```

```
if (strcmp(lptr[0],NULL) == 0)          /* IGNORE LINE IF  */
  continue;                             /* TYPE IS MISSING */

if (strcmp(lptr[0],"VIEW") == 0)        /* IGNORE LINE IF */
  continue;                             /*  TYPE IS VIEW  */

if ((strcmp(lptr[0],"LINE") == 0) &&    /* IGNORE LINE IF  */
 (*lptr[9] == 4))                       /* IT IS A PHANTOM */
  continue;

sscanf(lptr[0],"%s",next_type);

switch (next_type[0]) {                 /* FIND WHICH GROUP */
  case 'L':                             /* THE CURRENT LINE */
 sscanf(lptr[10],"%d",&next_grp);       /*       IS IN      */
 break;                                 /*                  */
  case 'A':                             /*                  */
 sscanf(lptr[11],"%d",&next_grp);       /*                  */
 break;                                 /*                  */
  case 'C':                             /*                  */
 sscanf(lptr[9],"%d",&next_grp);        /*                  */
 break; }                               /*                  */

if (next_grp == grp)                        /* ADJUST GRP */
  { groupsize[grp] += 1; ele += 1; }        /*  AND ELE   */
else                                        /*  COUNTERS  */
  { grp = next_grp; groupsize[next_grp] = 1; ele = 1; } /*  */

sscanf(lptr[0],"%s",element[grp][ele].type);    /* CONVERT    */
sscanf(lptr[1],"%f",&element[grp][ele].X);      /* STRING     */
sscanf(lptr[2],"%f",&element[grp][ele].Y);      /* ELEMENTS   */
sscanf(lptr[4],"%f",&element[grp][ele].R1);     /* TO         */
switch (next_type[0]) {                         /* APPROPRIATE*/
case 'L':                                       /* VARIABLE   */
 sscanf(lptr[5],"%f",&element[grp][ele].R2);    /*    y2      */
 sscanf(lptr[9],"%d",&element[grp][ele].line);  /* line type  */
```

```
      break;                                                    /*            */
     case 'A':                                                  /*            */
      sscanf(lptr[5],"%f",&element[grp][ele].R2);       /*  theta 1   */
      sscanf(lptr[6],"%f",&element[grp][ele].R3);       /*  theta 2   */
      sscanf(lptr[10],"%d",&element[grp][ele].line);    /* line type  */
      lr2 = element[grp][ele].R2;                               /*            */
      lr3 = element[grp][ele].R3;                               /*            */
      break;                                                    /*            */
     case 'C':                                                  /*            */
      sscanf(lptr[8],"%d",&element[grp][ele].line);     /* line type  */
      break;}                                                   /*            */
     } /* while loop */

   maxgrp = grp;
   fclose(in);
   sort_data();
   return(0);
} /* READ_DATA */

sort_data()

{
#define sep_allow 0.001 /*ALLOWABLE SPACE BETWEEN CONNECTED ELEMENTS*/

   char ltype[7],next_found;
   int i,j,elem1,elem2,first_elem;
   float minx,yval;              /*SMALLEST X AND CORRESPONDING Y VALUE*/
   float lx,ly;                  /*FIRST ENDPONT FOR LINE*/
   float lx2,ly2;                /*TEMPORARY PLACEHOLDER FOR SWITCHING*/
   float lxc,lyc;                /*CENTER POINT FOR ARC*/
   float lr1,lr2,lr3;            /*SECOND ENDPT FOR LINE,ANGLES FOR ARC*/
   float x1,x2,y1,y2,cross_prod  /*ORDERED ENDPTS*/
   float next_x,next_y;

   for (i = 1; i <= maxgrp; i++) {  /*LOOP FOR EACH GROUP (OBJECT)*/
```

```
*ltype = *element[i][1].type;
if (ltype[0] == 'C')
  continue;

minx = 1000.0;                    /* INITIALIZE GROUP PROPERTIES */

/* DETERMINE LEFT-MOST NODE */

for (j = 1; j <= groupsize[i]; j++)
  {
  *ltype = *element[i][j].type;
  lx = element[i][j].X;
  ly = element[i][j].Y;
  lr1 = element[i][j].R1;
  lr2 = element[i][j].R2;
  lr3 = element[i][j].R3;

 switch(ltype[0])
{
case 'L':
   if (lx < minx) {
     elem1 = j;
     x1 = lr1;  y1 = lr2;
     minx = lx; yval = ly; }

   if (lr1 < minx) {
     elem1 = j;
     x1 = lx;  y1 = ly;
     minx = lr1; yval = lr2; }

   if ((fabs(lx - minx) < sep_allow) &&
       (fabs(ly - yval) < sep_allow)) {
     elem2 = j;
     x2 = lr1;  y2 = lr2; }

   if ((fabs(lr1 - minx) < sep_allow) &&
```

```
        (fabs(lr2 - yval) < sep_allow)) {
      elem2 = j;
      x2 = lx;  y2 = ly; }

    break;

  case 'A':
    lr2 = lr2 * pi/180.0;  lr3 = lr3 * pi/180.0;
    if ((lx + lr1*cos(lr2)) < minx) {
      elem1 = j;
      x1 = (lx + lr1*cos(lr3));  y1 = (ly + lr1*sin(lr3));
      minx = (lx + lr1*cos(lr2)); yval = (ly + lr1*sin(lr2)); }

    if ((lx + lr1*cos(lr3)) < minx) {
      elem1 = j;
      x1 = (lx + lr1*cos(lr2));  y1 = (ly + lr1*sin(lr2));
      minx = (lx + lr1*cos(lr3)); yval = (ly + lr1*sin(lr3)); }

    if ((fabs((lx + lr1*cos(lr2)) - minx) < sep_allow) &&
        (fabs((ly + lr1*sin(lr2)) - yval) < sep_allow)) {
      elem2 = j;
      x2 = (lx + lr1*cos(lr3));  y2 = (ly + lr1*sin(lr3)); }

    if ((fabs((lx + lr1*cos(lr3)) - minx) < sep_allow) &&
        (fabs((ly + lr1*sin(lr3)) - yval) < sep_allow)) {
      elem2 = j;
      x2 = (lx + lr1*cos(lr2));  y2 = (ly + lr1*sin(lr2)); }

    break;
  }
  } /* ELEMENT LOOP */

  /* ALLIGN THE ELEMENTS COUNTER-CLOCKWISE */

  cross_prod = ((x1-minx)*(y2-yval) - (x2-minx)*(y1-yval));
  if (cross_prod > sep_allow)                    /*DETERMINE FIRST*/
```

```
first_elem = elem1;                          /*  ELEMENT      */
 else if (cross_prod < -sep_allow)           /*               */
first_elem = elem2;                          /*               */
 else {                                      /*               */
if (y1 < y2)                                 /*               */
  first_elem = elem1;                        /*               */
else                                         /*               */
  first_elem = elem2; }                      /*               */
 .
 *ltype = *element[i][first_elem].type;
 switch(ltype[0])
{
case 'L':
  lx = element[i][first_elem].X;
  ly = element[i][first_elem].Y;
  next_x = element[i][first_elem].R1;
  next_y = element[i][first_elem].R2;
  break;
case 'A':
  lxc = element[i][first_elem].X;
  lyc = element[i][first_elem].Y;
  lr1 = element[i][first_elem].R1;
  lr2 = element[i][first_elem].R2;
  lr3 = element[i][first_elem].R3;
  lr2 = lr2 * pi/180.0;  lr3 = lr3 * pi/180.0;
  lx = lxc + lr1*cos(lr2);
  ly = lyc + lr1*sin(lr2);
  next_x = lxc + lr1*cos(lr3);
  next_y = lyc + lr1*sin(lr3);
  break;
}
 if ((fabs(lx - minx) > sep_allow) ||
  (fabs(ly - yval) > sep_allow)) {

