NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

THE DESIGN AND IMPLEMENTATION OF AN ADVANCED VISION-BASED ROBOTIC SYSTEM

Yoram Bloch

A Thesis
in
the Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montreal, Quebec, Canada

March 1994

ISBN 0-315-90944-7

Canada

ABSTRACT

The Design and Implementation of an Advanced Vision-Based Robotic System

Yoram J. Bloch

This thesis describes an architecture for implementing an Advanced Vision-based Robotic System (AVRS). The AVRS has been designed and implemented as a replacement for controllers of conventional industrial ("dumb") robotic systems. The main goal of this thesis is to show that the proposed system can be implemented on any 4 degrees-of-freedom manipulator at a very low cost of hardware/software and maintenance. This can result in considerable saving to the industry in upgrading "dumb" robots to "smart" (intelligent) ones.

The new AVRS hardware consists of a master processor PC-AT 486 system operating in a master/slave configuration and a frame grabber card. An approach was developed for the recognition of symmetric (rectangular, square, circular and triangular), approximately flat objects in a 2D environment. A novel approach for translating an object location in camera coordinates to robot coordinates was developed and implemented on the IBM7545 Robot where the most important variables for video-robot calibration can be changed on-line to speed up system integration. An application to the location of ICs, and then using robot "pick & place" operations in real-time, was developed based on the methodologies mentioned above. This low cost system locates 1 object in 0.2 sec., 5 objects in 1.0 sec., and 15 objects in 8 sec.

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor, Dr. R. V. Patel for suggesting the topic of this thesis, for providing me with the freedom to determine the direction of the project, for his continuous support and encouragement and for his confidence in my work during my two years as a graduate student at Concordia.

I would like also to thank my colleagues Nicky Ayoub, Gustavo Vegas, Pierre Chevrier, Guy Gosselin, Dave Chu and Réal Gagnier. Their professional advice made this research more productive and much more interesting.

No acknowledgements would be complete without thanking my immediate family *TAMI, SHERRI, JONATHAN*, my parents *LEON & ROZA* and my parents in-law *VIOLET & DAVID BEN-SAMUEL* to whom this thesis is dedicated, for the continuous support they have provided me through these years of study.

A special thanks goes to LONKA and HENIEK BERLACH in Montreal, Canada, to BASHA and YANEK GELBART in Stockholm, Sweden, and to GENIA and LOVA CHRAKOWSKI in Melbourne, Australia.

Finally, financial support from NSERC is also gratefully acknowledged.

Contents

List of Symbols

angl_err(e)	The camera is not precisely mounted in parallel to the second link implying that the camera window is rotated about the axis of the second link by *angl_err* degrees.
AVRS	Advanced Vision-Based Robotic System.
β_1	The angle between the lines Ref_Point-Tip and Tip-IC Center.
b_angle	The angle between the lines Ref_Point-Tip and Link_2 of the robot manipulator.
b_x, b_y	The pixel indices which represent the Ref_Point in camera coordinates.
c_x, c_y	Represents [interval in mm/one pixel] for 'i' and 'j' axes in camera coordinates.
d_{ct}	Distance of Camera to Table.
d_{ft}	Distance of IC' face to Table.
d_c_ic	Distance between the Ref_Point to IC Center.
d_t_c	Distance between the Ref_Point to Tip.
$e_p(N)$	Desired joint position(N)- actual joint position(N) at a sampling distance N.
$e_v(N)$	$\frac{e_p(N)-e_p(N-1)}{T_s}$.
E_e	End-effector.

i_{c_r}, j_{c_r}	The indices representing a location of *corner* in Camera coordinates.
$i_{c_{r_N}}, j_{c_{r_N}}$	The new value of i_{c_r}, j_{c_r} after taking into account the parallax effect in camera coordinates.
i_{mll}, j_{mll}	The middle point of the longest side of a triangle in camera coordinates.
i_{tfcp}, j_{tfcp}	The first investigated triangle corner point found in camera coordinates.
i_{t_4}, j_{t_4}	The location which decides if the investigated triangle is really triangular.
i_C, j_C	The center of the imaginary (external) rectangle in camera coordinates.
$i_{C'}, j_{C'}$	The center of the enclosed rectangle based on its true corners in camera coordinates.
I_{cp}, J_{cp}	The pixel at the center of the camera coordinates (255,241).
IC	Integrated Circuit.
J1	The angle created between robot 'X'- axis and Link_1.
J2	The angle created between robot Link_1 and Link_2.
L_d	The longest distance between two skeleton points.
k_p	Constant position scalar feedback gain.
k_v	Constant velocity scalar feedback gain.

r	$\approx \frac{1}{2}$ to $\frac{2}{3}$ of the width of the desired rectangle.
Ref_Point	A point chosen as a reference near the End-effector closest corner in camera coordinates.
th	Image threshold value.
t_{pd}	Propagation delay time.
T_s	Robot servo loop sampling period.
X, Y	Robot coordinates.
X_{IC}, Y_{IC}	Location of IC center in Robot coordinates.
X_{E_e}, Y_{E_e}	Location of the End-effector in Robot coordinates.
μ_{bg}	The mean value of the background gray level.
σ_{bg}	The standard deviation of the background gray level.
ϵ	The minimum distance in pixels between objects in camera coordinates.
δ	A maximum distance in pixels within an object.
alpha(α)	The angle formed by the 'i' axis and Ref_Point-IC Center in camera coordinates.
theta(θ)	The angle formed by the 'i' axis and Ref_Point-E_e in camera coordinates.
phi(ϕ)	The angle that converts the absolute distance E_e-IC Center to the proportional movement of the robot e_e in X, Y, in order to move from the current e_e location to the IC Center in Robot coordinates.
gamma(γ)	The angle formed by Ref_Point-E_e and Ref_Point-IC Center in camera coordinates.
$\tau(N)$	The joint torque at a sampling instant N.

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 The Evolution of Industrial Robotic System

The first electrical robot appeared in the early 60s. Since then, robots have successfully performed repetitive work such as assembly, material handling and inspection for continuous high performance production of identical parts in a fixed automation environment. For this kind of repetitive operation, the robot controller is composed of very basic hardware and software with few if any sensors. Controllers for these systems are therefore based on simple processing hardware and unsophisticated control and trajectory generation software. These types of robots are generally referred to as "dumb" robots since they are not responsive to changes in their environment.

The IBM 7545 robot is representative of a large class of industrial robotic systems and can be classified as a "dumb" robot. Although the 7545 system is an older generation IBM system, its kinematic configuration is essentially identical to that of current IBM robotic systems, the only significant difference being the *intelligence* of the controllers. This is why the first step in our research was to produce an (Advanced Robot Controller) ARC by bypassing the manufacturer-supplied controller with a

controller possessing sufficient processing power and versatility to execute various advanced robot control strategies.

However, having a sophisticated controller does not solve the tedious task of precisely identifying "location" from where an object is to be picked up or at which it is to be placed. Since frames and tables are associated with an object location, a little change in its location in robot coordinates could cause serious problems. One approach to overcome this problem is to incorporate visual information. This can be done by attaching a camera to the robot arm and using it as an "eye". The computer analyzes the information from the camera and sends control signals to the robot accordingly. The structure of the system designed to handle this kind of operation should be based on large memory and high speed processing in order to store and process the huge amount of data and work in real time.

The goal in the proposed thesis is to create a very low cost vision-based system based on high processing power and incorporate its operation into the Advanced Robot Controller (ARC) [1] for the IBM 7545 Robot.

1.2 Summary of the Research Work

The solution for resolving the problems mentioned above involved the following tasks:

1. Increase the speed of the ARC by changing its Master from the PS/2-50 to a 486-PC-AT system, with an option to interface to up to 4 gigabytes of Random Access Memory (with the OS2 operating system).

2. Implement additional hardware functions to complete the new ARC structure and create a *robot vision environment.*

3. Mount a *CCD camera* on the robot arm with a *frame grabber* card on the bus of the PC AT 486-System.

The above steps were taken in order to perform various robot vision applications, and to develop an architecture which would support future enhancements. The resulting controller can be programmed for use in an industrial or research environment.

The vision based controller has been designed and implemented based on a Master/Slave configuration where the Slave implements robot servo control and takes over a significant percentage of the computation required. This allows more computationally complex control strategies and pattern-recognition functions for robot vision applications to be executed on the Master at the same time.

The pattern-recognition function implemented, analyzes the camera input and uses special algorithms (not convolution) developed to find precise locations (positions and orientations) in a multiple-object environment with different shapes (triangles, rectangles, squares and circles) and sizes. A special Camera-Robot Calibration algorithm was developed to translate camera coordinates to robot coordinates all in a 2D environment for application involving almost flat objects (for example, electronic components). The proposed system can be implemented on any "dumb" robot composed with 4 degrees of freedom at a very low cost of hardware/software and maintenance.

The Real-Time system implemented recognizes up to 15 dual-in-line ICs (rectangles) in one frame grabber operation, and calculates the path to these ICs within 8 seconds.

Different video-based robotic systems have recently been developed. In [40], the authors describe a vision-based robotic system with two cameras attached to the gripper's end-effector. In order to build a world model it is important to compute

the depth map of the environment. To compute depth, one needs to find out, with reasonable accuracy, the relative transformation between the 2-D image and the referential representing the tool. The solution to this problem is obtained by performing a calibration of the stereo pair of cameras. As a result of this process, one obtains two matrices which express the above mentioned transformations. If the transformation between origins of the two cameras' referentials is known, then the coordinates of any point in the 3D world can be calculated by using its projections on the left and right images. In addition to reference [40], references [20], and [25-28] deal with the calibration issue. Reference [20], describes 3-D robot vision calibration under three different constraint conditions in an eye-on-hand configuration: the calibration object is placed in a pre-determined location, and the camera locates the object from a single frame of measurement. The object location is unknown but the camera is still able to locate the object from a single frame of measurement. The object location is unknown and the camera can only locate certain features of the object. In [25], a calibration matrix is developed to map the image coordinates of an IRI D256 vision processor equipped with a CCD camera directly on to the coordinates for an IBM 7540- SCARA manipulator. The transformation is obtained by training a neural network with a set of one hundred data points which relate two dimensional image coordinates to corresponding two-dimensional robot coordinates. The results demonstrate the ability of neural networks to 'learn' the transformation to reasonable accuracy, and also form the basis for adaptive self-calibration of robot-vision system. Reference [26] describes an approach which has some similarities to ours. A semi-automatic method for calibrating a robot-vision interface is presented. It puts a small work-load on the operator, and requires a simple calibration jig and a solution of a very simple system of equations. In [27], the author presents three classes of extrinsic calibration procedures. All use closed-form solutions. The class A calibration

procedure requires a reference object at a recalibrated location. The class B calibration procedure takes advantage of robot mobility. It requires a reference frame, but not precalibration. The class C procedure, by taking full advantage of both robot mobility and dexterity, requires no reference object but the simplest one - a visible point, which is similar to our approach. In [28], a description is presented on ongoing research to achieve coarse calibration of multiple visual sensors. The emphasis is not on high accuracy calibration, but on real-time adaptive calibration procedures which will enable a mobile robot to successfully navigate in complex environments.

Solutions to pattern-recognition problems are discussed in references [23, 29, 30, 33, 37, 38]. In [23] the authors have attempted to use a neural network as a decision-making system which determines how to move the robot to reach the exact target on the basis of the image acquired by the robot "eye". This function has been taught automatically to the neural network. The total system works as follows: (1) A target object is set at a known position, and the position is taught to the system; (2) The robot moves randomly around the target and the neural network learns the relation between the relative positions and images; (3) After enough learning, the robot can identify the target located at an arbitrary position. Reference [29] describes a robotic vision system where stationary calibrated cameras, mounted at a specific height from the workspace plane are used as position sensors for the location determination of mobile robots. The position of the robot is derived from a shape with a contrasting color drawn on its top through segmentation and detection processes. The authors use a process similar to ours in order to determine the object position; camera calibration, geometric distortions correction, segmentation, parallax correction and scaling. Reference [30] addresses the problem of determining the positions and dimensions of objects for which only shape models are available and the object size is unknown. One application domain for generic object recognition is the handling and sorting of

postal objects. Because metrical information relating object features to one another is not available, the more common feature-based approaches are inadequate. INGEN (inference engine for generic object recognition), uses a data-driven approah to determine the position and size of objects with generic shapes such as parallelepipeds and cylinders. This system successfully recognizes occluded objects in heaps. It also handles scenes which have irregularities in surfaces and edges as well as shadows which are common to postal objects. Reference [33] describes a low cost, real-time, robot vision system. The aim of the project was to automate certain aspects of the catering operating at Heathrow airport. The vision tasks proposed are: use a vision system to determine whether an in-flight meal tray has been correctly loaded by a robot: and develop a working approach to the identification and orientation of randomly arranged recyclable items on a tray returned from use on an airliner. The eventual objective of this task is to use this information to guide a robot to unload the tray automatically. The brief for the vision system has been to produce a versatile low-cost industrial vision system, capable of performing the required inspection tasks, within a limited processing time, and requiring no specialized image processing hardware. The robot-vision system described in [37] is implemented for a mobile robot and a robot manipulator. The intelligent robotic system is capable of: (a) on-line recognition and locating of 3-D objects and obstacles with a single or multiple camera system; (b) vision directed navigation of a mobile robot and (c) optimal task and trajectory planning for a robotic manipulator with obstacle and singularity avoidance capabilities. Another approach for a robot-vision system in the postal service ([30]) is described in [38]. Three dimensional picture processing enables the use of robots for sorting objects. The system, with the aid of a special image processing computer, permits out of order parcel heaps to be separated in real time. As in our system, Halogen and/or fluorescent lights (stroboscopic lighting) makes the vision system significantly

independent of ambient lighting.

System performance is addressed in references [21, 32, 35, 36, 39]. In [21], concepts, implementation, and evaluation of a novel processor for feature extraction are described. The processor is a freely programmable RISC (Reduced Instruction Set Computer) machine with a modified Harvard architecture, designed to operate as a processor in combination with each parallel processor of the multiprocessor robot vision system BVV 3. The processor has been tested in a number of real-world experiments, including road following. In this application it has demonstrated its ability to analyze a TV image in less than 4 ms. Our system analyzes an image which includes one object in 150ms approximately. Reference [32] describes research on Transputer-based machine vision systems. Because of the massive amount of computation, conventional uniprocessor computers cannot fulfill the computational requirements of practical real-time industrial applications. The authors have developed parallel algorithms and special architectures for machine vision purposes. In reference [35], to meet the requirements of robot vision, a multiprocessor system architecture was conceived which is based on a small number of loosely coupled microcomputers, interconnected by two separate bus systems for video data and for inter-processor message exchange. Three generations of vision systems based on this architecture have been built. The first two use only standard microprocessors of moderate computing power, but they have, nevertheless, shown remarkable performance in real-time vision application. The third one uses, in addition, a special processor, similar to a digital signal processor, for further enhancing the performance of the system. In reference [36], the authors describe a high-performance transputer-based robot-vision system that performs real-time tracking of moving objects, real-time optical flow computation, and high-speed depth map generation. The transputer vision board was equipped with three image frame memories, each of which could be used simultaneously for image

input, image processing, and image display. The vision board was also equipped with a standard image compression chip, used as a correlation processor. Reference [39] which is very similar to our application deals with a real time 2D visual sensor used to control in a closed loop, a robot which assembles a workpiece held by its grippers moving in a plane. A fast vision algorithm is designed as the means to measure the position and the dimensions of a circular pattern in less than 40 ms.

Reference [34] describes automatic light source placement for an active photometric stereo system. Since photometric stereo systems normally use multiple light sources fixed to the environment, they cannot avoid shadows caused by surrounding objects. The authors suggest to use a movable light source and to adapt its placement activity to the task environment (like our Halogen light mounted near the robot end-effector).

1.3 Thesis Organization

An introduction to the evolution of industrial robotic systems and summary of the research work has been given in this chapter. Chapter 2 describes the theory behind the implementation of the pattern-recognition function which is later applied to IC recognition and other aspects when dealing with video signal processing. Chapter 3 presents a complete hardware description of the vision-based ARVS and includes the new Master/Slave hardware interface. The Slave's additional functionality and the new lighting environment for present and future applications is also included in this chapter. Chapter 4 describes the implemented software of the pattern-recognition function in our environment. Chapter 5 contains a description of the software for robot motion. Concluding remarks and suggestions for future work are given in Chapter 6.

Chapter 2

VIDEO SIGNAL PROCESSING - THEORY

2.1 Introduction

To find the location of objects in a picture, an initial threshold procedure that distinguishes between the background and the object is applied first, then the convolution operation is applied to obtain data about the boundaries of the object, and then comes the tedious process of line detection, line thinning, line merging, line expansion, and detection of the corners of an object in the case of a square, rectangle or triangle. Another problem arises when there is a need to find out about the boundaries of a circular object, Then a process is applied to determine the circular curve [2].

The convolution process works well when :

1. The object is close to the camera. This results in high resolution (*interval/pixel*);
2. The background signal is distributed equally at the camera input window;
3. The object is well defined (rectangle, square, ... etc.).

The disadvantages of the convolution process appear when :

1. The object is relatively small and is not located near the camera.

2. The object is not well defined.

3. The input window is too wide and there are local changes in the background signal which result in different intensities on the boundaries of the objects.

4. The convolution operator is applied near the corners of the object, and the camera is not located near the object, resulting in large error.

5. The color of the object is not homogeneous.

6. The lighting in the camera environment is not good.

In the proposed algorithm, we shall explain how to establish the correct threshold at the input of an arbitrary window, how to find the number of objects of simple shapes in a scene, and how to locate rectangular, square, triangular or circular objects.