/*STARTING WITH FIRST ELEMENT,IF 1ST ENDPT OF AN ELEMENT */
/*MATCHES THE END OF FIRST ELEMENT, GO TO ITS 2ND ENDPT. */
```

```
/*IF ST ENDPT DOES NOT MATCH, CHECK 2ND ENDPT.  IF THIS  */
/*ENDPOINT MATCHES, ELEMENT IS ADDED TO CHAIN AFTER      */
/*SWITCHING ENDPTS.  PROCEDURE REPEATED FOR ALL ELEMENTS */

switch(ltype[0])
  {
  case 'L':
    element[i][first_elem].X = next_x;
    element[i][first_elem].Y = next_y;
    element[i][first_elem].R1 = lx;
    element[i][first_elem].R2 = ly;
    next_x = lx;
    next_y = ly;
    break;
  case 'A':
    strcpy(element[i][first_elem].type, "ARC2");
    element[i][first_elem].R2 = lr3 * 180.0/pi;
    element[i][first_elem].R3 = lr2 * 180.0/pi;
    next_x = lx;
    next_y = ly;
    break;
  }}
 while ((fabs(next_x - minx) > sep_allow) ||
     (fabs(next_y - yval) > sep_allow)) {
next_found = 'F';
for (j = 1; j <= groupsize[i]; j++) {
  if (j == first_elem)
    continue;
  if (next_found == 'T')
    continue;
  *ltype = *element[i][j].type;
  switch(ltype[0])
    {
    case 'L':
      lx = element[i][j].X;
      ly = element[i][j].Y;
```

```
      lx2 = element[i][j].R1;
      ly2 = element[i][j].R2;
      break;
    case 'A':
      lxc = element[i][j].X;
      lyc = element[i][j].Y;
      lr1 = element[i][j].R1;
      lr2 = element[i][j].R2;
      lr3 = element[i][j].R3;
      lr2 =lr2 * pi/180.0;  lr3 = lr3 * pi/180.0;
      lx = lxc + lr1*cos(lr2);
      ly = lyc + lr1*sin(lr2);
      lx2 = lxc + lr1*cos(lr3);
      ly2 = lyc + lr1*sin(lr3);
      break;
    } /* switch */
  if ((fabs(lx - next_x) < sep_allow) &&
      (fabs(ly - next_y) < sep_allow)) {
    next_x = lx2;
    next_y = ly2;
    next_found = 'T';
    first_elem = j;
    continue;}

  if ((fabs(lx2 - next_x) < sep_allow) &&
      (fabs(ly2 - next_y) < sep_allow)) {
    next_x = lx;
    next_y = ly;
    next_found = 'T';
    first_elem = b;
    switch(ltype[0])
      {
      case 'L':
      element[i][j].X = lx2;
      element[i][j].Y = ly2;
      element[i][j].R1 = lx;
```

```
            element[i][j].R2 = ly;
            break;
            case 'A':
           strcpy(element[i][j].type, "ARC2");
           element[i][j].R2 = lr3 * 180.0/pi;
           element[i][j].R3 = lr2 * 180.0/pi;
           break;
            } /* switc  */
          }
       } /* for loop */
     } /* while loop */
    } /* group loop */
  return(0);
} /* SORT_DATA */

group_params()

{
  char ltype[7];
  unsigned int i,j,line;
  float alpha,theta_mid;
  float lx,ly,lr1,lr2,lr3,cross_prod;
  float deltPerim,fpX,fpY,spX,spY,spXY;      /*DELTA FOR EACH ELEMENT*/
  float xmid,xtip,ymid,ytip,ystart1,ystart2; /*SEE TEXT FOR DEFINITION*/
  float deltArea,deltArea1,deltArea2,deltArea3;  /*AREAS OF ARC*/
  float deltspX,deltspX1,deltspX2,deltspX3;
  float deltspY,deltspY1,deltspY2,deltspY3;
  float deltspXY,deltspXY1,deltspXY2,deltspXY3;
  float x1,x2,y1,y2,Xc,Xc1,Xc2,Xc3,Yc,Yc1,Yc2,Yc3;
  float Xca,Yca;
  float m1,m2,ma,mb,mc,md,C,E;

  for (i = 1; i <= maxgrp; i++) {   /* LOOP FOR EACH GROUP (OBJECT)*/

    fpX = fpY = spX = spY = spXY = 0.0; /*INITIALIZE GROUP PROPERTIES*/
    group[i].area = 0.0;                                          /**/
```

```
group[i].perim = 0.0;                                                    /**/

for (j = 1; j <= groupsize[i]; j++) {     /*LOOP FOR EACH ELEMENT*/
  *ltype = *element[i][j].type;
  lx = element[i][j].X;
  ly = element[i][j].Y;
  lr1 = element[i][j].R1;
  lr2 = element[i][j].R2;
  lr3 = element[i][j].R3;
  line = element[i][j].line;
  switch(ltype[0]) {
 case 'L':
   deltArea = (lx*lr2 - ly*lr1)/2.0;   /* AREA FOR LINE ELEMENT */
   if (line == '2')                    /* A HOLE IS SHOWN       */
     deltArea = -deltArea;             /* BY LINE TYPE 2 (DASH) */

  Xc = (lx+lr1)/3.0; Yc = (ly+lr2)/3.0;          /* CENTROID */

  deltPerim = sqrt(pow((lr1-lx),2) + pow((lr2-ly),2));/*PERIMETER*/

  if (fabs(lx) > epsilon)                  /* SECOND PRODUCT */
    m1 = ly/lx;                                              /**/
  else                                                       /**/
    m1 = 1.0/epsilon;                                        /**/
  if (fabs(lr1) > epsilon)                                   /**/
    m2 = lr2/lr1;                                            /**/
  else                                                       /**/
    m2 = 1.0/epsilon;                                        /**/
                                                             /**/
  if (lx < lr1) {                                            /**/
   xmid = lx; ymid = ly; xtip = lr1; ytip = lr2;             /**/
   if (m1>m2) {                                              /**/
     ma = m1; mb = md = m2; ystart1 = ly;                    /**/
     if (fabs(xmid) > epsilon)                               /**/
    ystart2 = mb*xmid;                                       /**/
    else                                                     /**/
```

```
      ystart2 = 0.0;                                                    /**/
       if (fabs(xtip-xmid) > epsilon)                                   /**/
      mc = (ytip-ymid)/(xtip-xmid);                                     /**/
       else                                                             /**/
      mc = 1.0/epsilon; }                                               /**/
     else {                                                             /**/
       ma = mc = m2; mb = m1; ystart2 = iy;                             /**/
       if (fabs(xmid) > epsilon)                                        /**/
      ystart1 = ma*xmid;                                                /**/
       else                                                             /**/
      ystart1 = 0.0;                                                    /**/
       if (fabs(xtip-xmid) > epsilon)                                   /**/
      md = (ytip-ymid)/(xtip-xmid);                                     /**/
       else                                                             /**/
      md = 1.0/epsilon; }}                                              /**/
  else {                                                                /**/
    xmid = ir1; ymid = ir2; xtip = ix; ytip = iy;                       /**/
    if (m1>m2) {                                                        /**/
       ma = mc = m1; mb = m2; ystart2 = ir2;                            /**/
       if (fabs(xmid) > epsilon)                                        /**/
     ystart1 = ma*xmid;                                                 /**/
       else                                                             /**/
     ystart1 = 0.0;                                                     /**/
       if (fabs(xtip-xmid) > epsilon)                                   /**/
     md = (ytip-ymid)/(xtip-xmid);                                      /**/
       else                                                             /**/
     md = 1.0/epsilon; }                                                /**/
   else {                                                               /**/
       ma = m2; mb = md = m1; ystart1 = ir2;                            /**/
       if (fabs(xmid) > epsilon)                                        /**/
    ystart2 = mb*xmid;                                                  /**/
       else                                                             /**/
    ystart2 = 0.0;                                                      /**/
       if (fabs(xtip-xmid) > epsilon)                                   /**/
    mc = (ytip-ymid)/(xtip-xmid);                                       /**/
       else                                                             /**/
```

```
        mc = 1.0/epsilon; }}                                        /**/
                                                                    /**/
    C = ystart1 - mc*xmid; E = ystart2 - md*xmid;                   /**/
                                                                    /**/
    deltspX  = ((pow(ma,3)-pow(mb,3))*pow(xmid,4) +                 /**/
     (pow(mc,3)-pow(md,3))*(pow(xtip,4)-pow(xmid,4)) +              /**/
        4*(C*pow(mc,2)-E*pow(md,2))*                                /**/
            (pow(xtip,3)-pow(xmid,3)) +                             /**/
                6*(pow(C,2)*mc-pow(E,2)*md)                         /**/
                 (pow(xtip,2)-pow(xmid,2)) +                        /**/
                    4*(pow(C,3)-pow(E,3))*(xtip-xmid))              /**/
                     /12.0;                                         /**/
                                                                    /**/
    deltspY  = (3*(ma-mb)*pow(xmid,4) +                             /**/
             3*(mc-md)*(pow(xtip,4)-pow(xmid,4)) +                  /**/
             4*(C-E)*(pow(xtip,3)-pow(xmid,3)))/12.0;               /**/
                                                                    /**/
    deltspXY  = (3*pow(xmid,4)*(pow(ma,2)-pow(mb,2)) +              /**/
          3*(pow(mc,2)-pow(md,2))*(pow(xtip,4)-pow(xmid,4))         /**/
            + 8*(C*mc-E*md)*(pow(xtip,3)-pow(xmid,3)) +             /**/
              6*(pow(C,2)-pow(E,2))*                                /**/
               (pow(xtip,2)-pow(xmid,2)))/24.0;                     /**/
                                                                    /**/
    if (deltArea < 0.0) {                                           /**/
      deltspX = -deltspX;                                           /**/
      deltspY = -deltspY;                                           /**/
      deltspXY = -deltspXY; }                                       /**/
    break;
  case 'A':
    lr2 = lr2*pi/180.0; lr3 = lr3*pi/180.0;
    x1 = lx + lr1*cos(lr2); y1 = ly + lr1*sin(lr2);
    x2 = lx + lr1*cos(lr3); y2 = ly + lr1*sin(lr3);

    Xc1 = (x1+lx)/3.0; Yc1 = (y1+ly)/3.0;          /* THE TWO      */
    Xc2 = (lx+x2)/3.0; Yc2 = (ly+y2)/3.0;          /* TRIANGLES OF */
                                                   /* ARC AREA     */
```

```
deltArea1 = (x1*ly - y1*lx)/2.0;                                    /**/
deltArea2 = (lx*y2 - ly*x2)/2.0;                                    /**/
                                                                    /**/

if (fabs(x1) > epsilon)                          /* SECOND PRODUCT */
  m1 = y1/x1;                                    /* FIRST TRIANGLE */
else                                                                /**/
  m1 = 1.0/epsilon;                                                 /**/
if (fabs(lx) > epsilon)                                             /**/
  m2 = ly/lx;                                                       /**/
else                                                                /**/
  m2 = 1.0/epsilon;                                                 /**/
                                                                    /**/
if (x1 < lx) {                                                      /**/
  xmid = x1; ymid = y1; xtip = lx; ytip = ly;                       /**/
  if (m1>m2) {                                                      /**/
    ma = m1; mb = md = m2; ystart1 = y1;                            /**/
    if (fabs(xmid) > epsilon)                                       /**/
   ystart2 = mb*xmid;                                               /**/
    else                                                            /**/
   ystart2 = 0.0;                                                   /**/
    if (fabs(xtip-xmid) > epsilon)                                  /**/
   mc = (ytip-ymid)/(xtip-xmid);                                    /**/
    else                                                            /**/
   mc = 1.0/epsilon; }                                              /**/
  else {                                                            /**/


    ma = mc = m2; mb = m1; ystart2 = y1;                            /**/
    if (fabs(xmid) > epsilon)                                       /**/
   ystart1 = ma*xmid;                                               /**/
    else                                                            /**/
   ystart1 = 0.0;                                                   /**/
    if (fabs(xtip-xmid) > epsilon)                                  /**/
   md = (ytip-ymid)/(xtip-xmid);                                    /**/
    else                                                            /**/
```

```
      md = 1.0/epsilon; }}                                          /**/
    else {                                                          /**/
     xmid = lx; ymid = ly; xtip = x1; ytip = y1;                    /**/
     if (m1>m2) {                                                   /**/
      ma = mc = m1; mb = m2; ystart2 = ly;                          /**/
      if (fabs(xmid) > epsilon)                                     /**/
     ystart1 = ma*xmid;                                             /**/
      else                                                          /**/
     ystart1 = 0.0;                                                 /**/
      if (fabs(xtip-xmid) > epsilon)                                /**/
     md = (ytip-ymid)/(xtip-xmid);                                  /**/
      else                                                          /**/
     md = 1.0/epsilon; }                                            /**/
    else {                                                          /**/
      ma = m2; mb = md = m1; ystart1 = ly;                          /**/
      if (fabs(xmid) > epsilon)                                     /**/
     ystart2 = mb*xmid;                                             /**/
      else                                                          /**/
     ystart2 = 0.0;                                                 /**/
      if (fabs(xtip-xmid) > epsilon)                                /**/
     mc = (ytip-ymid)/(xtip-xmid);                                  /**/
      else                                                          /**/
     mc = 1.0/epsilon; }}                                           /**/
                                                                    /**/
 C = ystart1 - mc*xmid; E = ystart2 - md*xmid;                      /**/
                                                                    /**/
deltspX1 = ((pow(ma,3)-pow(mb,3))*pow(xmid,4) +                     /**/
    (pow(mc,3)-pow(md,3))*(pow(xtip,4)-pow(xmid,4)) +               /**/
      4*(C*pow(mc,2)-E*pow(md,2))*                                  /**/
        (pow(xtip,3)-pow(xmid,3)) +                                 /**/
          6*(pow(C,2)*mc-pow(E,2)*md)*                              /**/
           (pow(xtip,2)-pow(xmid,2)) +                              /**/
             4*(pow(C,3)-pow(E,3))*(xtip-xmid))                     /**/
              /12.0;                                                /**/
                                                                    /**/
deltspY1 = (3*(ma-mb)*pow(xmid,4) +                                 /**/
```