Finally, we will incorporate these concepts into our system in order to recognize the locations (positions and orientations) of dual-in-line IC's (rectangles) so that the robot can "pick" and later "place" them at desired locations.

2.1.1 Thresholding

It is desirable that the intensity going out from the light source will not cause reflection of light from the object. An ideal light source is a fluorescent light where the light source is mounted high above the camera, distributing the light equally around the workspace of the camera. The background material should be white matte to avoid reflection, e.g., a regular white matte paper. Since our practical objects are regular

dual-in-line ICs in black packages, putting them on a white matte background gives excellent results.

When the distribution of the light is different across the workspace we may define a different threshold for each window in the workspace. Furthermore, when the window is too large, the threshold within each segment in the window should be evaluated. The evaluation of $\mu \pm \sigma$ should be performed where μ is the mean and σ is the standard deviation of the gray level of the object.

The worse case should always be considered. If μ_{bg} is the mean value of the background gray level, μ_1 is the mean value of the object gray level and σ_{bg} *and* σ_1 are the standard deviation of the background and the object gray level respectively, then the following relation should be satisfied:

$$\mu_{bg} \pm \sigma_{bg} > \mu_1 \pm \sigma_1 \tag{2.1}$$

This assumes that the background gray level is much brighter than the object gray level. In case that the distribution of the object or the background gray level across the window changes dramatically, different threshold levels should be applied according to each segment of a given window.

2.1.2 Pattern Recognition

A theoretical description of the process to locate a multi-object input such as a disc, a square, a rectangle and a triangle in two dimensions is given below. The prerequisites are:

1. Ideally, the object should be (almost) flat.

2. If the object is not flat, then the distance of the camera to the surface of the object must be much greater than the distance of the surface of the object to the table.

The process is composed of the following stages:

1. Collecting data from an input image and separating it into partial groups.

2. Removing irrelevant data from the memory, leaving only the data that describes the boundaries of each partial group. This process creates an outer skeleton of each object partial group.

3. Joining skeleton partial groups into groups which describe the final skeleton objects in the multi-object environment.

4. Enclosing the skeleton inside a rectangle in order to determine the shape of each object and then calculating the corners, center position and orientation of each object.

Collecting data from an input image

After selecting the correct offset and range for the input image, the objects and the background image should be well defined such that, if $\mu_i, i = 1, ..., n$, are the mean gray levels of the n objects and $\sigma_i, i = 1, ..., n$, are the standard deviations of the gray levels of the n objects, the threshold th is chosen accordingly to the relation

$$\mu_{bg} - \sigma_{bg} > th > \frac{i = 1,..,n}{max}(\mu_i + \sigma_i) \tag{2.2}$$

We assume that each object is well defined in the workspace implying that each object has minimum distance d from any other nearby object. When the image is scanned, the gray level value (gl) at each pixel is compared with the threshold level. For any $gl \leq th$ the location i, j of the pixel in the image is stored in memory. For $gl > th$, the algorithm discards the location of the pixel.

Memory consideration

We have to consider memory constraints because the dynamic memory allocation must not exhaust the available RAM. For example, the window of the image composed of an array of 512 × 480 pixels consists of 245,760 pixels. Storing the whole image in memory, results in 245,760 × 4 bytes (2 bytes for x-axis and 2 bytes for y axis) i.e., 983,040 bytes, which will overload the memory available under DOS.

The user has to define the maximum density of objects possible inside the window image taking into consideration first that the objects have to be at minimum distance from each other. second, the limit that is imposed on the size of the largest object, and finally, that the system must operate in real-time. It has been found that about 15 objects (ICs), 7mm wide by 30mm long can fit into our chosen window. In order to evaluate the dynamic memory allocation needed to store the data, we will assume that the largest worse case object size of a 20 pin dual in line IC' is a rectangle, 7mm wide by 30mm long. If we assume that the whole object has a fixed gray level which satisfies, $gl(i,j) < th$ for each pixel, and the pixel interval ratio is $3.5\ pixels/1mm$ then, the largest worse case number of pixels per object that will be stored in memory is $7mm\ (3.5pixels/1mm) \times 30mm\ (3.5pixels/1mm) \approx 2570$ pixels or 10,280 bytes per object or 154,200 bytes for 15 objects out of 500,000 bytes available for dynamic memory allocation under DOS.

A convenient way to save on memory is, if we choose an ϵ which is the worse case (shortest) distance between any two objects and we choose a δ such that $\delta \ll \epsilon$ which is the distance between two consecutive pixels in the same row (i) within the image stored in memory. For example we can take a worse case minimum distance between any object for which $\epsilon = 18$ pixels and $\delta = 4$ pixels to satisfy $\delta \ll \epsilon$, and save up to 75% on the dynamic memory used as well as computation cost. In this way the maximum dynamic memory allocation occupied by 15 object is 154,200bytes

divided by 4, i.e., 40Kbytes of memory. It should be noted that the first and the last pixels in a line describing part of an object boundary are always kept in memory. The above scheme answers our main demands for running in real-time and using a small portion of memory.

Processing of the Image Data

There are three ways to insert a pixel into a group,

1. Join to the current partial group if we find one pixel in the current group for which the distance to the candidate pixel is less than ϵ.
2. Scan the other partial groups for pixels that satisfy the condition in 1.
3. When 1 or 2 are not satisfied, open a new partial group with the candidate pixel.

In order to reduce computation time, the algorithm keeps in memory only the pixel indices that describe the boundaries (skeleton) of each partial group, such that for every j, store i_min_j, j and i_max_j, j in memory. If the distance of a pixel in one skeleton partial group to a pixel in another skeleton partial group is less than ϵ, then we join the two partial groups together. Doing this, we finally create one group which defines one object. At the end of the last stage, the process outputs groups of pixels that describe object skeletons.

In order to find the features (corners, radii) that describe the shape of an object we enclose each object boundary inside an imaginary rectangle. All the sides of the imaginary rectangles are parallel to the camera window. If we assume that each skeleton of an object is described by n given pixels where each pixel is composed of an (i, j) location, then we define the four lines of the imaginary rectangle as

$$i\_max = \overset{max}{i}\ i \tag{2.3}$$

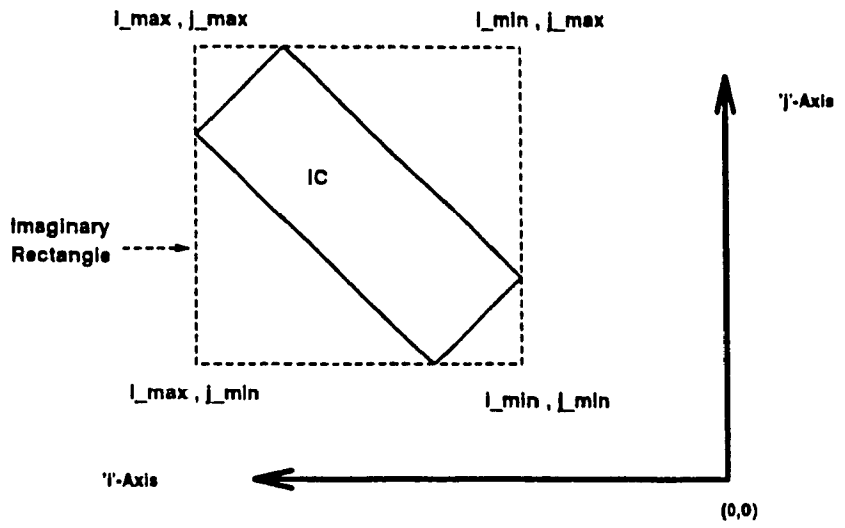

Figure 2.1: The four corner points of the imaginary rectangle.

$$i\_min = \min_{i} i \tag{2.4}$$

$$j\_max = \max_{j} j \tag{2.5}$$

$$j\_min = \min_{j} j \tag{2.6}$$

and the four corner points of the imaginary rectangle are (Figure 2.1):

$$(i\_max, j\_max),\ (i\_min, j\_max),\ (i\_max, j\_min),\ (i\_min, j\_min) \tag{2.7}$$

Both the rectangle enclosed inside the imaginary rectangle and the imaginary rectangle share the same center, which is

$$i_C = \frac{i\_max + i\_min}{2}\ ;\ j_C = \frac{j\_max + j\_min}{2} \tag{2.8}$$

When the center i_C, j_C has been located, then the corner points of the internal rectangle can be located using the following algorithm:

Algorithm 2.1

Input: (i_C, j_C); n skeleton points, (j_k, i_k), for $k = 1, ..n$.

Output: four rectangle corner points, $c_r = 1, .., 4$.

1. Choose an integer $r > 0$ such that $r \approx \frac{1}{2}$ *to* $\frac{2}{3}$ of the width in pixels of the desired rectangle.

2. For $i = 1, .., 4$ do:

   Compute the four farthest points $c_r = 1, .., 4$, from the center (i_C, j_C) among the points on the skeleton using the distance measure

$$\sqrt[2]{(j_k - j_C)^2 + (i_k - i_C)^2}\,, \qquad k = 1, ..., n, \tag{2.9}$$

   such that if (i_1, j_1) is any point of the 4 farthest points, then each of the other three farthest candidate corners (i_{c_r}, j_{c_r}) will have to satisfy

$$\sqrt[2]{(i_{c_r} - i_1)^2 + (j_{c_r} - j_1)^2} > r \tag{2.10}$$

   End of loop.

3. To find the center of an object $(i_{C'}, j_{C'})$ when irregularities are present around the object boundaries but not near corners (IC), use the results in the last step such that,

$$i_{C'} = \frac{1}{4} \sum_{c_r=1}^{4} i_{c_r} \quad j_{C'} = \frac{1}{4} \sum_{c_r=1}^{4} j_{c_r} \tag{2.11}$$

4. Exit algorithm.

Finally, we obtain the orientation of the rectangle relatively to the camera window using

$$\alpha = atan(\frac{i_1 - i_2}{j_1 - j_2}) \quad or \quad \alpha = atan(\frac{i_3 - i_4}{j_3 - j_4}) \tag{2.12}$$

The width and length of the rectangle are given by

$$\| i_1, j_1 - i_2, j_2 \| \approx \| i_3, j_3 - i_4, j_4 \| \quad \| i_1, j_1 - i_3, j_3 \| \approx \| i_2, j_2 - i_4, j_4 \| \tag{2.13}$$

If additional objects shaped like circle or square exist in the camera workspace, then a fifth sample (i_5, j_5) is taken which is the farthest distance from (i_C, j_C) and satisfies $r \approx \frac{1}{3}$ (the width of the rectangle). If for $c_r = 1,..,5$,

$$\| \; i_C, j_C - i_{c_r}, j_{c_r} \; \| \; \textit{is Constant} \tag{2.14}$$

then the object under investigation is a *circle*. If,

$$\| \; i_C, j_C - i_{c_r}, j_{c_r} \; \| \; \textit{is not Constant} \tag{2.15}$$

then the object under investigation is a *square* or a *rectangle*. If the width is equal to the length, i.e.,

$$\| \; i_1, j_1 - i_2, j_2 \; \| \approx \| \; i_1, j_1 - i_3, j_3 \; \| \tag{2.16}$$

then the object is a *square*. Otherwise the object is a *rectangle*. The approach to locate the corners of a triangle inside an imaginary rectangle is a little bit different. The *rule* is that at least one of the external rectangle corners should be located at one of the corners of the enclosed triangle. In practice, to find such a point, we compute the distance of each point $k=1,...,n$ on the skeleton of the enclosed triangle to the four imaginary rectangle corners. If we define the four corners of the imaginary rectangle as (i_{c_r}, j_{c_r}), $c_r = 1,..,4$, then, we look for a (i_k, j_k) point such that, $\| \; i_{c_r}, j_{c_r} - i_k, j_k \; \|$ is close to zero, and this is the first corner of the triangle. If we define L_d as the longest distance between two skeleton points then, assuming that the given point is the first corner point i_{cp}, j_{cp} of the triangle, then the second corner point is located at a point (i_k, j_k) that satisfies,

$$\| \; i_{cp}, j_{cp} - i_k, j_k \; \| = \; L_d \tag{2.17}$$

The third corner point is found by computing the longest distance of any triangle skeleton point to the line created by the first and second corner points already found.

If we define a line generated by the first and second points as

$$aI + bJ + c = 0 \tag{2.18}$$

and the maximum distance L_d of point from that line is defined on the skeleton image as (v, w) then, that point must satisfy,

$$\frac{av + bw + c}{\sqrt[2]{a^2 + b^2}} = max\ L_d \tag{2.19}$$

for any point on the skeleton image (v_k, w_k), $\quad k = 1, .., n.$

To check that the object is a triangle, we have to take the following steps:

1. Find the longest line between the three corners which compose the possible triangle.

2. Compute the middle point of the longest line (i_{mll}, j_{mll}).

3. Choose $r = \frac{1}{3}$ x [shortest distance between two assumed triangle corners].

4. Choose the farthest point (i_{t_4}, j_{t_4}) on the investigated skeleton image from i_{mll}, j_{mll} such that,

$$\| i_{c_r}, j_{c_r} - i_k, j_k \| > r, \quad c_r = 1.., 3 \tag{2.20}$$

5. If the farthest point (i_{t_4}, j_{t_4}) is located on one of the triangle lines, then the object is a triangle.

   If (i_{t_4}, j_{t_4}) is not on one of the lines, then the object is not a triangle.

More general way to check that the object is a triangle is as follows,

1. Choose $r = \frac{1}{3}$ x [shortest distance between two of the three assumed to be triangle corners].

2. Look for a second farthest point using again equation 2.19 when the third assumed corner is blocked by r. If the second farthest point is not on one of the lines, then the object is not a triangle.

2.2 From Theory to Practice

2.2.1 Pattern-Recognition of Multiple Objects (Shapes & Sizes)

The proposed algorithm was designed to give fast answers using an inexpensive real-time system concerning the location (position & orientation) of objects in the work space. If a particular object of a given size needs to be found, then at the input level, the size of the object should be specified. Furthermore, if the object sizes differ significantly, then the size of the window of the camera should be changed to accommodate the smallest size that might appear in the workspace. This means that the interval/pixel ratio should be made smaller.

When working in real-time in a multi-object environment, we need to consider the following points:

1. Choose a window that will contain as many objects as possible with an acceptable level of accuracy.

2. As the size of the window increases, the accuracy of the result (object location) decreases.

3. If one intends to place an object accurately in a given location, then a second camera should guide the object to the exact location. That camera should be mounted near the robot E_e (End-effector) so that a close-up image which

contains the object and the final location can be obtained. In this way a surface mount or dual-in-line IC can be inserted onto a printed-circuit board when the first camera supplies the approximate location of the target and the close-up camera makes adjustments of the position and orientation of the IC to allow smooth insertion.

4. If very high accuracy is not needed, then only one camera is sufficient for pick and place operations.

2.2.2 IC Recognition

A question that arises when enclosing a rectangle by another is whether the sides of the external rectangle always pass through the corners of the internal rectangle. In practice, this is not always possible to ensure. The reason is that objects do not always have smooth shapes. When the sides of the internal rectangle are parallel or almost parallel to those of the external rectangle (one rectangle overlapping another), but the sides of the internal rectangle are not smooth, then the size of the external rectangle will depend on the structure (data) of the sides of the internal rectangle and not on its corners. This means, that the external rectangle sides may move up, down, left or right. So, finding the center of the external rectangle does not always supply us with the exact center of the enclosed rectangle, but will surely lead us to somewhere around the center of the internal rectangle.

This assumption is used when we look for an approximate center of a dual in line IC, since in some cases, parts of pins around the sides of the IC but not at the corners may effect the final result. This is why in Algorithm 2.1 we use information about the corners rather than the sides.

Remarks

The corners of an IC are not subject to irregularities for the following reasons:

1. In most ICs, a space exists between the last pin on the row and the corner.

2. In most ICs, the pin near the corner is very thin and does not protrude as the others do.

3. Since the color of the pins (silver) is much brighter than the color of the package (black), fixing the offset and the range of the picture will eliminate most of the effects of the pins on the rectangular package. The four pins that are near the four corners will never appear in the picture if appropriate lighting is used. The effect of the other pins is negligible in this location.

4. The system is structured to reduce any errors that may occur. Let us assume that the pin at one of the corners affects the final result, and that due to an error, the corner is defined $1mm$ from its true location. Since the center of the rectangle is found by summation of the four i's and j's and division of the result by four, the error is divided by four as well and the final computed deviation from the center of the IC is $0.25mm$. The system keeps the error low when computing the orientation of the IC. As an example, for a 14 pin package, size $6mm \times 19mm$, moving one corner $1mm$ from its true location, the orientation error on one side is 3°. On the other side of the package, the error is 0°. The average worst case orientation error will be 1.5° which is less than the robot worst case error of 3°.

5. The worst case error situation occurs when only one corner of the IC is affected in the image by one pin near this corner. Since, in the image the two farthest

Horizontal Window			Vertical Window		
1	1	1	1	0	-1
0	0	0	1	0	-1
-1	-1	-1	1	0	-1

Table 2.1: Sobel convolutions.

corners from the center of the image will always be ahead of the pins close to these corners, the pins will never have any effect on the farthest corners of the IC. Looking at the other two corners of the IC, if the pin beside each corner is seen by the camera, then the effect on the error approaches zero. When only one pin at the corner is seen by the camera then, as explained here and in the preceding paragraph, this is the worst case.

6. It is now easy to understand why using convolution is not the best approach for our application. In summary, the reasons are as follows:

   (a) Irregularities caused by the pins around the IC will supply false information about the boundaries of the IC package.

   (b) The error around the corners of the IC will be high as a result of convolution.

   (c) In an image input, when two ICs are near each other, and parallel to each other, the algorithm may see two ICs as one. That is why using the data inside the IC and not only at the boundaries, gives a better indication about the structure of each object in such an environment.

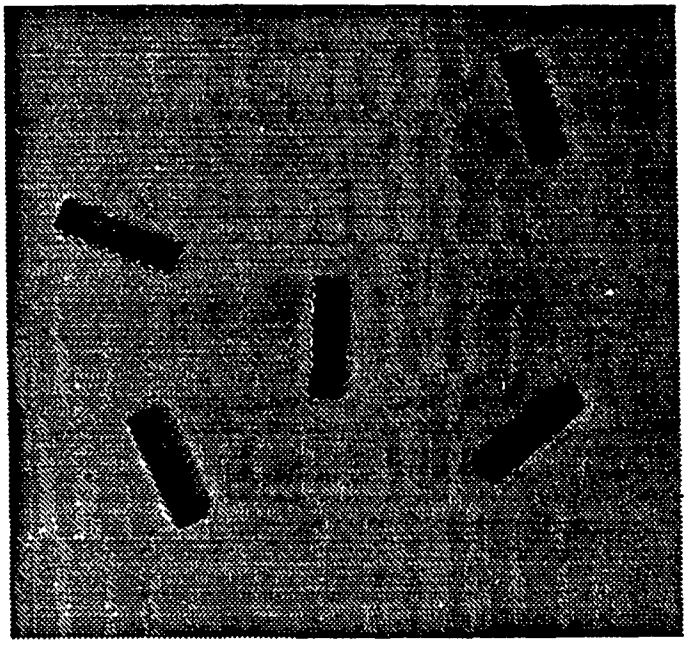

Figure 2.2: The original picture taken by the frame grabber.

Figure 2.3: Horizontal Sobel convolution applied to the original picture.

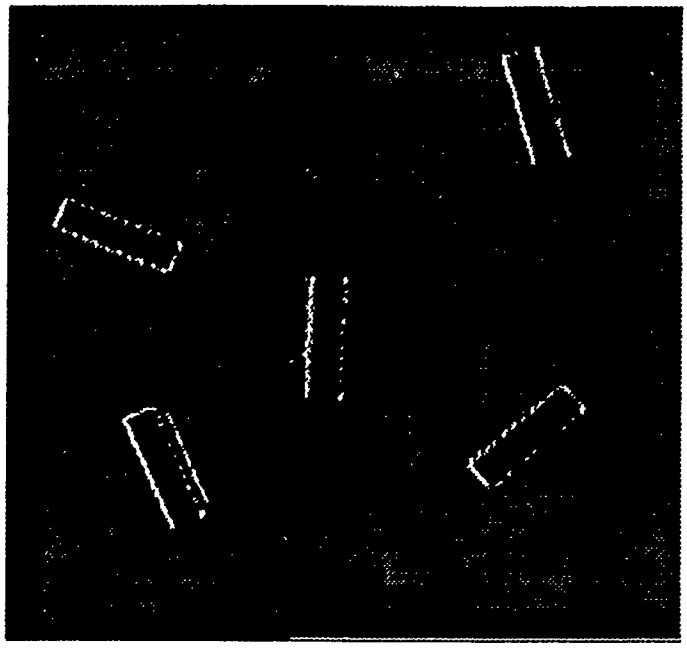

Figure 2.4: Vertical Sobel convolution applied to the original picture.

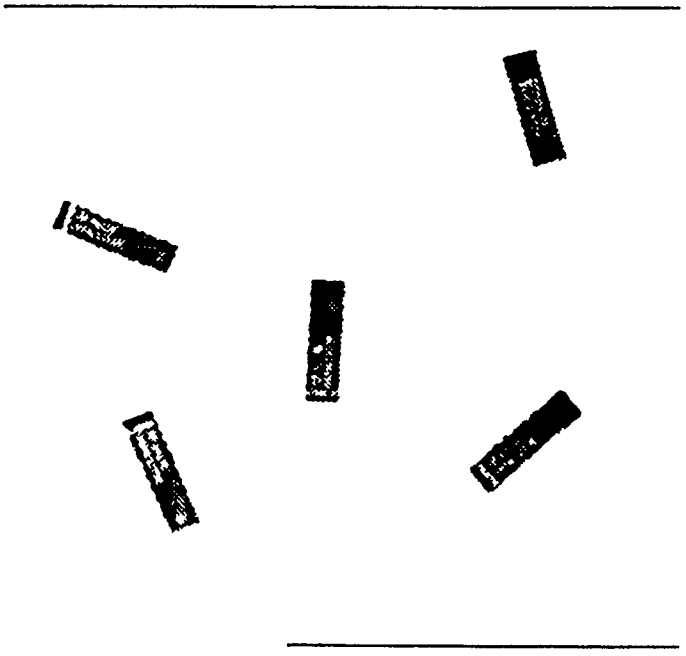

Figure 2.5: The original picture after adjustment of range and offset.

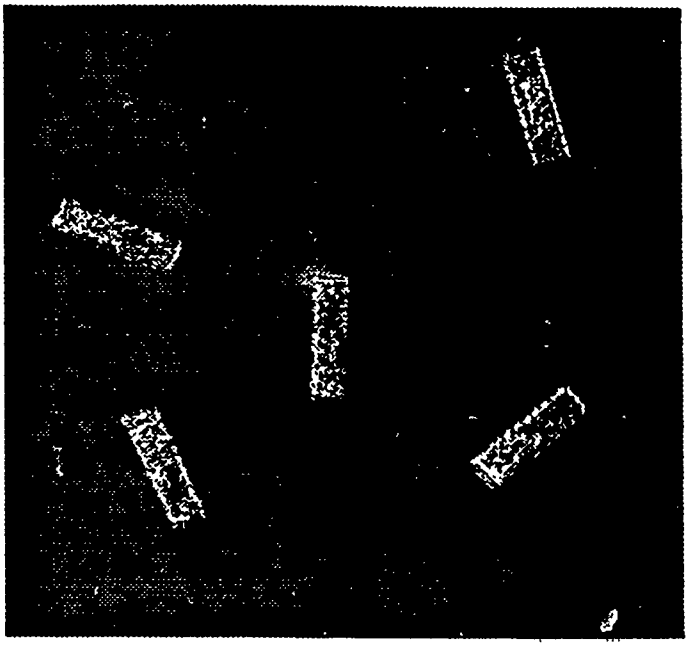

Figure 2.6: Vertical Sobel convolution applied to the adjusted picture.

From Figures 2.2 - 2.6, it is easy to understand why Sobel convolution (edge detection) was not used. Figure 2.2 is the original picture taken by the frame grabber, and Figures 2.3 and 2.4 are Horizontal and Vertical Sobel Convolutions (See Table 2.1) performed on Figure 2.2. As can be seen, the results are affected by inaccuracy due to dark spots created by the IC package reflection caused by the fluorescent light, pins that directly affect the boundaries of the objects, and missing data on edges of the boundaries. Figure 2.5 shows much better results after offset and range adjustments were applied to the original image (Figure 2.2). This results in no spots, little effect of the ICs pins on the shape of the rectangle, and no noise around the corners of the rectangle. Figure 2.6 shows the result when implementing Vertical Sobel Convolution on Figure 2.5. Due to the image being very "busy"[5] (high mixture of black and white pixels reflected from the described object and created by image offset and range adjustments) in Figure 2.5, bad edge detection as well as inaccurate results occur.

2.2.3 Parallax Effect in Determining Locations of ICs

When the camera is located far away (high ratio of camera-table distance to object surface-table distance) from almost flat objects, and the size of the window is small (small ratio of 'center of image-corner of image distance' to 'camera-table distance'), then the *parallax effect* is negligible. If the window of the camera is large, and the camera is located in a medium/short distance of an almost flat object, then the parallax effect should be evaluated. The ratios are chosen according to the demanded precision of the application.

If we assume that the table is parallel to the camera window (lens), and that the object is shaped like a box (IC), then we have to take the following steps:

1. Measure the distance from the camera to the table, d_{ct}.

2. Measure the distance from the object (IC) surface to the table, d_{ft}.

3. If the four corners of the IC in the image are defined as (i_{c_r}, j_{c_r}), $c_r = 1, \ldots 4$, and the indexed value of the pixel at the center of the image is (I_{cp}, J_{cp}), then the values of the true new indices of the corners i_{cr_N} and j_{cr_N} are given by

$$i_{c_{r_N}} = i_{c_r} + (I_{cp} - i_{c_r})\frac{d_{ft}}{d_{ct}} \tag{2.21}$$

$$j_{c_{r_N}} = j_{c_r} + (J_{cp} - j_{c_r})\frac{d_{ft}}{d_{ct}} \tag{2.22}$$

It should be noted that $i_{c_{r_N}}$ and $j_{c_{r_N}}$ will in general have real values which may be approximated to the closest integers.

Chapter 3

HARDWARE DESCRIPTION FOR THE VISION-BASED ROBOTIC SYSTEM

3.1 Introduction

The vision-based robotic system consists of the following components:

1. The camera [16] is manufactured by HITACHI, a color NTSC standard $\frac{2}{3}$ inch single layer MOS color image sensor with 760 x 485 pixels.

2. The frame grabber card OC-300 [15] developed by CORECO INC., is based on the *PC-AT* bus interface, and is able to read and process an image composed of 512 x 484 pixels. The OC-300 is a high speed frame grabber which is able to grab 30 images per second.

3. The robotic system consists of an IBM 7545 [12] robot with a teach pendant. The robot is of SCARA (Selective Compliance Assembly Robotic Arm) type

and has four degrees of freedom. A high speed master controller based on IBM 486-AT system [13], processes the frame grabber card output, performs pattern recognition on the picture grabbed by the frame grabber, and sends information to the controller (a slave based on the EV80C196KA Evaluation Board [14]) of the robot concerning the path that the robot should take in order to perform its task. The slave takes over a significant percentage of the computations required for the servo control, and thus allows more computationally complex control strategies to be executed on the master. The latter was initially based on an IBM PS/2 model 50 computer.

3.1.1 Integration

To integrate between all the components of the system, process the data in real-time and provide a better lighting environment the following steps were taken:

1. The *PS/2* was replaced by a *PC-AT* computer system based on the INTEL 486 micro-processor. This provided very high speed execution compared to the *IBM-PS/2* model 50. Also the frame grabber card available was compatible with an *AT-BUS*. The 486 based system was chosen to perform as a new master for the Advanced Robot Controller in a *Master/Slave* configuration.

2. A new interface card between the Master PC-AT 486 and the slave EV80C196KA was designed and implemented. It should be noted that the PS/2 and the PC-AT have completely different buses.

3. A better lighting environment was created by the following actions:

   (a) The master controls a group of three fluorescent lights whenever the balance of lighting is not satisfactory.

(b) The master uses a relay [18], and is able to control a Halogen light which is mounted next to the camera. This will be particularly useful in case (for future applications) it is required to distinguish between colors.

4. A special hardware function was added to control the gripper at the E_e of the robot using the new Master/Slave configuration. This was not available under the PS/2 based system.

5. A special effort was made to avoid any change in the Slave interface to the robot hardware and to the Dual-Port RAM [17].

6. Special care was taken to install the camera:

   (a) The camera was installed precisely horizontal to the given workspace near the E_e.

   (b) A special frame was designed to avoid any movement of the camera relative to the robot during the motion of the robot.

The implemented system is shown in Figure 3.1. A picture of the *IBM7545* robot manipulator with the camera and the halogen light mounted on the robot arm is given in Figure 3.2. The master *486-SYSTEM* sitting on top of the robot controller is shown in Figure 3.3, and the *RGB monitor*, *PC-XT* and the *Power Supply* for the *Slave-EV80C196KA Board* are shown in Figure 3.4.

3.2 Master Hardware Interface Functions

3.2.1 Master/Slave Interface

Special care needs to be taken when designing the hardware interface functions of the Master PC-AT 486 System. If, for example, one of the functions generates a busy

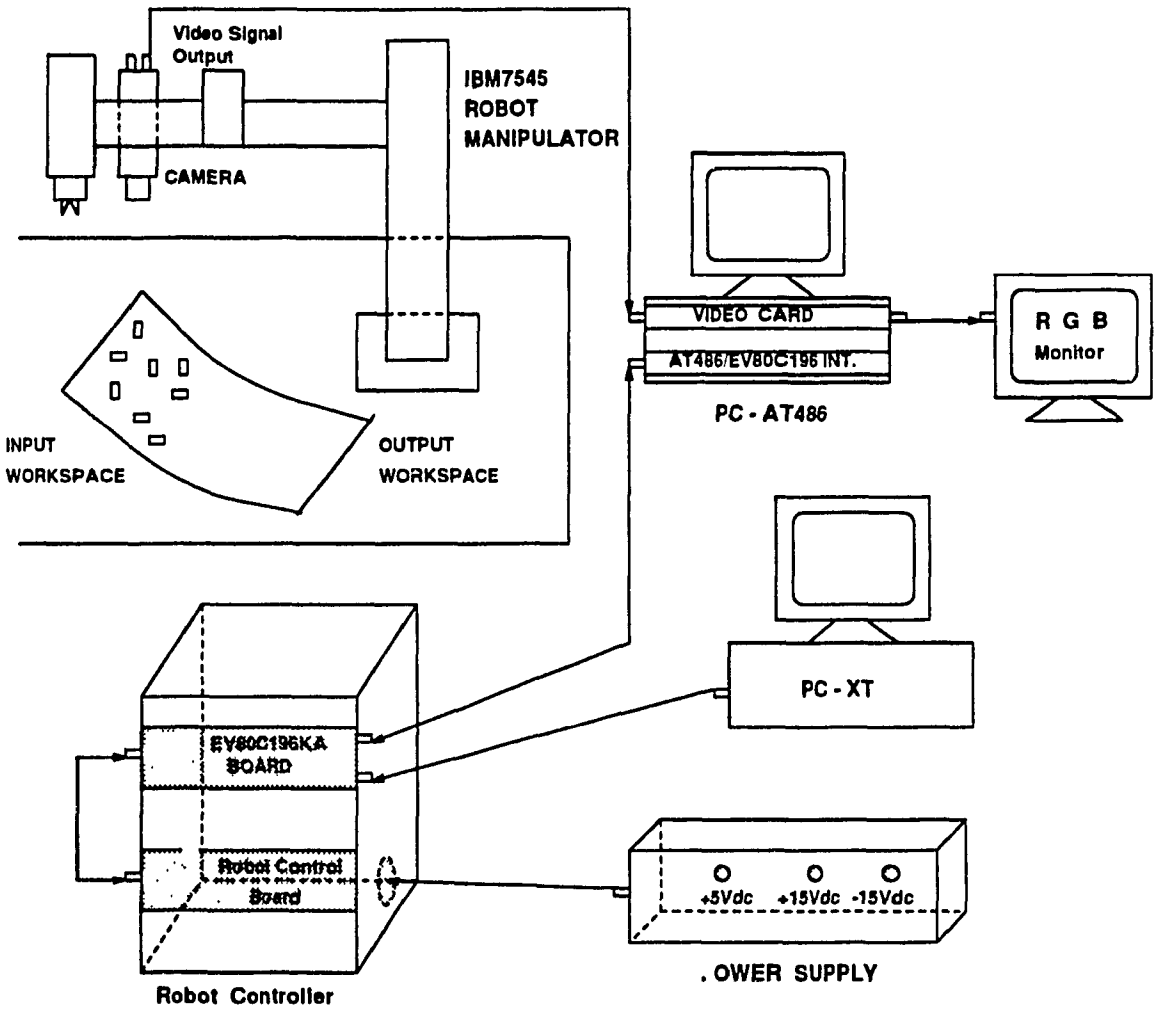

Figure 3.1: Description of the Vision-Based Robot Manipulator System.

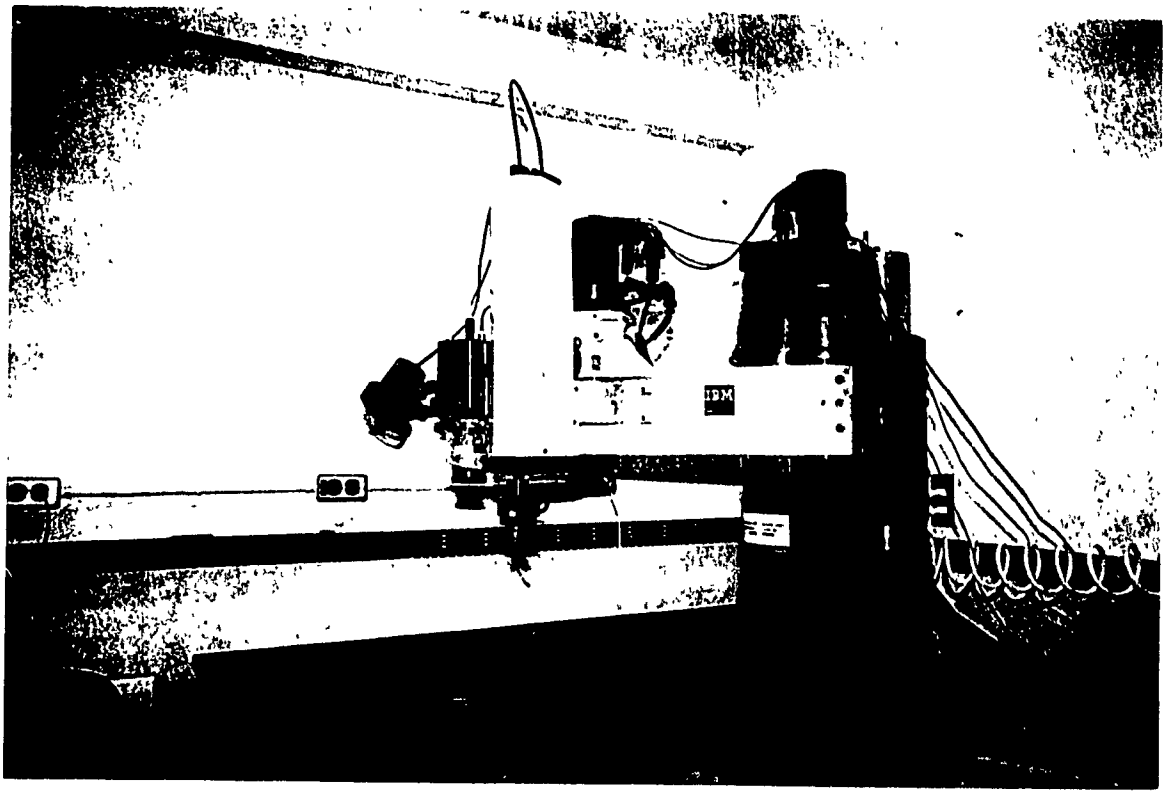

Figure 3.2: The *IBM 7545* Robot Manipulator with the CCD camera and the halogen light mounted on the robot arm.