```
          3*(mc-md)*(pow(xtip,4)-pow(xmid,4)) +                    /**/
          4*(C-E)*(pow(xtip,3)-pow(xmid,3)))/12.0;                 /**/
                                                                   /**/
deltspXY1 = (3*pow(xmid,4)*(pow(ma,2)-pow(mb,2)) +                 /**/
    3*(pow(mc,2)-pow(md,2))*(pow(xtip,4)-pow(xmid,4))              /**/
     + 8*(C*mc-E*md)*(pow(xtip,3)-pow(xmid,3)) +                   /**/
       6*(pow(C,2)-pow(E,2))*                                      /**/
        (pow(xtip,2)-pow(xmid,2)))/24.0;                           /**/

if (fabs(lx) > epsilon)                          /* SECOND PRODUCT */
  m1 = ly/lx;                                   /* SECOND TRIANGLE */
else                                                               /**/
  m1 = 1.0/epsilon;                                                /**/
if (fabs(x2) > epsilon)                                            /**/
  m2 = y2/x2;                                                      /**/
else                                                               /**/
  m2 = 1.0/epsilon;                                                /**/
                                                                   /**/
if (lx < x2) {                                                     /**/
 xmid = lx; ymid = ly; xtip = x2; ytip = y2;                       /**/
 if (m1>m2) {                                                      /**/
   ma = m1; mb = md = m2; ystart1 = ly;                            /**/
   if (fabs(xmid) > epsilon)                                       /**/
  ystart2 = mb*xmid;                                               /**/
   else                                                            /**/
  ystart2 = 0.0;                                                   /**/
   if (fabs(xtip-xmid) > epsilon)                                  /**/
  mc = (ytip-ymid)/(xtip-xmid);                                    /**/
   else                                                            /**/
  mc = 1.0/epsilon; }                                              /**/
 else {                                                            /**/
   ma = mc = m2; mb = m1; ystart2 = ly;                            /**/
   if (fabs(xmid) > epsilon)                                       /**/
 ystart1 = ma*xmid;                                                /**/
   else                                                            /**/
 ystart1 = 0.0;                                                    /**/
```

```
      if (fabs(xtip-xmid) > epsilon)                                    /**/
     md = (ytip-ymid)/(xtip-xmid);                                      /**/
      else                                                              /**/
     md = 1.0/epsilon; }}                                               /**/
  else {                                                                /**/
    xmid = x2; ymid = y2; xtip = lx; ytip = ly;                         /**/
    if (m1>m2) {                                                        /**/
      ma = mc = m1; mb = m2; ystart2 = y2;                              /**/
      if (fabs(xmid) > epsilon)                                         /**/
     ystart1 = ma*xmid;                                                 /**/
      else                                                              /**/
     ystart1 = 0.0;                                                     /**/
      if (fabs(xtip-xmid) > epsilon)                                    /**/
     md = (ytip-ymid)/(xtip-xmid);                                      /**/
      else                                                              /**/
     md = 1.0/epsilon; }                                                /**/
    else {                                                              /**/
      ma = m2; mb = md = m1; ystart1 = y2;                              /**/
      if (fabs(xmid) > epsilon)                                         /**/
     ystart2 = mb*xmid;                                                 /**/
      else                                                              /**/
     ystart2 = 0.0;                                                     /**/
      if (fabs(xtip-xmid) > epsilon)                                    /**/
     mc = (ytip-ymid)/(xtip-xmid);                                      /**/
      else                                                              /**/
     mc = 1.0/epsilon; }}                                               /**/
                                                                        /**/
C = ystart1 - mc*xmid; E = ystart2 - md*xmid;                           /**/
                                                                        /**/
deltspX2 = ((pow(ma,3)-pow(mb,3))*pow(xmid,4) +                         /**/
   (pow(mc,3)-pow(md,3))*(pow(xtip,4)-pow(xmid,4)) +                    /**/
     4*(C*pow(mc,2)-E*pow(md,2))*                                       /**/
        (pow(xtip,3)-pow(xmid,3)) +                                     /**/
         6*(pow(C,2)*mc-pow(E,2)*md)*                                   /**/
           (pow(xtip,2)-pow(xmid,2)) +                                  /**/
            4*(pow(C,3)-pow(E,3))*(xtip-xmid))                          /**/
```

```
                /12.0;                                                  /**/
                                                                        /**/
deltspY2 = (3*(ma-mb)*pow(xmid,4) +                                     /**/
          3*(mc-md)*(pow(xtip,4)-pow(xmid,4)) +                         /**/
          4*(C-E)*(pow(xtip,3)-pow(xmid,3)))/12.0;                      /**/
                                                                        /**/
deltspXY2 = (3*pow(xmid,4)*(pow(ma,2)-pow(mb,2)) +                      /**/
   3*(pow(mc,2)-pow(md,2))*(pow(xtip,4)-pow(xmid,4))                    /**/
    + 8*(C*mc-E*md)*(pow(xtip,3)-pow(xmid,3)) +                         /**/
          6*(pow(C,2)-pow(E,2))*                                        /**/
           (pow(xtip,2)-pow(xmid,2)))/24.0;                             /**/
                                                                        /**/
if (deltArea1 < 0.0) {                                                  /**/
  deltspX1 = -deltspX1;                                                 /**/
  deltspY1 = -deltspY1;                                                 /**/
  deltspXY1 = -deltspXY1; }                                             /**/
if (deltArea2 < 0.0) {                                                  /**/
  deltspX2 = -deltspX2;                                                 /**/
  deltspY2 = -deltspY2;                                                 /**/
  deltspXY2 = -deltspXY2; }                                             /**/

alpha = fabs((ir3 - ir2)/2.0);              /*THE PARTIAL CIRCLE*/
theta_mid = (ir2 + ir3)/2.0;                                            /**/
cross_prod = (cos(theta_mid)*sin(ir3) -                                 /**/
        sin(theta_mid)*cos(ir3));                                       /**/
Xc3 = lx + 2.0*ir1*sin(alpha)/(3.0*alpha) * cos(theta_mid);             /**/
Yc3 = ly + 2.0*ir1*sin(alpha)/(3.0*alpha) * sin(theta_mid);             /**/
deltArea3 = alpha*pow(ir1,2);                                           /**/
if (line == '2')                             /* A HOLE IS SHOWN */
  deltArea3 = -deltArea3;                    /* BY A DASHED LINE */
/*if (itype[3] == '2') */
/*  deltArea3 = -deltArea3; */
if (cross_prod < 0.0)                                                   /**/
  deltArea3 = -deltArea3;                                               /**/
                                                                        /**/
Xca = lx + 2.0*ir1*sin(alpha)*cos(theta_mid) /*SECOND PRODUCT */
```