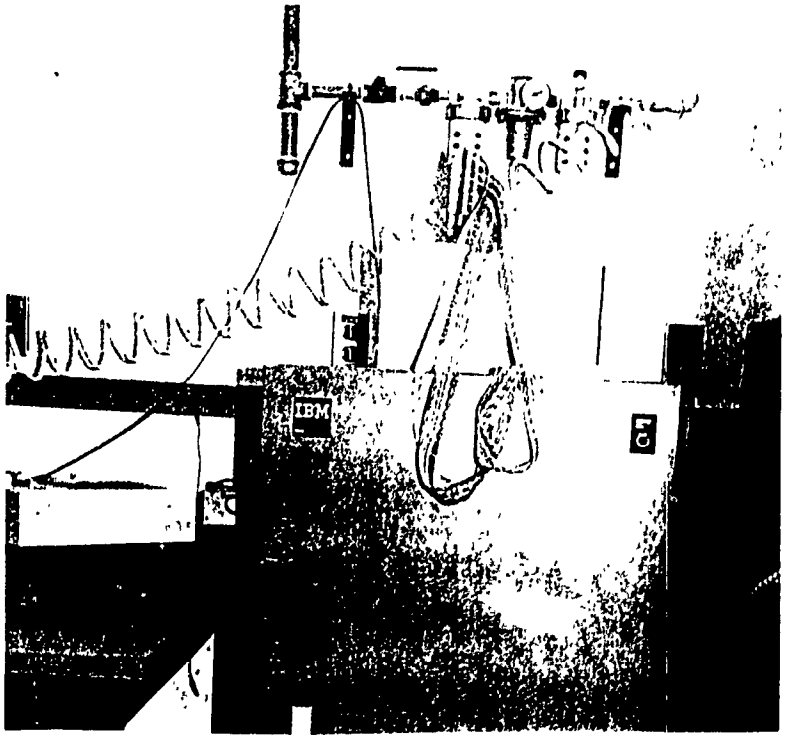

Figure 3.3: The Master *486-SYSTEM* with the interface Master/Slave board.

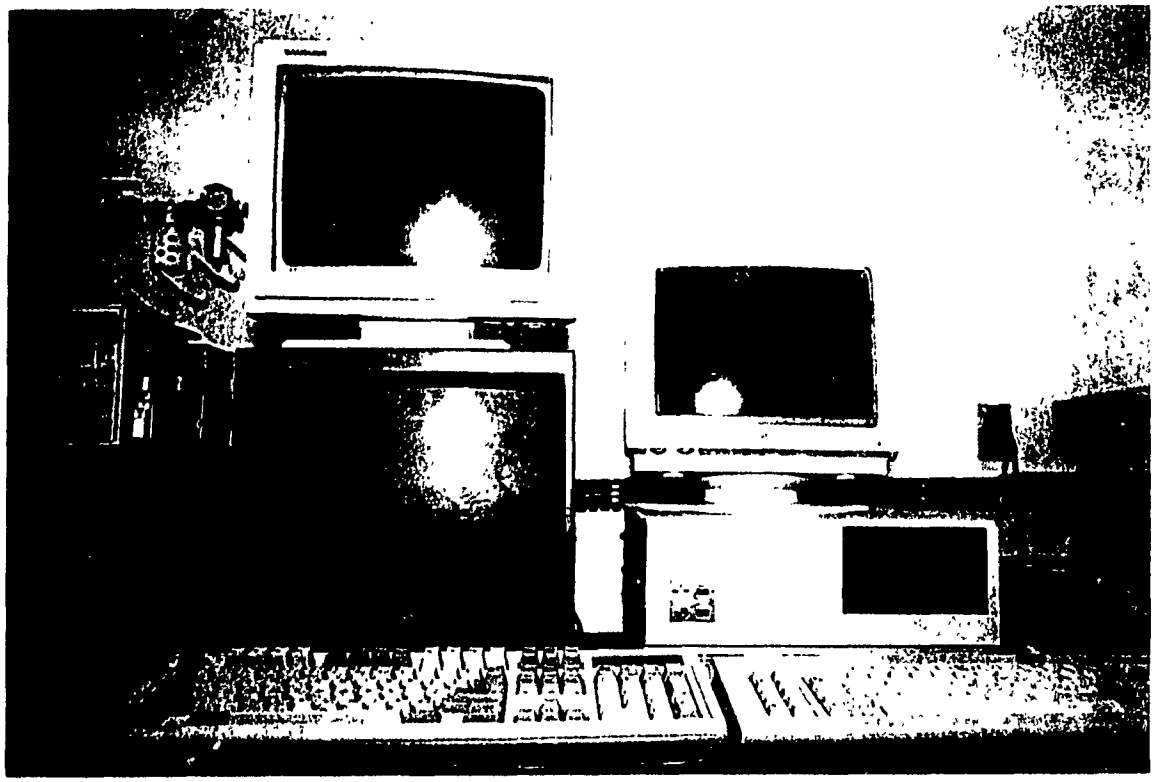

Figure 3.4: The RGB monitor, PC-XT and the power supply for the EV80C196KA Board.

signal for the Master, then the interface should advance the signal fast enough so that it can reach the Master on time to avoid a system crash. For this reason, the best solution is to use the Bus-Interface ICs based on the ALS-TTL group, and where possible complementary ICs based on the AS-TTL group which are the fastest ICs in their group (TTL) at present. The average worst case of signal propagation delay time t_{pd} for ALS-TTL Bus Interfacing Group is 20nsec, and for the AS-TTL Group is 6nsec. Special attention was given to simplifing the functions to reduce signal delays.

The schematic diagrams that describe the Master/Slave interface and the lighting environment function are given in Figures 3.5 and 3.6.

The PC-AT Bus is composed of 24 address lines. The seven most significant bits LA17-23 (Unlatched addresses 17-23) are the output signals used to provide memory

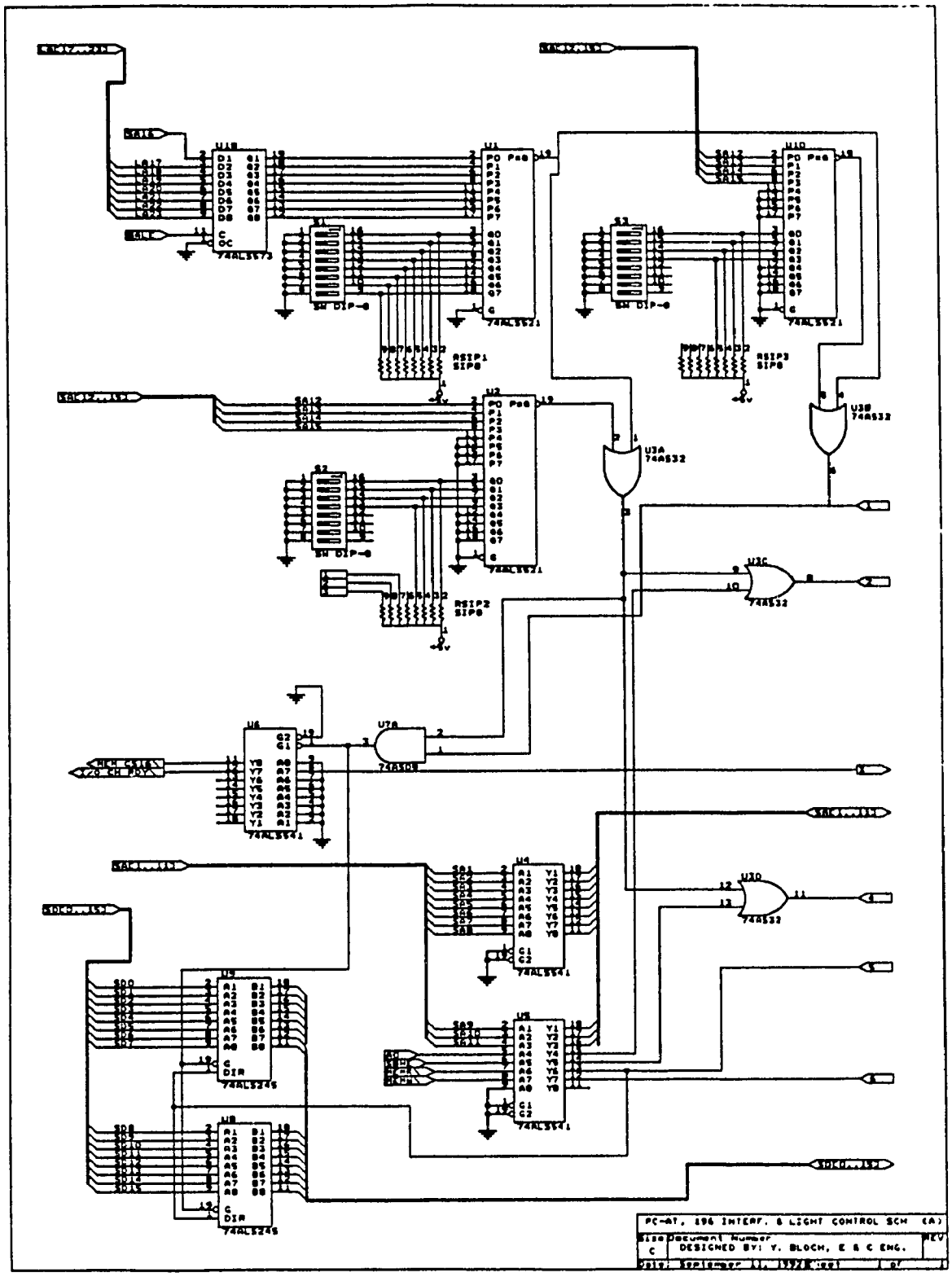

Figure 3.5: The schematic diagram of the Master/Slave interface and the lighting environment function Part-A.

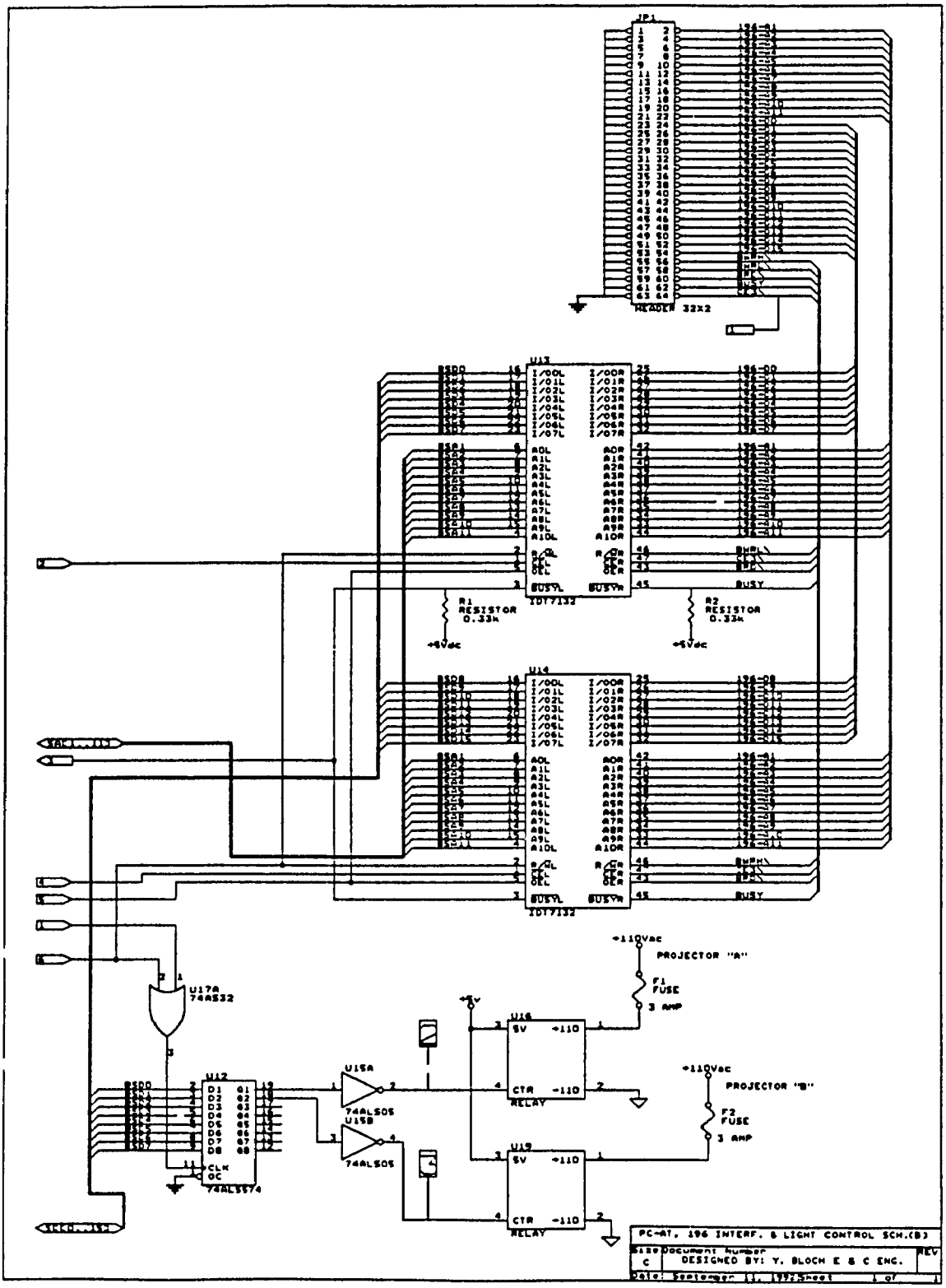

Figure 3.6: The schematic diagram of the Master/Slave interface and the lighting environment function Part-B.

address information about the present bus cycle. These addressing signals, unlike SA0-SA20, are not valid during the entire bus cycle. They become valid during the actual time of the BALE bus signal and can be latched on the falling edge of the BALE signal. We use the above technique with IC U18 (74ALS573) to latch the LA17-23 so that the address will be available at the output of the U18 during the whole bus cycle.

For the eight bit identity comparator IC U1, the Dip switch S1 was initialized to recognize the two most significant hexadecimal addresses *0Cxxxx*. There are two additional Dip switches on the interface board. S2 and S3 initialize the comparators U2 & U10 respectively for the third most significant hexadecimal address. If the hexadecimal address *0C8xxx* is released by the Master then the combination of U1 & U2 will control the $\overline{CE}$ terminal in U13 & U14 of the Dual-Port-Ram. If the hexadecimal address *0C9xxx* is released by the *486* then the combination of U1 & U10 will control the lighting environment function.

If the output signals of U1 and U2 are low then it enforces the signal at the output of U3A to become low. That signal, together with the two control signals A0 and SBHE (System Bus High Enable) decode at the output of U3c and U3d the type of bus cycle shown in Table 3.1.

The control signals of the 486 microprocessor $\overline{MEMW}$ and $\overline{MEMR}$ control the $R/\overline{WL}$ and $\overline{OEL}$ input terminals of the Dual-Port-Ram (DPR).

Special care was taken for the $\overline{I/O\ CH\ RDY}$ signal which indicates to the Master processor that the Dual Port Ram is busy serving the slave at the same required address. To cause the busy signal released by the DPR to reach the 486 terminal as soon as possible the output of IC U3A is connected to the IC of U7A which enables IC U6 the Octal Buffer with 3 State outputs. When the busy signal generated by the DPR changes from high to low on IC U13 pin 3, the way through IC U6 to the 486

SBHE Value	*A0 Value*	*Function*
0	0	16 Bit Transfer
0	1	Lower Byte Transfer
1	0	Upper Byte Transfer
1	1	Invalid

Table 3.1: Data bus size operation decoding of the Master.

IC no.	t_{pd}
U18	14 $\eta sec.$
U1	20 $\eta sec.$
U3A	6 $\eta sec.$
U3C	6 $\eta sec.$
U13	35 $\eta sec.$

Table 3.2: t_{pd} of ICs involved in the generation of the *Busy* signal.

terminal is already open.

For the worst case scenario, we have calculated the propagation delay time t_{pd} from the point where the *signal address* arrives at the (Master/Slave) interface board until the *busy* signal leaves the interface board. The data used was supplied by *TI* and is given in Table 3.2. The value of t_{pd} was found to be 81nsec. It should be noted that the worst case t_{pd} of the DPR (U13) and U3C is 41nsec. which is 15nsec. more than that of U6 (20nsec.) and U7A (6nsec.) so, as stated above, the busy signal released by the DPR does not cause a delay at IC U6.

Address	*D0*	*D1*	*Function*
0C9000	1	0	Halogen Lights On
0C9000	0	1	Fluorescent Lights On
0C9000	1	1	Fluorescent & Halogen Light On
0C9000	0	0	Fluorescent & Halogen Light Off

Table 3.3: The lighting system controlled by *D0 & D1*.

We used our *Logic Analyzer* to evaluate the timing of the bus cycle. It was found that the particular bus cycle to the DPR address is over *200nsec.*, which is sufficient time for the busy signal to reach the PC terminal. So, the additional generation of a wait state is not needed.

3.2.2 The Lighting Environment Function

When instructions to control the lighting environment are generated by the PC, comparators U1 and U10 outputs are low; therefore, the output of the OR gate at IC U3B is low. When that signal arrives at U17A pin 1, at the rising edge of $\overline{MEMW}$ signal, the output of U17A changes from low to high latching the data on the Data Bus into the D-TYPE Edge-Triggered Flip-Flop. With open collector drivers U15A/B at the output of the D-Type Flip Flop, sufficient current is driven into the solid state relays U16/19 input loop in order to activate the 110v lighting system. The lighting system is controlled by the value of bits *D0 & D1* arriving at IC U12 as shown in Table 3.3.

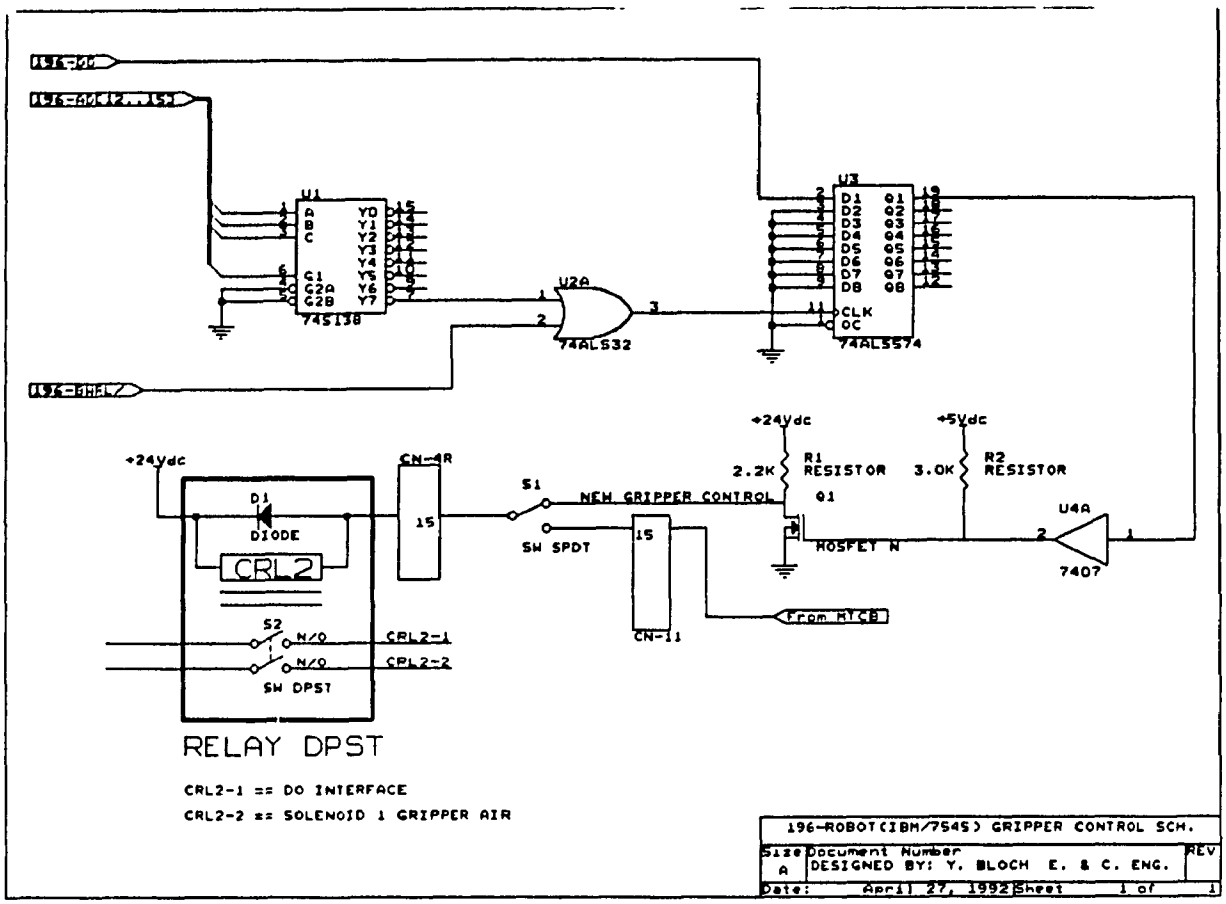

Figure 3.7: The schematic diagram of the Slave and E_e interface.

3.3 Hardware Design of the Slave and E_e Interface

The schematic that describes the Slave and E_e Interface is given in Figure 3.7. To create a hardware interface from the slave (EV80C196KA) microcontroller to the gripper, we used a free terminal on IC U1 *Pin 7* (figure 3.7) at the control unit level of the slave, and thereby avoided adding a new decoder to our hardware design.

When the slave sends the value of *0F000H* on the address bus, the 3-line to 8-line decoder is activated, i.e., $Y7$ output changes from high to low. Since the interface to the gripper is *write only*, we combine the $Y7$ signal with the slave $\overline{BWRL}$ (Bus Write

Q1	*Driver Output*	*N-Mosfet Output*	*Relay*	*Gripper*
High	5V	0V	Active	Close
Low	0V	24V	Non-Active	Open

Table 3.4: The mechanism which controls the *lighting system*.

Low) control signal into U2A *OR Gate* whose output becomes low when the above signals are low.

The next stage is IC U3 Octal D-Type Edge Triggered Flip Flop where *D0* of the Data Bus of the slave is connected to *D1* of U3. On the rising edge of $\overline{BWRL}$ the value on *D0* is latched into IC U3. The output signal of IC U3 is *Q1*, which is connected to U4A an open collector driver, which keeps a stable voltage level output 0V or 5V at the input of an N-MOSFET. If the input voltage is 0V, then the Mosfet channel is closed and the voltage at the output of the Mosfet is 24V. Thus, the relay is non-active and the gripper is open. If the input voltage to the Mosfet is 5V, then the Mosfet channel is open. The voltage at the output of the Mosfet is 0V. The relay is active and the gripper is closed. Table 3.4 describes this process.

The connection between the new interface and the gripper relay is through an *SPDT* switch at the entrance to the controller relay board, connector CN-11 pin 15 [12].

Chapter 4

DESCRIPTION OF THE VISION-BASED SOFTWARE

4.1 Introduction

As mentioned in the previous chapters, the Master processor controls the whole system directly. The functions that it performs are as follows:

1. It interfaces with the user.
2. It initializes the first movement toward the input workspace of the objects (ICs).
3. It uses a special pattern-recognition function developed to recognize the number of IC's and their locations (positions and orientations).
4. The data gathered in item 3 is sent to the EV80C196KA (slave processor) to create consecutive "pick and place" movements using a PD Control Algorithm and a Cubic Polynomial Trajectory Generation Algorithm until all the ICs have been collected.

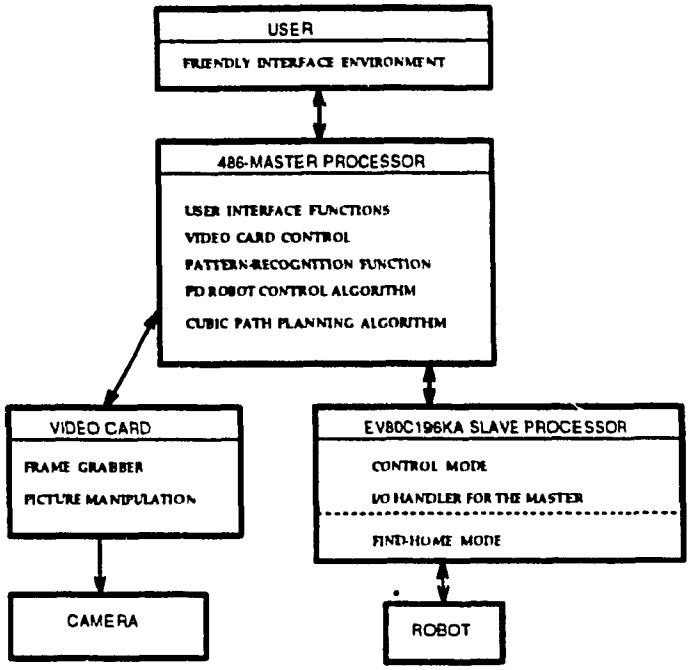

Figure 4.1: Block diagram of the Vision-Based Robot Control software.

5. At the end, the master sends the slave the last instruction to return to the initial point.

The software was developed to recognize approximately flat and symmetric (circular, rectangular, square, triangular) objects in a multi-object environment in a matter of seconds. The software that was developed to recognize different IC sizes (rectangles) can easily be applied to any flat and symmetric object. Since, in the electronics industry, most of the parts are flat and symmetric, the presented approach can be easily applied in 2D. Figure 4.1 shows a block diagram of the system and the role of each part in the overall design. The following section defines and describes some variables involved in image analysis which are part of our pattern-recognition software.

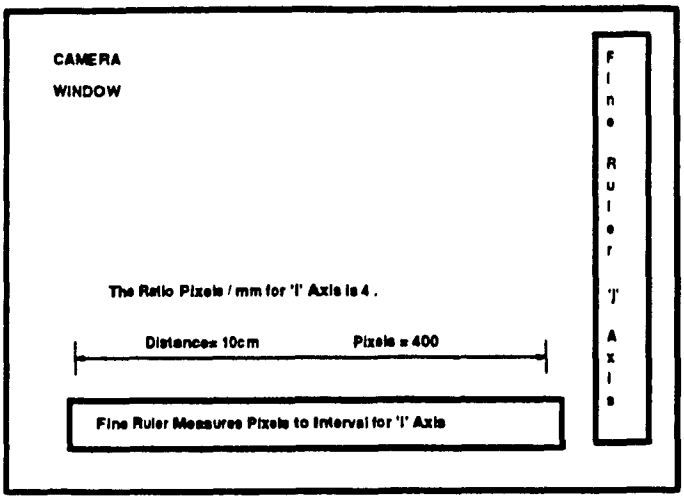

Figure 4.2: Description of pixels/interval measurement.

4.2 Procedure to Obtain Parameters

In order to define the location of a flat object in a workspace, the following evaluations need to be carried out:

1. Finding the Pixels/Interval ratio; after the camera has been installed on the robot arm and the window size is chosen, measurements are done to determine the ratio of pixels/interval(mm). The measurements are taken along the *i-axis* and *j-axis* of the camera window, since sometimes a deviation in the Image Sensor or the camera lens is created during the manufacturing process, or by the fact that the camera is not installed vertically with respect to the robot workspace. The measurement can be done with a precise ruler. More details are given in Figure 4.2.

   The variables in our program that represent pixels/interval(mm), are c_x (i-axis) and c_y (j-axis) and the values found are 3.56 pixels/1mm and 3.48 pixels/1mm respectively.

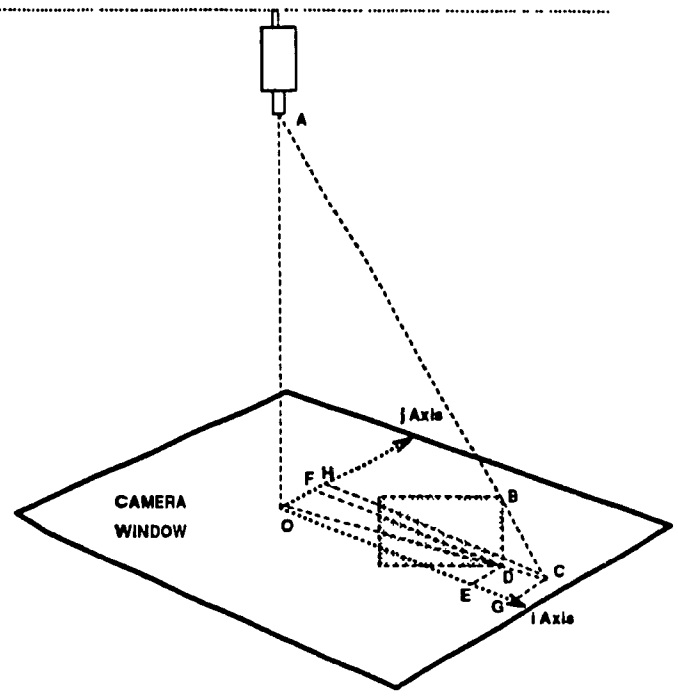

Figure 4.3: Description of the parallax effect and the solution.

2. Description of the parallax effect parameters:

In Figure 4.3, we see a black object in the Camera Window. The object (box) is placed such that its sideline DB is vertical to the camera surface and one of its upper corners is at point B. The Parallax effect occurs when the 2D image shows that the corner of the object is located at point C, when actually the corner is located at point D and the size of the error is DC. Thus we have to create a mechanism that will eliminate this error once point C is found.

Let us define a point O at the center of the camera window. The distance from the camera to the table is OA. The distance from the surface of the object (box) to the table is BD, and the distance from the center of the camera window to the reflection of the corner B in the camera window is OC. If these three lines are given, we can find the error distance DC since, the triangles $\overline{AOC} \sim \overline{BDC}$

Figure 4.3 Lines	*Program Variables*	*Distance, Indices*	*Function*
AO	d_{ct}	420mm	Distance from camera to table
OC	d_{ft}	8mm	Distance from IC face to table
O	I_{cp}, J_{cp}	255,241	Pixel indices at the center
OG	i_{c_r}	—	i index at the false corner value
OH	j_{c_r}	—	j index at the false corner value
OE	$i_{c_{r_N}}$	—	i index at the true corner value
OF	$j_{c_{r_N}}$	—	j index at the true corner value

Table 4.1: The parallax variables.

thus,

$$\frac{OC * BD}{AO} = CD \tag{4.1}$$

and then $OD = OC - CD$. The true distance of the corner from the camera window center is OD. The various distances shown in Figure 4.3 and their program variables are presented in Table 4.1. It should be noted that the parallax effect is zero at the center of the camera window, and is maximum at the corners of the camera window. From the above correlation we find the true object corner locations $i_{c_{r_N}}, j_{c_{r_N}}$ using equations 2.21 and 2.22 in Section 2.2.3.

3. Since the camera is installed parallel to the second link of the *IBM 7545*, and is vertical to the robot workspace, it is assumed that the i-axis of the camera window is parallel to the second link (Figure 4.4); but, this may not exactly be the case in practice, since the camera may not always be mounted precisely parallel to the second link. That is, the camera window may sometimes be at

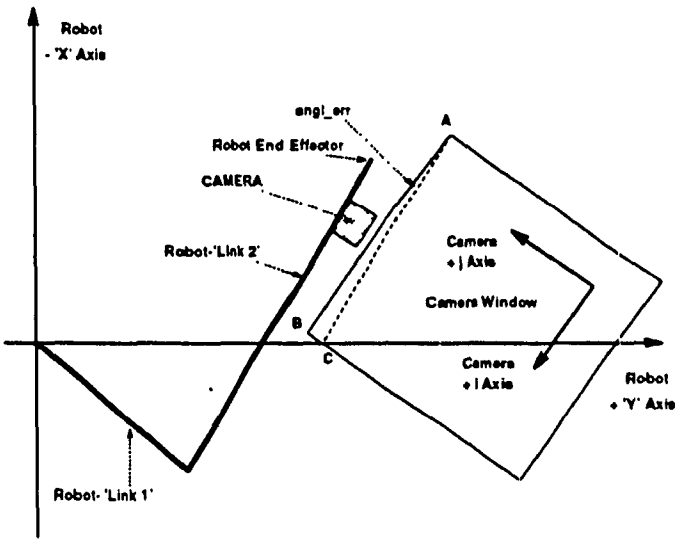

Figure 4.4: General system coordinates and location of *angl_err*.

an unknown angle $\angle\overline{BAC}$ to the second link. This error should be taken into account in computations performed. At this point we can start translating the position and orientation of the IC from camera coordinates to robot coordinates. The first and second joints control the XY position of the E_e, the third joint, Z, controls the height of the E_e above the table and the fourth joint, controls the orientation (roll) of the gripper. The translation of both position and orientation will be explained in the following two sections.

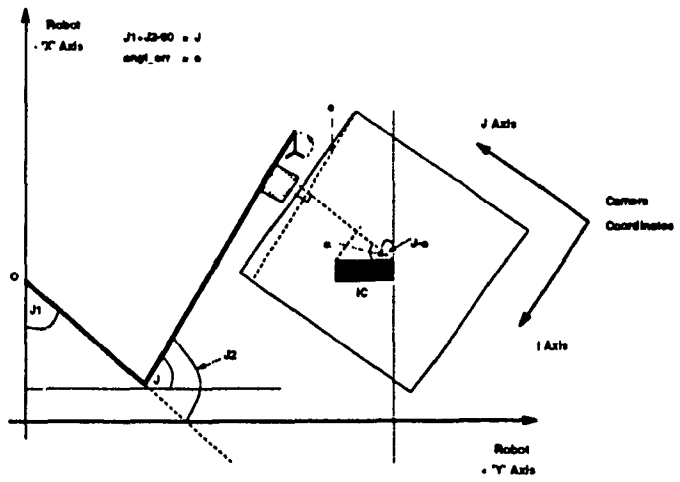

Figure 4.5: Translation of IC orientation from camera coordinates to robot coordinates.

4.3 Translating IC Orientation from Camera Coordinates to Robot Coordinates

As shown in Figure 4.5, we define $\angle J1$ for the first joint at the origin, and $\angle J2$ for the second joint. Then $\angle J$ is given by

$$J = J1 + J2 - 90^{o} \tag{4.2}$$

The $\angle \alpha$ is the translation of the orientation of the IC in the camera coordinates to the j-axis of the camera and the $\angle J\text{-}e$ is the angle that translates the j-axis of the camera coordinates to the X-axis of the robot coordinates, which is also parallel to the axis of rotation of the E_e. Therefore, the final formula is

$$O^{r} = -(\alpha + J1 + J2 - 90^{o} - angl\_err) \tag{4.3}$$

where O^{r} is the Orientation in robot coordinates.