```
            /(3.0*alpha);                                                    /**/
    Yca = ly + 2.0*lr1*sin(alpha)*sin(theta_mid)/(3.0*alpha);                /**/
                                                                             /**/
    deltspX3 = pow(lr1,4)/8.0*                                               /**/
            (2*alpha-sin(2*alpha)*cos(2*theta_mid)) +                        /**/
            alpha*pow(lr1,2)*(pow(Yca,2) - 4*pow(lr1,2)*                     /**/
            pow(sin(alpha),2)*pow(sin(theta_mid),2)                          /**/
            /(9.0*pow(alpha,2)));                                            /**/
                                                                             /**/
    deltspY3 = pow(lr1,4)/8.0*                                               /**/
            (2*alpha+sin(2*alpha)*cos(2*theta_mid)) +                        /**/
            alpha*pow(lr1,2)*(pow(Xca,2) -                                   /**/
            4*pow(lr1,2)*pow(sin(alpha),2)*pow(cos(theta_mid),2)/**/
            /(9.0*pow(alpha,2)));                                            /**/
                                                                             /**/
    deltspXY3 = -pow(lr1,4)/8.0*                                             /**/
            (sin(2*alpha)*sin(2*theta_mid)) +                                /**/
            alpha*pow(lr1,2)*(Xca*Yca -                                      /**/
            4*pow(lr1,2)*pow(sin(alpha),2)*sin(theta_mid)                    /**/
            *cos(theta_mid)/(9.0*pow(alpha,2)));                             /**/


                                                          /* TOTAL WEDGE */
    deltArea = deltArea1 + deltArea2 + deltArea3;              /* AREA */
                                                                     /**/
    Xc = (Xc1*deltArea1 + Xc2*deltArea2 + Xc3*deltArea3)/*CENTROID*/
       /deltArea;                                                    /**/
    Yc = (Yc1*deltArea1 + Yc2*deltArea2 + Yc3*deltArea3)             /**/
       /deltArea;                                                    /**/
  /**/
    deltPerim = 2.0*alpha*lr1;                          /* PERIMETER */
  /**/
    deltspX = deltspX1 + deltspX2 + deltspX3;    /* SECOND PRODUCT */
    deltspY = deltspY1 + deltspY2 + deltspY3;                        /**/
    deltspXY = deltspXY1 + deltspXY2 + deltspXY3;                    /**/
    break;
  case 'C':
```

```
    deltArea = pi*pow(lr1,2);                    /*      AREA        */
     if (line == '2')                            /*                  */
       deltArea = -deltArea;                     /*                  */

     Xc = lx; Yc = ly;                                  /* CENTROID */

     deltPerim = 2.0*pi*lr1;                           /* PERIMETER */

     deltspX = deltArea*(pow(lr1,2)/4.0 + pow(Yc,2)); /*SECOND PROD*/
     deltspY = deltArea*(pow(lr1,2)/4.0 + pow(Xc,2));          /**/
     deltspXY = deltArea*Xc*Yc;                                /**/
     break; }

    group[i].area += deltArea;          /* TOTAL OBJ. AREA     */
    group[i].perim += deltPerim;        /* TOTAL OBJ.PERIMETER */
    fpX += deltArea*Yc;                 /* FIRST PRODUCT -- X  */
    fpY += deltArea*Xc;                 /* FIRST PRODUCT -- Y  */
    spX += deltspX;                     /* SECOND PRODUCT -- X */
    spY += deltspY;                     /* SECOND PRODUCT -- Y */
    spXY += deltspXY;                   /* SECOND PRODUCT --XY */
    } /* element loop */
  if (fabs(group[i].area) > epsilon) {
    group[i].xbar = fpY/group[i].area; /*DETERMINE CENTROID LOCATION*/
    group[i].ybar = fpX/group[i].area; }     /* OF THE ENTIRE GROUP */

  spX = fabs(spX); spY = fabs(spY);
  if (group[i].area < 0.0)
    spXY = -spXY;

  spX -= fabs(group[i].area)*pow(group[i].ybar,2);/*SECOND PRODUCTS*/
 spY -=fabs(group[i].area)*pow(group[i].xbar,2); /*WRT THE CENTROID*/
  if (spXY > 0.0)                                             /**/
    spXY-=fabs(group[i].area)*group[i].xbar*group[i].ybar;    /**/
  else                                                        /**/
    spXY+=fabs(group[i].area)*group[i].xbar*group[i].ybar;    /**/
```

```
    if (fabs(spY-spX) > epsilon)                    /* PRINCIPAL ANGLE */
      group[i].pang = (atan(2*spXY/(spY-spX))/2.0) * 180/pi;        /**/
    else                                                            /**/
      group[i].pang = 0.0;                                          /**/
    if (spY < spX)                                                  /**/
      group[i].pang += 90.0;                                        /**/
    group[i].pang = 180.0 - group[i].pang;                          /**/

    } /* group loop */
    return(0);
} /* GROUP_PARAMS */

data_out(outdat)

  char    outdat[8];        /*NAME OF OUTPUT FILE*/
{
  int     i;
  char    outstring[100];
  FILE    *out;

  print_groups();
  print_props();

  out = fopen(outdat,"w");

  fprintf(out,"30 NOITEMS = %d\r",maxgrp);
  fprintf(out,"100 DATA(P(A,5)");

  for (i=1;i<=maxgrp;i++) {
                        /* FACTOR UP GROUP VALUES TO ADD PRECISION */
    group[i].pang = 10*(group[i].pang);                             /**/
    group[i].area = 100*(group[i].area);                            /**/
    group[i].xbar = 100*(group[i].xbar);                            /**/
    group[i].ybar = 100*(group[i].ybar);                            /**/
    group[i].perim = 100*(group[i].perim);                          /**/
    fprintf(out,", %5.0f, %5.0f, %5.0f, %5.0f, %5.0f",group[i].area,
```

```
    group[i].xbar,group[i].ybar,group[i].perim,group[i].pang);
     }
   fprintf(out,")\r");

   fclose(out);
   return(0);
} /* DATA_OUT */

print_groups()

{
   int i,j;

   printf("\n\ngroup\telement\tTYPE\t  X\t  Y\t   R1\t   R2\t R3\n");
   printf("=================================================\n");
   for (i=1;i<=maxgrp;i++){
     for (j=1;j<=groupsize[i];j++)
       printf("%3d\t%3d\t%s %f %f %f %f %f\n",i,j,element[i][j].type,
         element[i][j].X,element[i][j].Y,element[i][j].R1,
         element[i][j].R2,element[i][j].R3);
     printf("-------------------------------------------------\n");}
     return(0);
} /* PRINT_GROUPS */

print_props()

{
   int i;

 printf("\n\nGROUP\t  XBAR\t YBAR AREA PERIMETER  PRINCIPLE AXIS\n");
  printf("=================================================\n");
  for (i=1;i<=maxgrp;i++){
    printf("%3d\t%f  %f  %f  %f   %f\n",i,group[i].xbar,
         group[i].ybar,group[i].area,group[i].perim,
         group[i].pang);
  printf("-------------------------------------------------\n");}
```