The negative sign preceding the parentheses indicates that the object orientation is in the counter clockwise direction. Equation 4.3 is valid for the entire workspace of the robot.

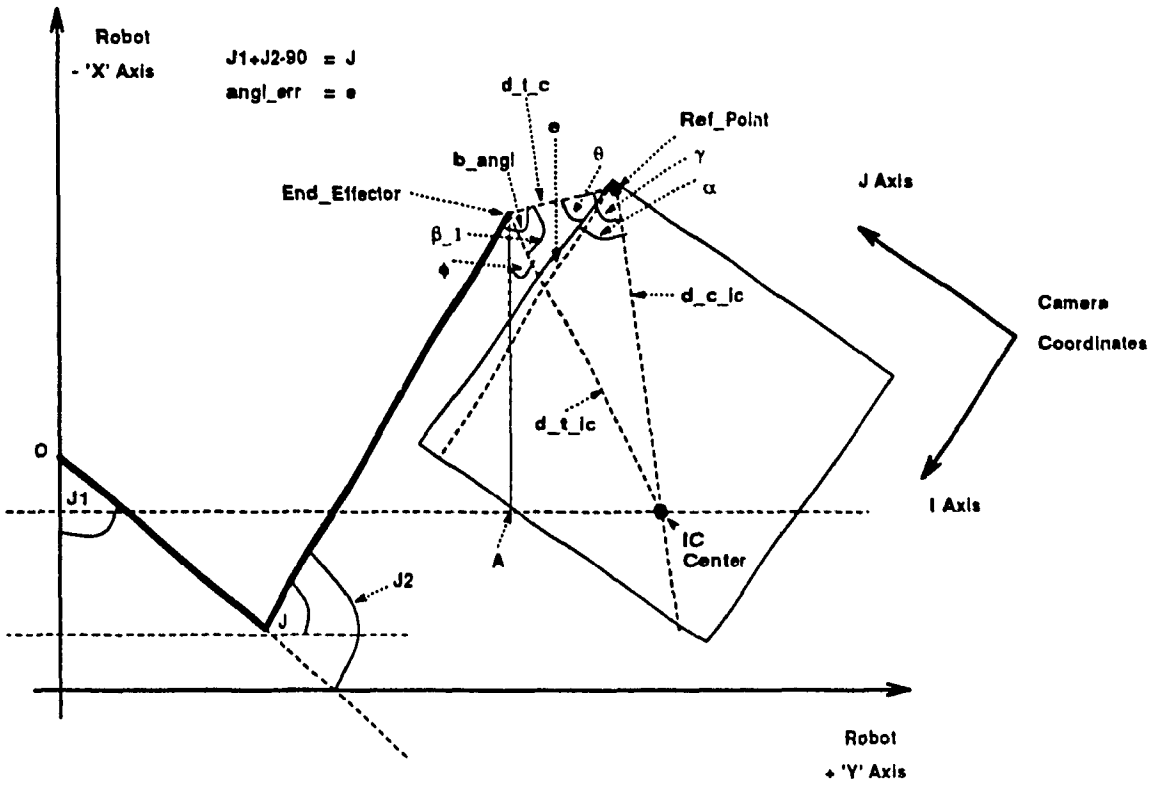

Figure 4.6: Translation of IC position from camera coordinates to robot coordinates.

For simplicity, we state that if the value of the Orientation $O^r > 90^o$, then $O^r_{new} = O^r - 180^o$, and if $O^r < -90^o$, then $O^r_{new} = O^r + 180^o$.

4.4 Translating IC Position from Camera Coordinates to Robot Coordinates

Figure 4.6 describes the approach used to translate IC positions from camera coordinates to robot coordinates. First of all, we should note the following points:

1. The actual location of the E_e can be obtained at any time.

2. If we choose a reference point (Ref_point in Figure 4.6) inside the camera window at the corner closest to the E_e, and we assume that the camera is mounted at

a fixed place near the E_e, then we can assume two additional parameters of constant data which can be measured and used when we translate IC positions from camera coordinates to robot coordinates:

(a) The distance between the E_e and the closest corner of the camera window is fixed (constant). It can be measured and is defined by d_t_c in Figure 4.6.

(b) The angle of the line joining the E_e and closest corner of the camera window to the second link is fixed (constant), and is defined by b_angl in Figure 4.6.

3. The center of the IC in camera coordinates (see the explanation in the next section), is given by pixel indices (i,j).

Having the above data, we use the following steps to find the location of the IC in robot coordinates:

1. First, we calculate θ as

$$\theta = 180^0 - b\_angl - angl\_err. \tag{4.4}$$

Again, in theory, $\theta + b\_angl$ should be equal 180°, but some uncertainty exists when the camera is mounted with the window at an unknown angle ($angl\_err$) with link_2 of the manipulator.

2. At this point, our goal is to find the angle created by the {(End_effector)-(Ref_Point)} and {(Ref_Point)-(IC_Center)} lines. This angle is composed of two angles:

(a) The first is θ which defines the angle between {(E_e)-(Ref_Point)} and {(Ref_Point)-(i-axis)} lines.

(b) The second is α which defines the angle between

{(Ref_Point)-(i-axis)} and {(Ref_Point)-(IC_Center)} lines.

If we define the constant location of the Ref_Point in camera coordinates as b_x for i coordinates and b_y for j coordinates, and the location of the center of the IC in the camera coordinates as i_C, j_C, then the value of α is,

$$\alpha = \arctan\left(\frac{\| (b\_y - j_C)c\_y \|}{\| (b\_x - i_C)c\_x \|}\right) \tag{4.5}$$

The ratio of c_y and c_x variables is to fix the deviation of the data collected from the picture to its true value.

Having found θ and α, it is easy to calculate γ,

$$\gamma = \theta + \alpha \tag{4.6}$$

3. The next step is to find the absolute distance between the *Ref_Point* to *IC Center* defined in Figure 4.6 by the variable *d_c_ic*. Using the same variables as in equation 4.5, we have

$$d\_c\_ic = \sqrt[2]{((b\_y - j_C)c\_y)^2 + ((b\_x - i_C)c\_x)^2} \tag{4.7}$$

4. Having found γ and *d_c_ic* along with the measured length d_t_c, we can use the cosine law to find the absolute distance *E_c – IC Center* in Figure 4.6 defined by *d_t_ic*,

$$d\_t\_ic = \sqrt[2]{(d\_t\_c^2 + d\_c\_ic^2 - 2 * d\_t\_c * d\_c\_ic * \cos(\gamma)} \tag{4.8}$$

5. To reach our final step we must find ϕ which gives us the angle that will translate the absolute distance *d_t_ic* into the robot *(X, Y)* coordinates. Using the three lines *d_t_c*, *d_c_ic*, *d_t_ic*, we use the cosine law again to find $\angle\beta\_1$, $0^o \leq \beta\_1 \leq 180^o$:

$$\beta\_1 = \arccos\left(\frac{(d\_t\_c^2 + d\_c\_ic^2 - d\_c\_ic^2)}{(2d\_t\_c * d\_c\_ic)}\right) \tag{4.9}$$

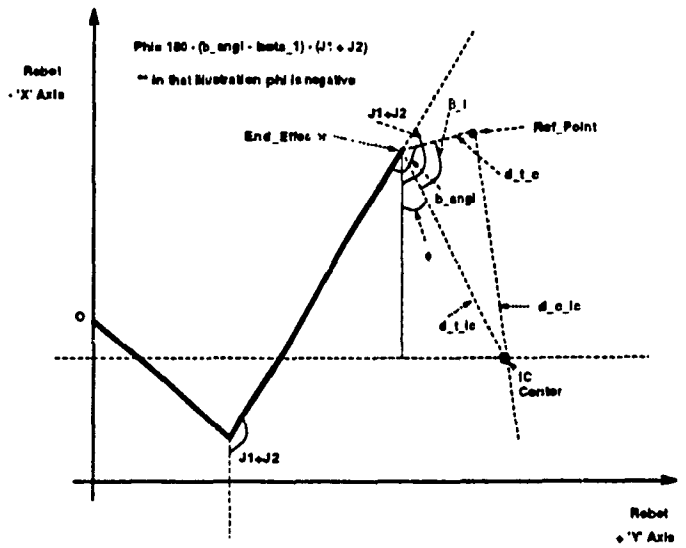

Figure 4.7: Illustration for equation 4.10.

6. Then,

$$\phi = 180^{\circ} - (b\_angl - \beta\_1) - (J1 + J2) \tag{4.10}$$

Note that ϕ may have positive or negative values.

We can find the location of the E_e in (X, Y) robot coordinates by using the data supplied by the robot manufacturer: $link\_1 \approx 400mm$ and $link\_2 \approx 250mm$,

$$X_{E\_e} = 250 * \cos(J1 + J2) + 400 * \cos(J1) \tag{4.11}$$

and,

$$Y_{E\_e} = 250 * \sin(J1 + J2) + 400 * \sin(J1) \tag{4.12}$$

Now, using the values of ϕ and d_t_ic found earlier, we can obtain an accurate value of the location of the IC center using the following equations,

$$X_{IC} = X_{E\_e} + d\_t\_ic * \cos(\phi) \tag{4.13}$$

and,

$$Y_{IC} = Y_{E\_e} - d\_t\_ic * \sin(\phi) \tag{4.14}$$

7. It should be understood that even though the data collected by measurements of *d_t_c* and *b_angl* is accurate, the final answer in equations 4.13, 4.14 contains the distance from the *origin* ('O' in Figure 4.7) to X_{E_c}, Y_{E_c}. This distance is based on the data supplied by the manufacturer, and includes an error of up to $\pm$ 1mm for each link. This means that *link_1* and *link_2* are approximate values, and once the measurements for *d_t_c* and *b_angl* are done, in order to complete the calibration of the robot-vision system, the operator should perform on-line calibration using the five variables: *d_t_c*, *b_angl*, *angl_err*, *link_1* and *link_2*.

It should be noted that the parameters *link_1* and *link_2*, are independent of the *visual parameters*. This means that any general calibration of *link_1* and *link_2* parameters may take place before proceeding to the calibration of the *d_t_c*, *b_angl*, and *angl_err* parameters.

4.5 Computing Orientation and Center Position of IC in Camera Coordinates

Figures 4.8 - 4.11 illustrate the procedure to find the center position and orientation of an IC.

From Figures 4.8 and 4.9, we see that two different cases may result when grabbing a picture for a pattern-recognition application (we use the theory explained in section 2.1.2).

In Figure 4.8, we see the case where the object (IC) is parallel to the camera window. Here the IC pins have a substantial effect on the width of the external rectangle. If the unwanted effect of the IC pins is on the left side of the IC then, when using equation 2.8, the center of the external rectangle will move to the left

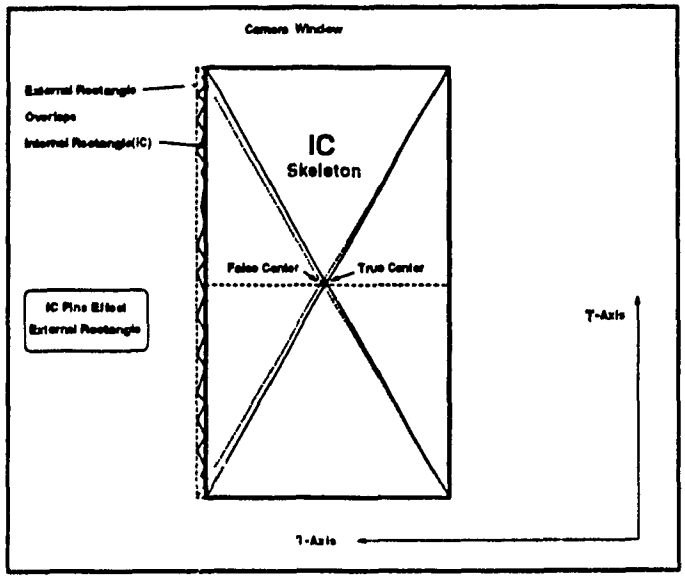

Figure 4.8: IC Pins affect the external rectangle.

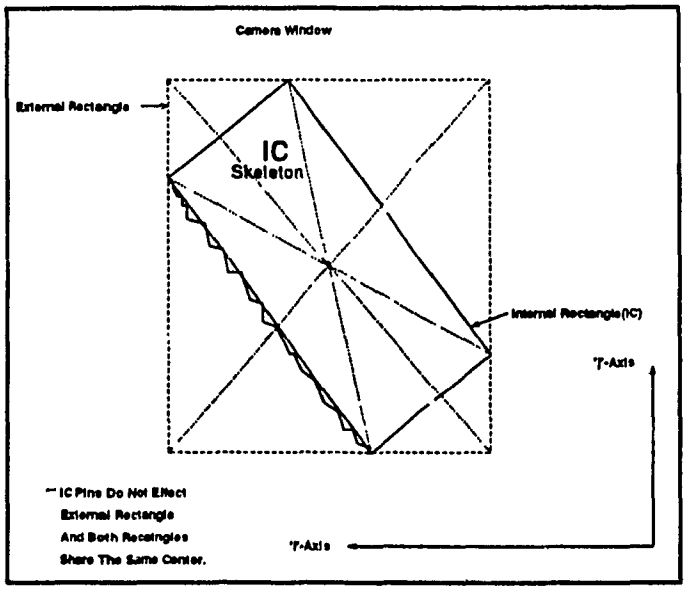

Figure 4.9: IC Pins do not affect the external rectangle.

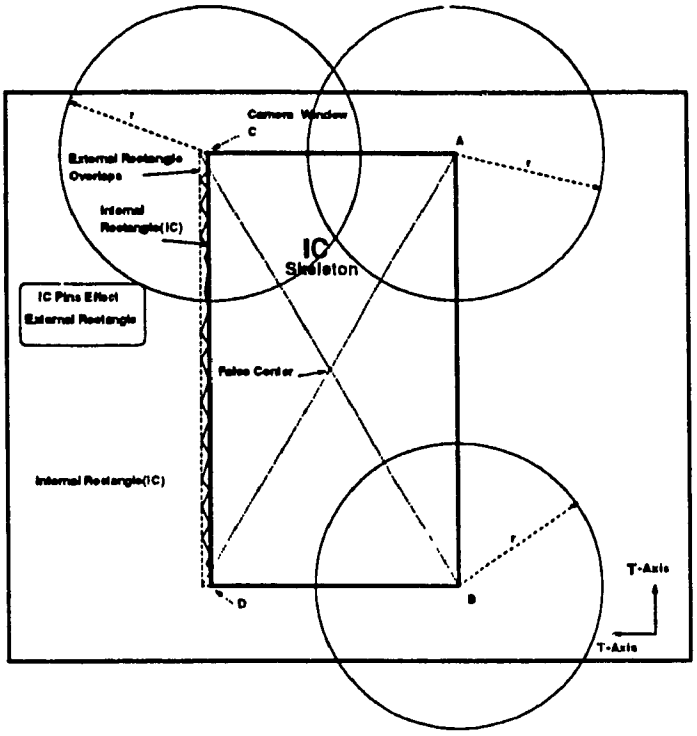

Figure 4.10: Locating the four IC' corners.

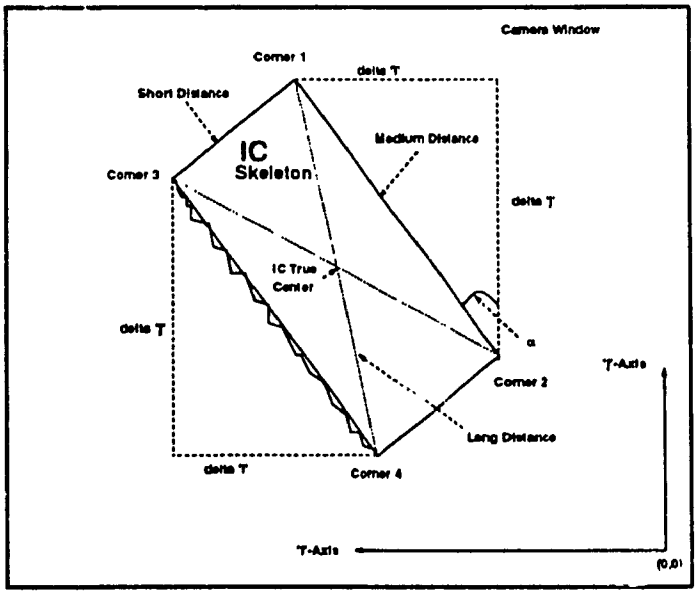

Figure 4.11: Locating the true center and angle α.

of the rectangle (IC) along the broken line. In dual-in-line ICs, the upper and lower sides of the ICs are pin free and depend on the IC package only. In Figure 4.9, the IC package is not parallel to the camera window axes and, as can be seen, the IC pins do not affect the external rectangle. Using equation 2.8, we expect to compute the true center of the external and the internal rectangles. To conclude this discussion the following points should be noted:

1. The unwanted effect of the IC pins may occur when the IC is positioned at angles in the range (0°,20°) to one of the camera window axes; 0° is the worst case.

2. The position of the IC in the camera window is important, since the effect of the IC pins is prominent when positioned in parallel near the boundaries of the window. Therefore, only one side of the IC pins may be seen by the camera and the error from the true center of the IC can reach up to 0.5 mm. On the other hand, if the IC is positioned near the center of the camera in parallel with the camera window axis, both sides of the IC pins can be seen by the camera so, the error tends to cancel out.