```
  return(0);
} /* PRINT_PROPS */
```

Appendix D. Moments of Inertia for a Four-Sided Polygon

Variable Definitions: (see Figure 12)

$A = m_{y1}$

$B = m_{y2}$

$C = y_{start}1 - m_c x_{mid}$

$D = m_{y3}$

$E = y_{start}2 - m_d x_{mid}$

$F = m_{y4}$

$y_1 = Ax$

$y_2 = Bx$

$y_3 = C + Dx$

$y_4 = E + Fx$

Integrating for I_{xx}:

$$I_{xx} = \int y^2 dA = \int_0^{x_{mid}} \int_{y_2}^{y_1} y^2 dy dx + \int_{x_{mid}}^{x_{tip}} \int_{y_4}^{y_3} y^2 dy dx$$

$$= \int_0^{x_{mid}} \frac{1}{3} (y_1^3 - y_2^3)\, dx + \int_{x_{mid}}^{x_{tip}} \frac{1}{3} (y_3^3 - y_4^3)\, dx$$

$$;(y_1^3 - y_2^3) = (A^3 - B^3)\, x^3$$

$$;(y_3^3 - y_4^3) = \left[C^3 + 3C^2Dx + 3CD^2x^2 + D^3x^3\right] - \left[E^3 + 3E^2Fx + 3EF^2x^2 + F^3x^3\right]$$

$$= (C^3 - E^3) + 3(C^2D - E^2F)x + 3(CD^2 - EF^2)x^2 + (D^3 - F^3)x^3$$

$$= \frac{1}{12}(A^3 - B^3)x_{mid}^4 + \frac{1}{3}[(C^3 - E^3)(x_{tip} - x_{mid}) + \frac{3}{2}(CD^2 - E^2F)(x_{tip}^2 - x_{mid}^2)$$

$$+ (CD^2 - EF^2)(x_{tip}^3 - x_{mid}^3) + \frac{1}{4}(D^3 - F^3)(x_{tip}^4 - x_{mid}^4)]$$

$$I_{xx} = \frac{1}{12}\{(A^3 - B^3)x_{mid}^4 + (D^3 - F^3)(x_{tip}^4 - x_{mid}^4) + 4(CD^2 - EF^2)(x_{tip}^3 - x_{mid}^3)$$

$$+ 6(C^2D - E^2F)(x_{tip}^2 - x_{mid}^2) + 4(C^3 - E^3)(x_{tip} - x_{mid})\}$$

Integrating for I_{yy}:

$$I_{yy} = \int x^2 dA = \int_0^{x_{mid}} x^2(y_1 - y_2)dx + \int_{x_{mid}}^{x_{tip}} (y_3 - y_4)x^2 dx$$

$$= \int_0^{x_{mid}} (A - B)x^3 dx + \int_{x_{mid}}^{x_{tip}} \left[(C - E)x^2 + (D - F)x^3\right] dx$$

$$= \frac{1}{4}(A - B)x_{mid}^4 + \frac{1}{3}(C - E)(x_{tip}^3 - x_{mid}^3) + \frac{1}{4}(D - F)(x_{tip}^4 - x_{mid}^4)$$

$$I_{yy} = \frac{1}{12}\{3(A - B)x_{mid}^4 + 3(D - F)(x_{tip}^4 - x_{mid}^4) + 4(C - E)(x_{tip}^3 - x_{mid}^3)\}$$

Integrating for I_{xy}:

$$I_{xy} = \int xydA = \int_0^{x_{mid}} \int_{y_2}^{y_1} ydyxdx + \int_{x_{mid}}^{x_{tip}} \int_{y_4}^{y_3} ydyxdx$$

$$= \int_0^{x_{mid}} \frac{1}{2}(y_1^2 - y_2^2)xdx + \int_{x_{mid}}^{x_{tip}} \frac{1}{2}(y_3^2 - y_4^2)xdx$$

$$= \int_0^{x_{mid}} \frac{1}{2}(A^2 - B^2)x^3 dx + \int_{x_{mid}}^{x_{tip}} \frac{1}{2}\left[(c^2 - E^2)x + (CD - EF)x^2 + (D^2 - F^2)x^3\right] dx$$

$$= \frac{1}{8}(A^2 - B^2)x_{mid}^4 + \frac{1}{2}\{\frac{1}{2}(C^2 - E^2)(x_{tip}^2 - x_{mid}^2)$$

$$+ \frac{2}{3}(CD - EF)(x_{tip}^3 - x_{mid}^3) + \frac{1}{4}(D^2 - F^2)(x_{tip}^4 - x_{mid}^4)\}$$

$$I_{xy} = \frac{1}{24}\{3(A^2 - B^2)x_{mid}^4 + 3(D^2 - F^2)(x_{tip}^4 - x_{mid}^4) + 8(CD - EF)(x_{tip}^3 - x_{mid}^3)$$

$$+ 6(C^2 - E^2)(x_{tip}^2 - x_{mid}^2)\}$$

Appendix E. Moments of Inertia For a Partial Circle

Variable Definitions: (see Figure 13)

Integrating for I_{xx}:

$$I_{xx} = \int_{\theta_1}^{\theta_2}\int_{o}^{r} \sin^2\theta \rho d\rho d\theta = \int_{\theta_1}^{\theta_2}\int_{0}^{r} \rho^3 d\rho \sin^2\theta d\theta \quad ;\theta_1 = \theta_{mid} - \alpha \quad ;\theta_2 = \theta_{mid} + \alpha$$

$$= \int_{\theta_1}^{\theta_2} \frac{1}{4} r^4 \sin^2\theta d\theta = \frac{1}{4} r^4 \left[\frac{\theta}{2} - \frac{\sin^2\theta}{4} \Big|_{\theta_1}^{\theta_2} \right] = \frac{r^4}{16} [2(\theta_2 - \theta_1) - \sin 2\theta_2 + \sin 2\theta_1]$$

$$= \frac{r^4}{16} [4\alpha - \sin(2\theta_{mid} + 2\alpha) + \sin(2\theta_{mid} - 2\alpha)]$$

$$= \frac{r^4}{16} \left[4\alpha - 2\left(\frac{\sin(2\theta_{mid} + 2\alpha) + \sin(-2\theta_{mid} + 2\alpha)}{2} \right) \right]$$

$$= \frac{r^4}{16} [4\alpha - 2\sin(2\alpha)\cos(2\theta_{mid})]$$

$$I_{xx} = \frac{r^4}{8} [2\alpha - \sin(2\alpha)\cos(\theta_{mid})]$$

Moving I_{xx} to zero using the Parallel Axis Theorem:

$$I_{x0} = I_x - \alpha r^2\left(\frac{2r\sin\alpha}{3\alpha}\sin\theta_{mid}\right)^2 + \alpha r^2 y_c^2$$

$$I_{x0} = \frac{r^4}{8}[\alpha - \sin(2\alpha)\cos(2\alpha_{mid})] + \alpha r^2\left[y_c^2 - \frac{4r^2\sin^2\alpha\sin^2\theta_{mid}}{9\alpha^2}\right]$$

Repeating the process for I_{yy}:

$$I_{yy} = \int_{\theta_1}^{\theta_2}\int_0^r \rho^3 d\rho\cos^2\theta d\theta = \int_{\theta_1}^{\theta_2}\frac{1}{4}r^4\cos^2\theta d\theta$$

$$= \frac{1}{4}r^4\left[\frac{\theta}{2} + \frac{\sin 2\theta}{4}\Big|_{\theta_1}^{\theta_2}\right] = \frac{r^4}{8}\left[(\theta_2 - \theta_1) + \frac{\sin 2\theta_2 - \sin 2\theta_1}{2}\right]$$

$$= \frac{r^4}{8}\left[2\alpha + \frac{\sin(2\alpha + 2_{mid}) + \sin(\alpha - 2\theta_{mid})}{2}\right]$$

$$I_{yy} = \frac{r^4}{8}[2\alpha + \sin(2\alpha)\cos(\theta_{mid})]$$

$$I_{y0} = I_y - \alpha r^2\left(\frac{2r\sin\alpha}{3\alpha}\cos\theta_{mid}\right)^2 + \alpha r^2 x_c^2$$

$$I_{y0} = \frac{r^4}{8}[2\alpha + \sin(2\alpha)\cos(2\theta_{mid})] + \alpha r^2\left[x_c^2 - \frac{4r^2\sin^2\alpha\cos^2\theta_{mid}}{9\alpha^2}\right]$$

Integrating for I_{xy}:

$$I_{xy} = \int_{\theta_1}^{\theta_2}\frac{1}{4}r^4\sin\theta\cos\theta d\theta = \frac{1}{4}r^4\left[-\frac{\cos 2\theta}{4}\Big|_{\theta_1}^{\theta_2}\right]$$

$$= \frac{-r^4}{16}(\cos 2\theta_2 - \cos 2\theta_1) = \frac{-4^4}{16}[\cos(2\theta_{mid} - 2\alpha) - \cos(2\theta_{mid} + 2\alpha)]$$

$$= \frac{r^4}{16}[\cos(2\theta_{mid} + 2\alpha) - \cos(2\theta_{mid} - 2\alpha)] = \frac{-r^4}{8}\sin(\theta_{mid})\sin(2\alpha)$$

$$I_{xy0} = I_{xy} - \alpha^2 \left(\frac{2r \sin \alpha}{3\alpha} \right)^2 \cos \theta_{mid} \sin \theta_{mid} + \alpha r^2 x_c y_c$$

$$I_{xy0} = \frac{-r^4}{8} [\sin(2\theta_{mid}) \sin(2\alpha)] + \alpha r^2 \left[x_c y_c - \frac{4r^2 \sin^2 \alpha}{9\alpha^2} \cos \theta_{mid} \sin \theta_{mid} \right]$$

$$x_c = lx + \frac{2}{3\alpha} (lr_1) \sin \alpha \cos \theta_{mid}$$

$$y_c = ly + \frac{2}{3\alpha} (lr_1) \sin \alpha \sin \theta_{mid}$$

Appendix F. Vision System Program Listings

COMPARISON@

```
PROCEDURE COMPARISON@
5 ERASE.OVERLAY
10 SET.PART.BLACK(1)
15 SET.THRESHOLD(1,60)
20 TAKE.PIX(1)
21 DRAW.WINDOW
25 DRAW.ALL.ITEMS
30 NOITEMS=1                                    /*OBJECTS ON MODEL PALLET*/
35 IF (NOITEMS + 1) <> QTY.ITEMS THEN
40   WRITE('INCORRECT PALLET, WRONG NUMBER OF ITEMS')
45   GO TO FINIS
50 END IF
55 A=QTY.ITEMS
60 INTEGER P(QTY.ITEMS,5)
65 DATA(P(1,1),200,0,0,600,1800)          /*PARAMETER DATA FOR REF OBJ*/
70 COUNT=0
100 DATA(P(2,1),200,100,0,0,600,1800)  /*MODEL DATA FOR OBJECT(S)*/
105 INTEGER C(QTY.ITEMS)               /*ARRAY OF IMAGE X CENTROIDS*/
110 INTEGER D(QTY.ITEMS)               /*ARRAY OF IMAGE Y CENTROIDS*/
```

```
115 INTEGER E(QTY.ITEMS)                    /*ARRAY OF IMAGE PRIN.AN.*/
120 H=1
125 DO WHILE H <= QTY.ITEMS
130   C(H)=(C.X(H) - C.X(1))                /*FIND RELATIVE VALUES*/
135   D(H)=(C.Y(H) - C.Y(1))
140   E(H)=(PN.A(H) - PN.A(1))
145   FOR K=1 TO QTY.ITEMS                              /*SET UPPER AND
150      LBA=(P(K,1) * RATIO% * RATIO%/10)/1094           LOWER BOUNDRIES*/
155      UBA=(P(K,1) * RATIO% * RATIO%/10)/1040
160      LBC=(P(K,2) * RATIO%)/1065
165      UBC=(P(K,2) * RATIO%)/900
170      LBD=(P(K,3) * RATIO%)/1065
175      UBD=(P(K,3) * RATIO%)/900
180      LBP=(P(K,4) * RATIO%)/1121
185      UBP=(P(K,4) * RATIO%/949
190      IF (P(K,5) + 44) > 1800 THEN
195          UBPA=ABS(P(K,5) - 1760)
200          LBPA=(P(K,5) - 40)
205      ELSE IF (P(K,5) - 40) < 0 THEN
210               UBPA=(P(K,5) + 40)
215               LBPA=(P(K,5) + 1760)
220           END IF
225      ELSE
230               UBPA=(P(K,5) + 40)
235               LBPA=(P(K,5) - 40)
240      END IF
241      GPAR=0
245      IF (AREA(H)>=LBA) AND (AREA(H)<=UBA) THEN GPAR=GPAR+1
246      IF (C(H)>=LBC) AND (C(H)<=UBA) THEN GPAR=GPAR+1
247      IF (D(H)>=LBD) AND (D(H)<=UBD) THEN GPAR=GPAR+1
248      IF (PERIM(H)>=LBP) AND (PERIM(H)<=UBP) THEN GPAR=GPAR+1
249      IF (E(H)>=LBPA) AND (E(H)<=UBPA) THEN GPAR=GPAR+1
250      IF GPAR=5 THEN
250             WRITE('PART ',H,' IS CORRECT')
255             COUNT=COUNT + 1
260      END IF
```

```
265   NEXT K
270   H=H + 1
275   END WHILE
280   IF COUNT=QTY.ITEMS THEN
285      WRITE('CORRECT PALLET')
290   ELSE
295      WRITE('INCORRECT PALLET')
300   END IF
305   WRITE('?')
310   FINIS:
315   END
```

CALIB@

```
PROCEDURE CALIB@
5 ERASE.OVERLAY
10 SET.PART.BLACK(1)
15 SET.THRESHOLD(1,60)
20 TAKE.PIX(1)
25 DRAW.ALL.ITEMS
30 FOR I=1 TO QTY.ITEMS
35   RATIO%=(ITEM.HEIGHT(I) * 10)/2
40   WRITE(RATIO%)
45 NEXT I
50 END
```

Appendix G. Communications Program Listings

```
/* Serial.c */

This is the terminal program.
The code was supplied by Peter Ibbotson of
Borland and modified for QuickC Version 2.0 and communications with
the GE Optomation II Vision System.