In order to find the true corners of the IC package, and to overcome the problem mentioned in the second item above, we use the algorithm described in section 2.1.2 and illustrated Figure 4.10. Here, the IC pins that the camera sees are located on the left side of the IC. Therefore the center of the external rectangle does not overlap the (true) center of the internal rectangle (IC). Assuming that the false center (f_c), is located to the left of the true center, the farthest point of f_c is either A or B, since the distance relations must satisfy

$$\| f_c - A \| \approx \| f_c - B \| > \| f_c - C \| \approx \| f_c - D \| \qquad (1.15)$$

Let us assume that the farthest point found is a point at corner A. To be sure that no other point will be chosen near that corner we impose a restriction that the next farthest point cannot be chosen if the distance from A to the second farthest point is less than r. In this way, we find the true corners B, C and D. Figure 4.11 describes the algorithm to find the IC's true center and the angle α. To locate the IC's true center, we use equation 2.11, and to find α we do the following:

1. Prepare an array which describes the location (i,j) of the four corner points.

2. Choose the first corner point which satisfies,

$$j_1 = max(j_{c_r}) \quad c_r = 1, \ldots, 4 \tag{4.16}$$

3. From the first corner point, we measure the absolute distance to all the other corners. By doing this, we can estimate the location of each corner of the IC. The longest distance always belongs to the *diagonal* between corners 1 and 4, the medium distance always belongs to the longest side connecting corners 1 to 2 of the IC package, and the shortest distance always belongs to the shortest side of the IC package connecting corners 1 to 3.

4. We can now find the orientation of the IC package in the camera window coordinates (α) in two ways. By calculating the slope between corners (1 and 2) and the j-axis, or the slope between corners (3 and 4) and the j-axis (equation 2.12). To reduce the error we average both slopes for the final IC orientation (α).

4.6 How Partial Groups of Objects are Created

Figure 4.12 illustrates how partial groups that describe an IC are created. As illustrated, the direction of scanning starts from a pixel at location $(0,0)$ and proceeds

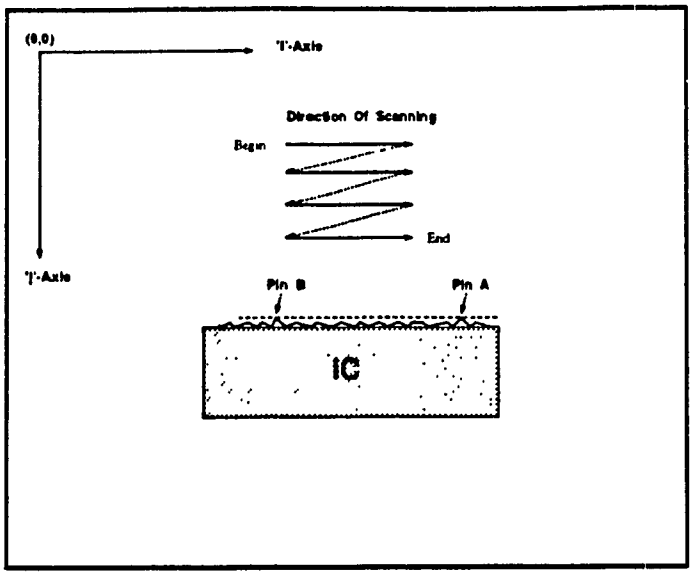

Figure 4.12: Explanation for the formation of partial groups in an IC.

to the right until the end of the line. When the end of line is reached, a new line of position $0,j$ starts. In some special cases where the i-axis of the camera coordinates is parallel to the longest side of the IC, it may happen that some parts of the pins protrude a little more than others. In Figure 4.12 we can see that *pin A* and *pin B* protrude more than others in the group. When the computer scans the picture from left to right, it passes through *pin B*, which is the first pixel belonging to the new IC, and opens a new partial group for additional pixels joining that same partial group (IC'). The computer continues scanning to the right and then finds a second part of the same IC *pin A* represented by a pixel where the distance between Pin A and Pin B is more than ϵ in camera coordinates (please see page 13). In this case the computer again opens a new partial group for the same IC, and when it finishes scanning the whole picture, it will put both partial groups into one group as explained on page 14.

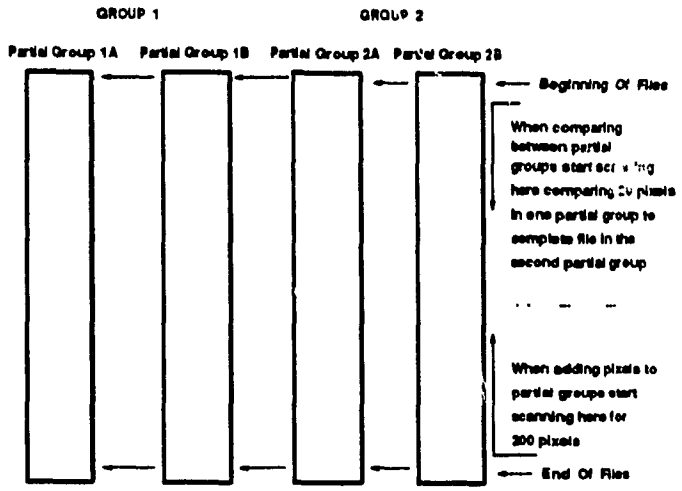

Figure 4.13: Illustration of how to reduce computational cost.

4.7 Reducing Computational Cost

In cases where the camera window has many IC's, say about 15, then joining pixels into partial groups and merging between partial groups increases the computation time drastically. It was found that in the worst case, when no special effort was made to reduce the amount of computation, locating 15 ICs in the camera window took up to 30 seconds. To reduce this time, two important points should be noted (please see Figure 4.13):

1. When trying to add the current pixel to partial groups, the current pixel should be compared first to the last pixel in the partial group and then to continue toward the beginning of the file of the partial group. If the longest IC is 30 mm, then the worst case number of pixels covering the line is 30mm x 3.5pixels/mm= 105 pixels. To be on the safe side, we compare the current pixel with the last 200 pixels for each partial group (please see page 13).

2. When trying to combine skeleton partial groups into a skeleton group (IC), we compare the whole of one skeleton partial group (pixels) to only the first 20

Number Of *ICs*	Time Required (seconds)
1	0.2
3	0.6
5	1
8	2
11	3.5
15	8

Table 4.2: Computation cost of the pattern recognition function.

pixels from the beginning of each skeleton partial group since, skeleton partial groups tend to match at the beginning of the file.

We performed some measurements to evaluate the computational cost of the pattern-recognition function. The results are given in Table 4.2.

4.8 Treating Undefined Objects

The software assumes that all the objects presented to the camera window are well defined. Sometimes, visible black points in the robot workspace may be presented to the camera. In that case the software will discard that data. The explanation is that the size of data created by the black points is very limited as compared to an IC. If a piece of data does not satisfy equation 2.13 which defines a rectangle, then the software will discard the data.

4.9 On-Line Robot-Vision Calibration

To avoid recompilation during robot-vision calibration (see Appendix A), the operator can change the key variable values on-line. The key variables are:

1. b_angl.
2. d_t_c.
3. angl_err.
4. link_1.
5. link_2.
6. Robot Error_Gains:
   (a) kp_1, kp_2, kp_Z, kp_R.
   (b) kv_1, kv_2, kv_Z, kv_R.

Further information concerning the first three variables is given in Section 4.4. To conclude this chapter, we give a flowchart of the pattern-recognition function in Figures 4.14- 4.18.

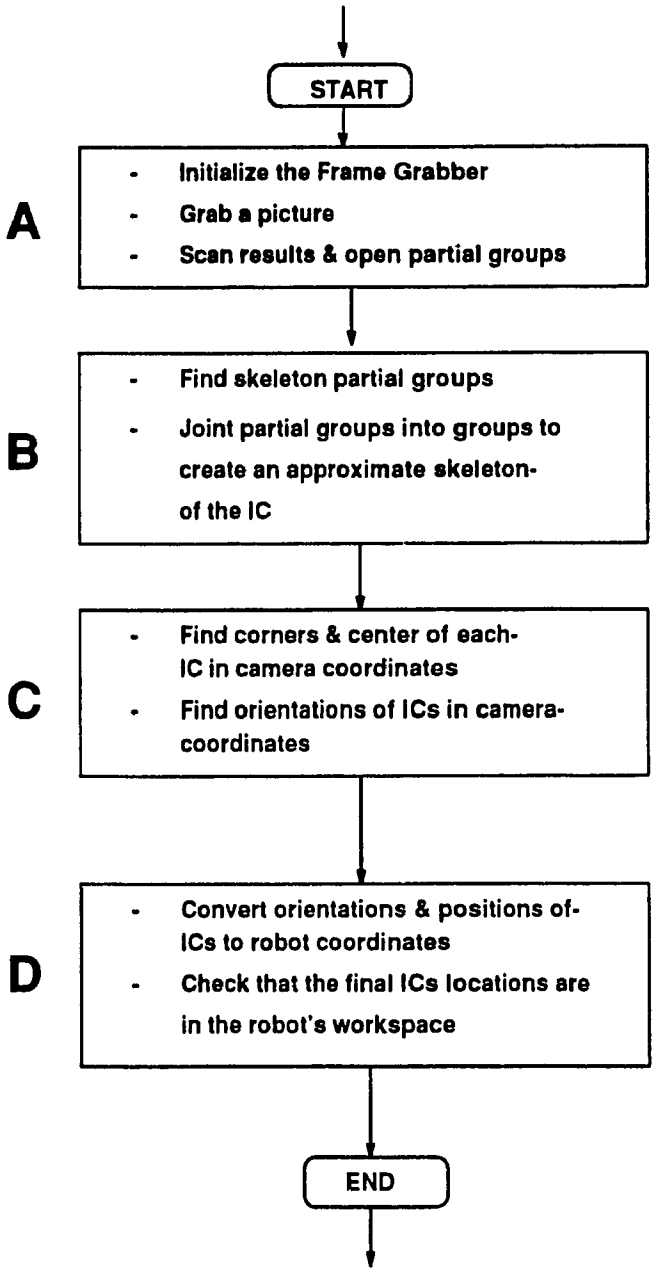

Figure 4.14: Main block diagram of the pattern-recognition function.

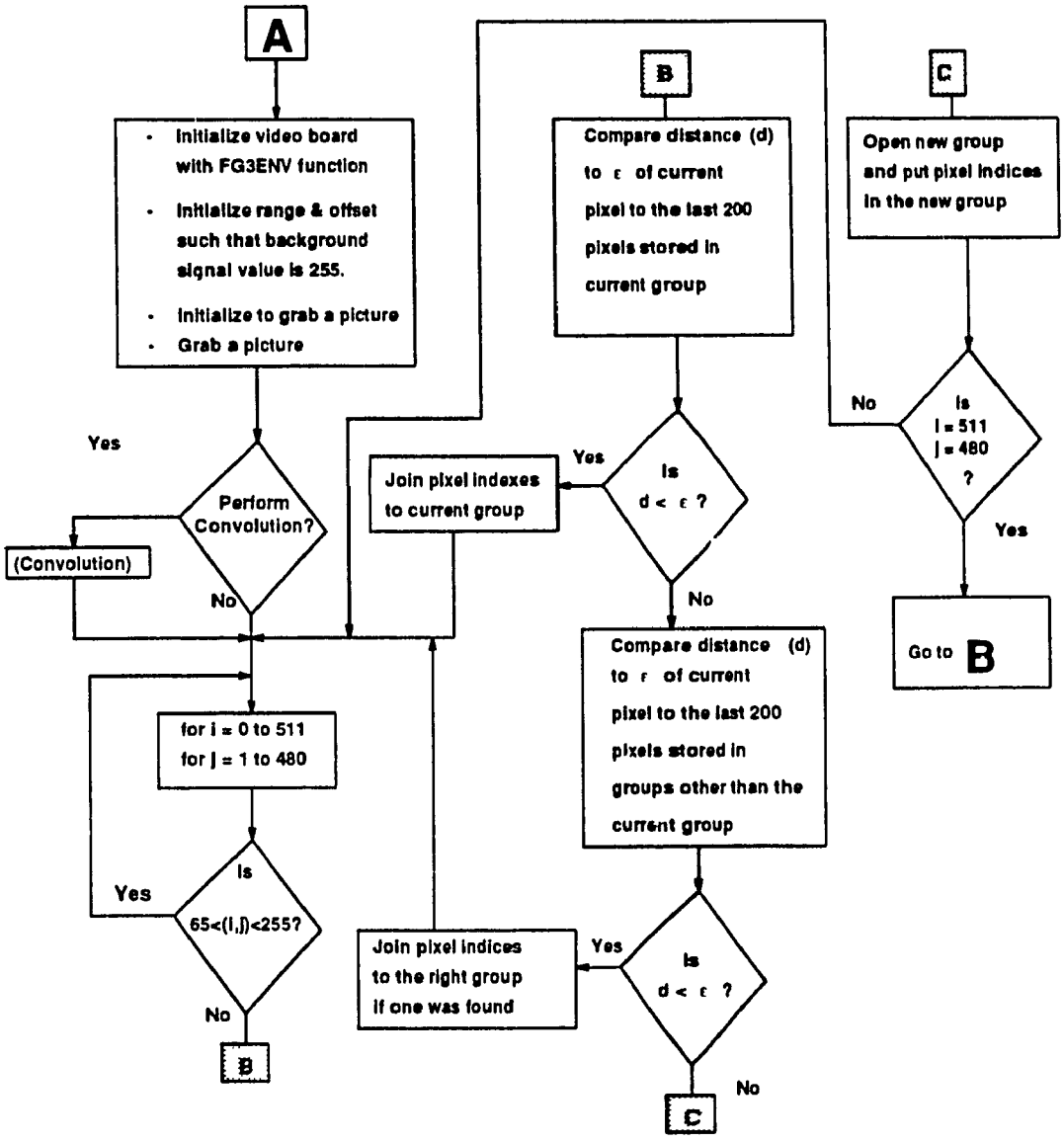

Figure 4.15: This is Section A of the Main Block Diagram.

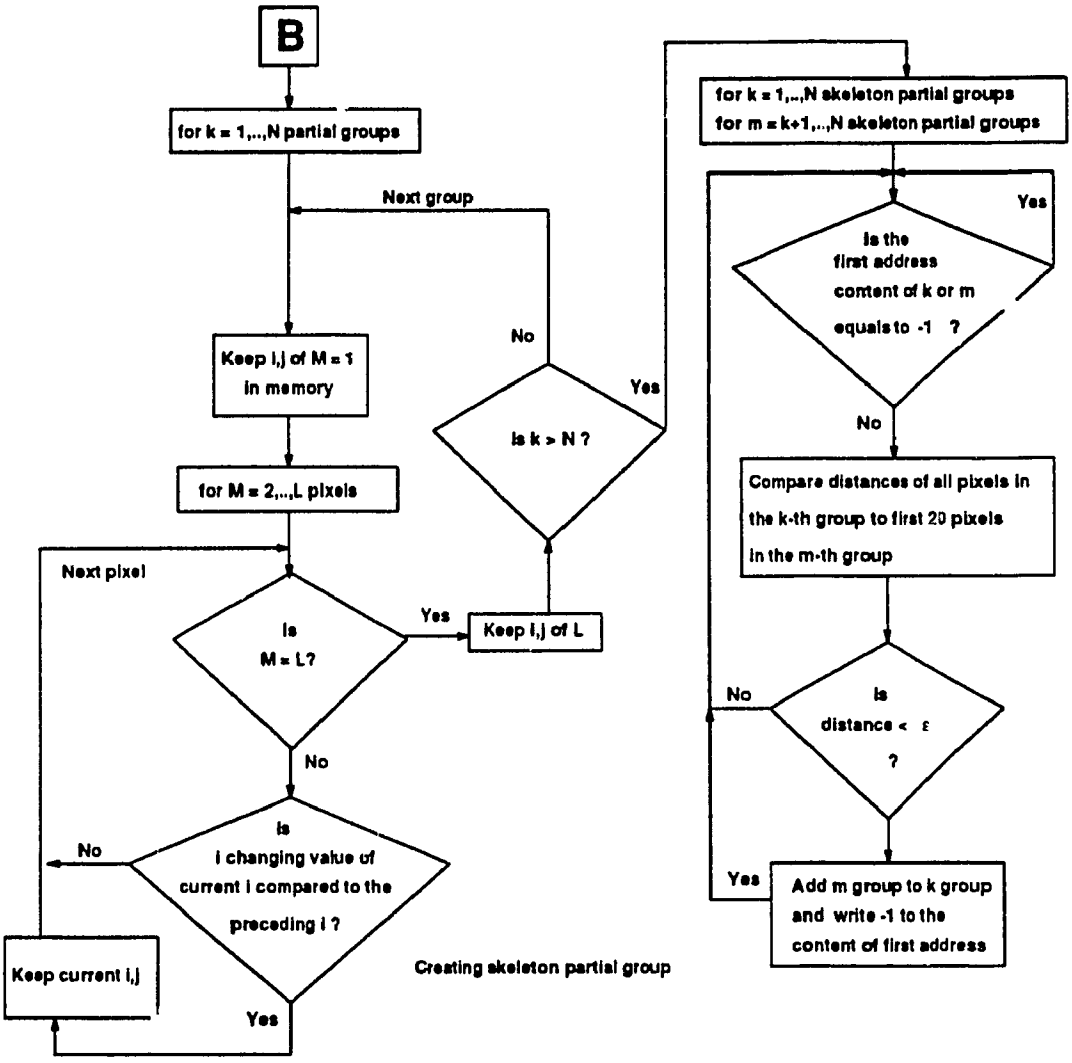

Figure 4.16: Section B of the main block diagram.