#include        "serial.h"
#include        <dos.h>
#include        <stdio.h>
#include        <conio.h>
#include        <bios.h>
#include        <process.h>

typedef void (interrupt far *intrfunc)();


int SError;
int portbase=0;
intrfunc oldvects[2];
char ccbuf[SBUFSIZ];
```

```
int startbuf=0;
int endbuf=0;
char runcal[]="RUN CALIB@\r";
char runcomp[]="RUN COMPARISON@\r";
char editcomp[]="EDIT COMPARISON@\r";
void put_str(char[]);

/*      this does the hard work (handling interrupts)          */

void interrupt com_int (void)

{ _disable ();
  if ((inp (portbase+IIR) & RX_MASK)==RX_ID)
    { if (((endbuf+1) & 0x1fff)==startbuf)
      SError=BUFOVFL;
      ccbuf[endbuf++]=inp (portbase+RX);
      endbuf&=0x1fff;
    };
  outp (ICR,EOI);
  _enable ();
}

/* this routine returns the current value in the buffer */

int getccb(void)

{ int res;
  if (endbuf==startbuf) return (-1);
  res=(int) ccbuf[startbuf];
  startbuf=(startbuf+1) % SBUFSIZ;
  return (res);
}

void setvects (void)

{
```

```
    oldvects[0]=_dos_getvect(0x0b);
    oldvects[1]=_dos_getvect(0x0c);
    _dos_setvect(0x0b,com_int);
    _dos_setvect(0x0c,com_int);
  }

  void resvects (void)

  { _dos_setvect(0x0b,oldvects[0]);
    _dos_setvect(0x0c,oldvects[1]);
  }

  void i_enable (int pnum)

  { int c;
    _disable();
    c = inp (portbase+MCR) | MC_INT;
    outp (portbase+MCR,c);
    outp (portbase+IER,RX_INT);
    c = inp (IMR) & (pnum==2 ? IRQ3 : IRQ4);
    outp (IMR,c);
    _enable();
  }

  void i_disable (void)

  { int c;
    _disable ();
    c=inp (IMR) | ~IRQ3 | ~IRQ4;
    outp (IMR,c);
    outp (portbase + IER,0);
    c=inp (portbase+MCR) & ~MC_INT;
    outp (portbase+MCR,c);
    _enable ();
  }
```

```
void comon (void)

{  int c,pnum;
   pnum = portbase == COM1BASE ? 1 : 2;
   i_enable (pnum);
   c=inp (portbase + MCR) | DTR | RTS;
   outp (portbase + MCR,c);
}

void initserial (void)

{ endbuf=startbuf=0;
  setvects();
  comon();
};

void comoff (void)

{ i_disable ();
  outp (portbase+MCR,0);
}

void closeserial(void)

{ comoff();
  resvects();
};

/* this outputs a character through the com port */

int SerialOut (char x)

{ long timeout = 0x0000ffff;
  outp (portbase+MCR,OUT2|DTR|RTS);
  /* wait for clear to send     */
  while ((inp(portbase + MSR) & CTS)==0)
```

```
    if ((--timeout)==0) return (-1);
  timeout=0x0000ffff;
  /* wait for outport register to clear          */
  while ((inp(portbase+LSR) & DSR)==0)
    if ((--timeout)==0) return (-1);
  _disable ();
  outp (portbase+TX,x);
  _enable ();
  return (0);
}

/* this routine sets the port */

int SetPort (int Port)

{
    switch( Port ) {
       case 1 : portbase =0x3fb; break;       /* COM1 */
       case 2 : portbase =0x2fb; break;       /* COM2 */
       default: return( -1 );
    }

  return (0);
}

/* this routine sets the speed */

int SetSpeed (int Speed)

{ char c;
  int divisor;
  if (Speed==0) return (-1);              /* avoid dividing by zero */
  else divisor=(int)(115200L/Speed);
  if (portbase==0) return (-11);
  _disable ();
  c=inp (portbase+LCR);
```

```
  outp (portbase+LCR,(c|0x80));                    /* set DLAB */
  outp (portbase+DLL,(divisor & 0x00ff));   /* set divisor  */
  outp (portbase+DLH,((divisor>>8)&0x00ff));
  outp (portbase+LCR,c);
  _enable();
  return (0);
}

/*      This routine sets the LCR          */

int SetOthers (int Parity,int Bits,int StopBit)

{ int temp;
  if (portbase==0) return (-1);
  if ((Parity<NO_PAR) || (Parity>OD_PAR)) return (-1);
  if ((Bits<5) || (Bits>8)) return (-1);
  if ((StopBit<1) || (StopBit>2)) return (-1);
  temp=Bits-5;
  temp|=((StopBit==1) ? 0x00 : 0x04);
  switch (Parity)
  { case NO_PAR : temp |= 0x00; break;
    case OD_PAR : temp |= 0x08; break;
    case EV_PAR : temp |= 0x18; break;
  }
  _disable();            /* turn off interrupts */
  outp (portbase+LCR,temp);
  _enable();             /* turn them back on */
  return (0);
}

/*      This routine sets the ports                    */

int setserial (int Port,int Speed,int Parity,int Bits,int StopBit)

{ if (SetPort(Port)==-1) return(-1);
  if (SetSpeed(Speed)==-1) return(-1);
```

```
  if (SetOthers(Parity,Bits,StopBit)==-1) return (-1);
  return (0);
}

void put_str(str)
char *str;
{
     while (*str != '\0')
           {
           SerialOut(*str);
           str++;
           }
}

/* Open the ports and echo to screen until       */
/* ESCape received                               */

main ()

{ int c;

  if (setserial (COM1,9600,NO_PAR,8,1) ==-1) exit (1);
  initserial();
  printf("You're now in terminal mode.
          Press ESC to quit the program.\n\n");

do {
    if (kbhit()) {
        c = getch();
        if (c==27)
            break;
        SerialOut(c);
        }

    if ((c=getccb())!=-1)
       putchar (c);
```

```
        putchar (c);

    } while (c!=27);

    closeserial();

    return 0;
}

/* MAIN FOR Calibcom.c */

main()

{ int c;

        if (setserial (COM1,9600,NO_PAR,8,1) ==-1) exit (1);
        initserial();
        put_str(runcal);
        closeserial();
        return 0;
}

/* MAIN FOR Comparecom.c */

main(int argc, char *argv[])

   FILE *dats;
   char par_str[maxstr];
   int c;

   if (argc !=2)
       printf("Need cadd.c file name\n");
   else
      {
      if ((dats=fopen(argv[1], "r")) != NULL)
          while (fgets(par_str,maxstr,dats))
```

```
                printf("%s\n",par_str);
            fclose(dats);
        } /* else */
    }  /* if argc */

    if (setserial (COM1,9600,NO_PAR,8,1) ==-1) exit (1);
    initserial;
    put_str(editcomp);
    put_str(par_str);
    put_str(runcomp);
    do {
         if ((c=getccb()) !=-1)
            putchar(c);
        }  while (c !=63);

    closeserial();
    return 0;
} /* MAIN */
```

Vita

Paula Marie Hollenhorst Berg was born on July 15th, 1961 in Rochester, Minnesota. In May 1985, she received a Bachelor of Science degree in Industrial Engineering, with honors, from the University of Wisconsin in Madison, Wisconsin. While at Madison, she was active in Tau Beta Pi and Alpha Pi Mu Engineering Honorary Societies. After working as a Production Operations Planner for McDonnell Douglas in St. Louis, Missouri, she gained admission to Virginia Polytechnic Institute and State School to pursue a Master of Science degree in Industrial Engineering and Operations Research. She is looking forward to joining her husband and cat in California.

Paula M. Berg