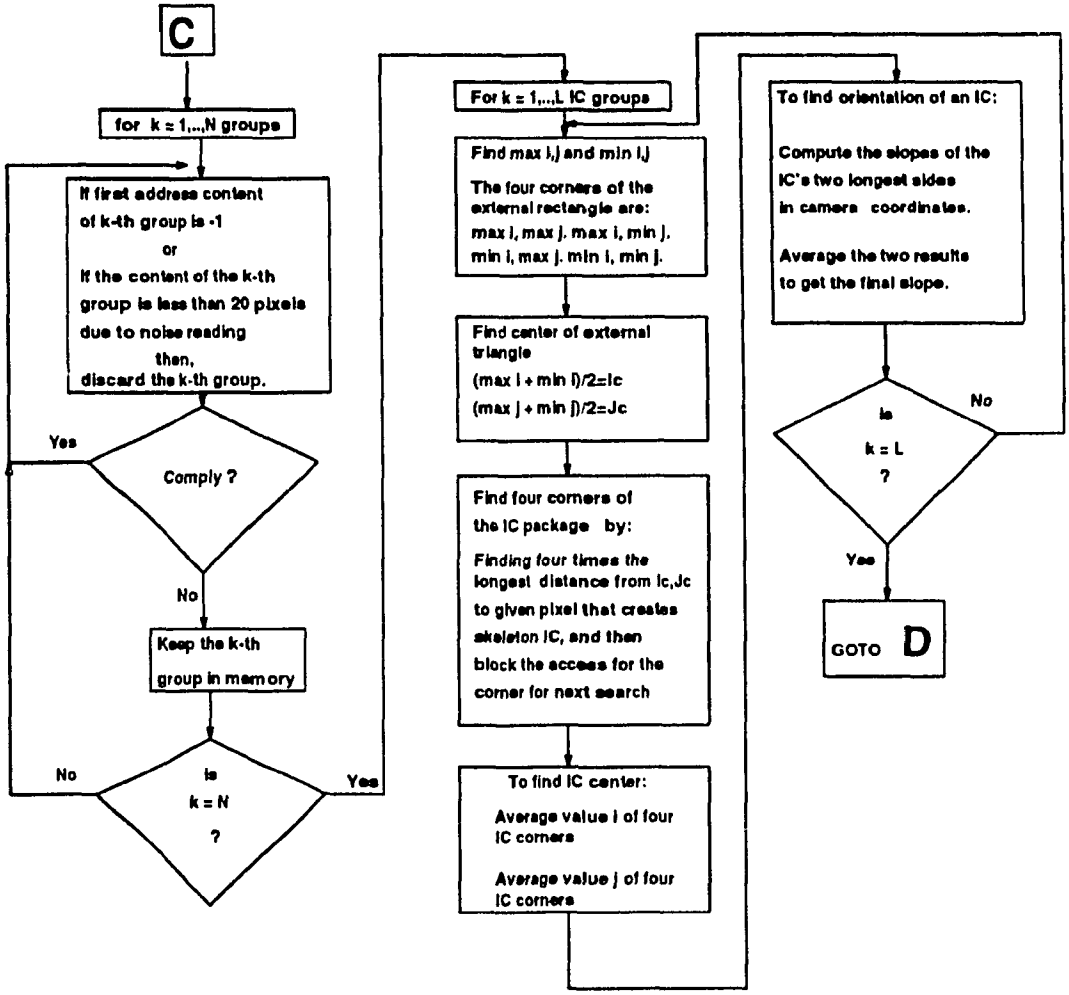

Figure 4.17: Section C of the main block diagram.

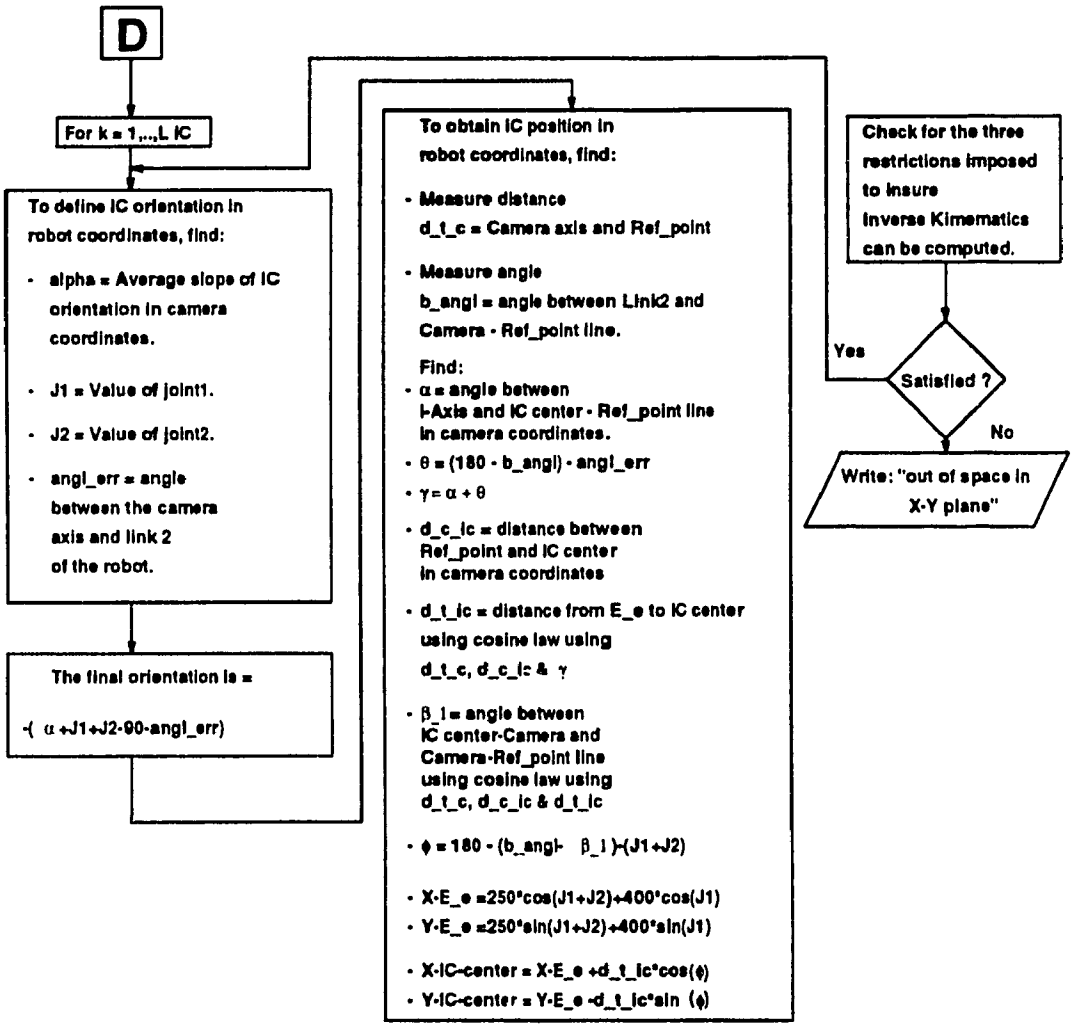

Figure 4.18: Section D of the main block diagram.

Chapter 5

ROBOT SOFTWARE DESCRIPTION

5.1 Introduction

This chapter gives a brief description of the software that controls the IBM7545 robot. Since much of this part was covered in [1], we mainly concentrate on the changes that have been made to the original software in order to adapt it to our needs.

The program uses a PD control algorithm which provides error-driven, independent joint control whereby each joint is controlled separately by a simple position-velocity servo-loop with predefined constant gains. The joint torque at a sampling instant N is given by

$$\tau(N) = k_p e_p(N) + k_v e_v(N) \tag{5.1}$$

where

$$e_p(N) = \textit{desired joint position}(N) - \textit{actual joint position}(N) \tag{5.2}$$

and

$$e_v(N) = \frac{e_p(N) - e_p(N-1)}{T_s} \tag{5.3}$$

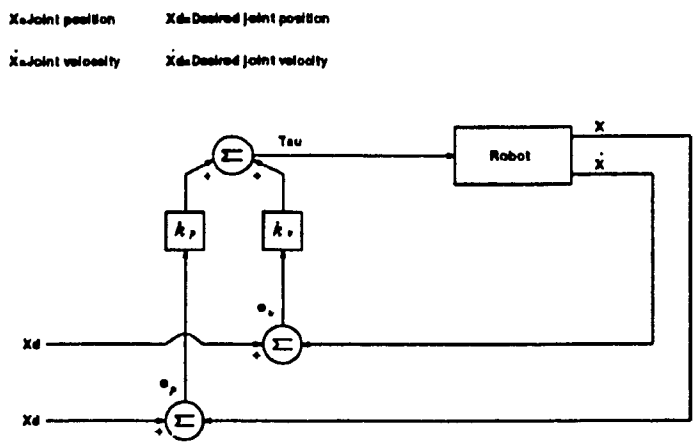

Figure 5.1: Block diagram of the trajectory-following controller.

are the position and velocity tracking errors, $k_p > 0$ and $k_v > 0$ are the position and velocity feedback error gains, and T_s denotes the sampling period. The PD controller was implemented in our application because

1. It requires very little computation and can therefore be implemented at high servo rates.

2. The mass of the payload (i.e. the IC's) is very small, and does not affect the robot dynamics, so adaptive or robust control is not needed.

Figure 5.1 shows a block diagram of our trajectory following controller. If there is no initial error, the robot will follow the desired trajectory almost exactly. If there is an initial error, it will decay rapidly because of the gains k_p and k_v chosen in equation 5.1.

The path planning algorithm is called by the control program and computes the entire path off-line prior to the execution of the move. The algorithm is based on a *cubic spline generator.* The algorithm for this path generator enables the specification of via point locations and also provides automatic selection of joint velocities at the selected via points. The velocities at the via points are computed according to the following scheme. First, the slopes (average joint velocities) between adjacent via

points are calculated. The velocity at a via point is then set to zero if the two slopes on either side are of opposite sign. If the slopes are of the same sign, the velocity is computed as the average of the two slopes. A full description of the *cubic spline generator* algorithm is given in [10].

5.2 Dual-Port Ram Utilization

The Dual-Port Ram (DPR) is a shared memory device and is used for passing parameters and variables between the Master and the Slave processors. Each parameter and variable is assigned a specific location (register) in the DPR. Since the new Master processor a *486* based *PC* replaced the old version, a *PS2/50*, some addresses have changed. Also a new variable, *gripper*, was added. The address and name of each register in the DPR is listed in Table 5.1.

5.2.1 The Gripper

The gripper function was added to the system in order to control the closing and opening of the gripper (at the E_e). As shown in Table 5.1, the Master controls the Gripper through address C802AH (Gripper Register). The function is defined in Table 5.2. The Slave sits in a continuous loop checking the *Command Register* and the *Gripper Register*. When one bit of the E_e function is set, the Slave performs the corresponding task and then clears the Gripper Register.

ADDRESS		SIZE	NAME	USER ACCESS	
Master	Slave			Master	Slave
C8000H	E000H	word	*sample_period*	w	r
C8002H	E002H	word	*timing_A*	r/w	r/w
C8004H	E004H	word	*timing_B*	r/w	r/w
C8006H	E006H	word	*command*	r/w	r/w
C8008H	E008H	word	*error*	r/w	r/w
C8010H	E010H	double word	*actual_position_joint_1*	r	w
C8014H	E014H	double word	*actual_position_joint_2*	r	w
C8018H	E018H	double word	*actual_position_joint_Z*	r	w
C801CH	E01CH	double word	*actual_position_joint_R*	r	w
C8020H	E020H	word	*torque_joint_1*	w	r
C8022H	E022H	word	*torque_joint_2*	w	r
C8024H	E024H	word	*torque_joint_Z*	w	r
C8026H	E026H	word	*torque_joint_R*	w	r
C802AH	E02AH	word	*gripper*	w	r/w

Table 5.1: Dual-port RAM register reference table.

Bit Number Set	Indication
0	Open Robot Gripper
7	Close Robot Gripper

Table 5.2: The gripper register function.

5.3 Robot Motion

The robot motion is separated into three stages:

1. The initial movement and the movement of the robot in its workspace grabbing one or more images.

2. The "pick and place" movements.

3. The last move toward its home position.

Figure 5.2 illustrates the height of the E_e above the table as a function of time when it advances toward a section of the table and discovers two IC's. There are three levels at which the E_e of the robot operates:

1. Height *0mm* (according to its home position) - This is when the gripper is moved from the home position to a point in order to grab a picture (points 0-1), and at the end of the move when it is on its way back to the home position after completing its task.

2. Height -200mm - This is when the gripper is carrying an IC or on its way to pick another IC.

3. Height -227.7mm - This is when the gripper picks or places an IC.

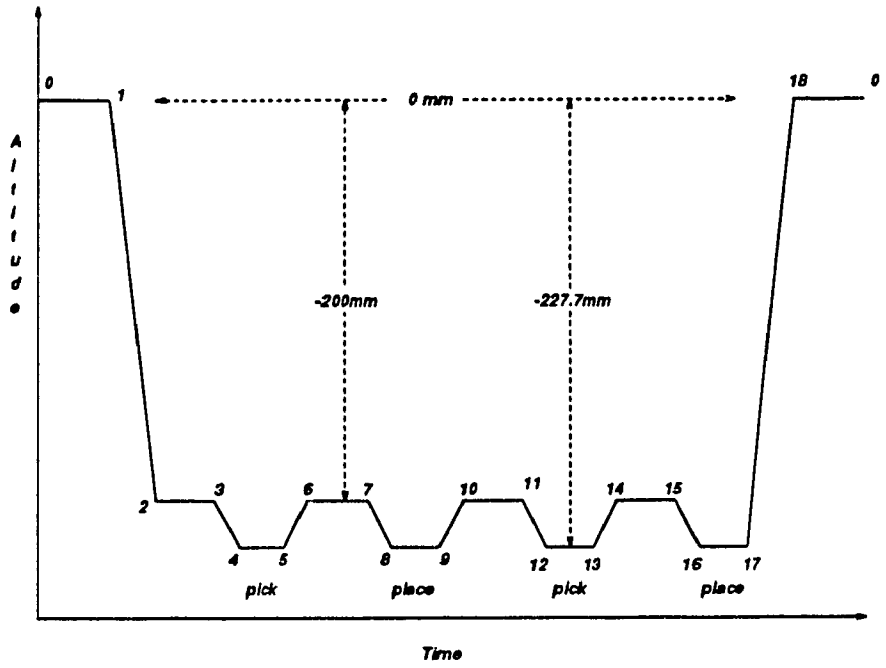

Figure 5.2: Motion and height of the end-effector above the table as a function of time.

The robot performs the task described in Figure 5.2 in the following stages:

1. Segment $0 \rightarrow 1$: The robot approaches its input window (1) in order to let the camera grab a picture and then to execute the *Pattern-Recognition* function.

2. Segment $1 \rightarrow 2$: The robot uses its joint-Z in order to reach a height of -200mm from the height of the home position (0mm).

3. Segment $2 \rightarrow 3$: The gripper is moved toward the location of the first IC using joints-1,2,R.

4. Segment $3 \rightarrow 4$: When the gripper is located above the center of the IC, the robot uses its joint-Z to position the gripper at the level of the IC to perform a pick operation. This is done by moving an additional -27.7mm in the Z direction. This places the gripper at the desired altitude of -227.7mm.

5. Segment 4 → 5 : The gripper picks up the IC. A delay of *0.5 sec.* occurs to give the mechanical part of the gripper (*controlled by an Air-Valve*) enough time to perform the closing operation.

6. Segment 5 → 6 : The robot starts its movement toward the "place" location. In order not to touch or move other IC's in the workspace, the gripper is moved back to -200mm using joint-Z.

7. Segment 6 → 7 : The gripper is moved toward the "place" location using joints-1,2,R.

8. Segment 7 → 8 : The robot uses its joint-Z to approach the -227.7mm altitude mark before placing the IC.

9. Segment 8 → 9 : The robot places the IC using the same delay as in item 5 by opening the gripper.

10. Segment 9 → 10 : The gripper is moved toward the -200mm height using joint-Z and is moved toward the second IC, and the above steps are repeated.

Segments 17 → 0 : After placing the last IC, the robot starts its return to the home position. The gripper is moved from -227.7mm to 0mm using joint-Z, and the "home" position is then reached using joints-1,2,R.

Remarks:

1. The robot may move into as many input windows as needed. At the end of each move, the system calls the *Pattern-Recognition* function. This results in finding IC locations, if any.

2. The robot movements are composed of three different motions,

(a) The first motion is from the home position to the input window and is composed of one move only (no via points).

(b) The second motion (if there are any ICs) is composed of two motions each of two via points. The first motion performs a "pick up" action for the IC and the second a "place" action.

(c) The last motion is composed of two movements (with one via point), and is used to instruct the robot to move toward its home position after all "pick" and "place" tasks have been performed.

3. In the program, there are two arrays, *In_pos* {15}{3} and *Out_pos* {15}{3}. *In_pos* {15}{3} contains the data of the present locations of the ICs and *Out_pos* {15}{3} contains the locations to where the ICs are to be moved to. The program fills the arrays from 0 down to 14. and *In/Out_pos* {*xx*}{0} contains the value of the angle converted to a Hexadecimal number for Joint_1, *In/Out_pos* {*xx*}{1} for Joint_2 and so on.

4. It is often useful to create robot motions with fixed joint velocities. The velocities of the joints are specified by an array *v*{4} as follows,

   (a) Joint_1 = 45^0 per sec.

   (b) Joint_2 = 50^0 per sec.

   (c) Joint_Z = $100mm$ per sec.

   (d) Joint_R = 70^0 per sec.

   Before the E_e of the robot moves to a new location, the distance that each joint will move is calculated. Then the time each joint needs for the move is computed. In the next stage, the chosen global time for the move is the longest time among the four joints. A minimum of 0.2 sec. global time is used for small

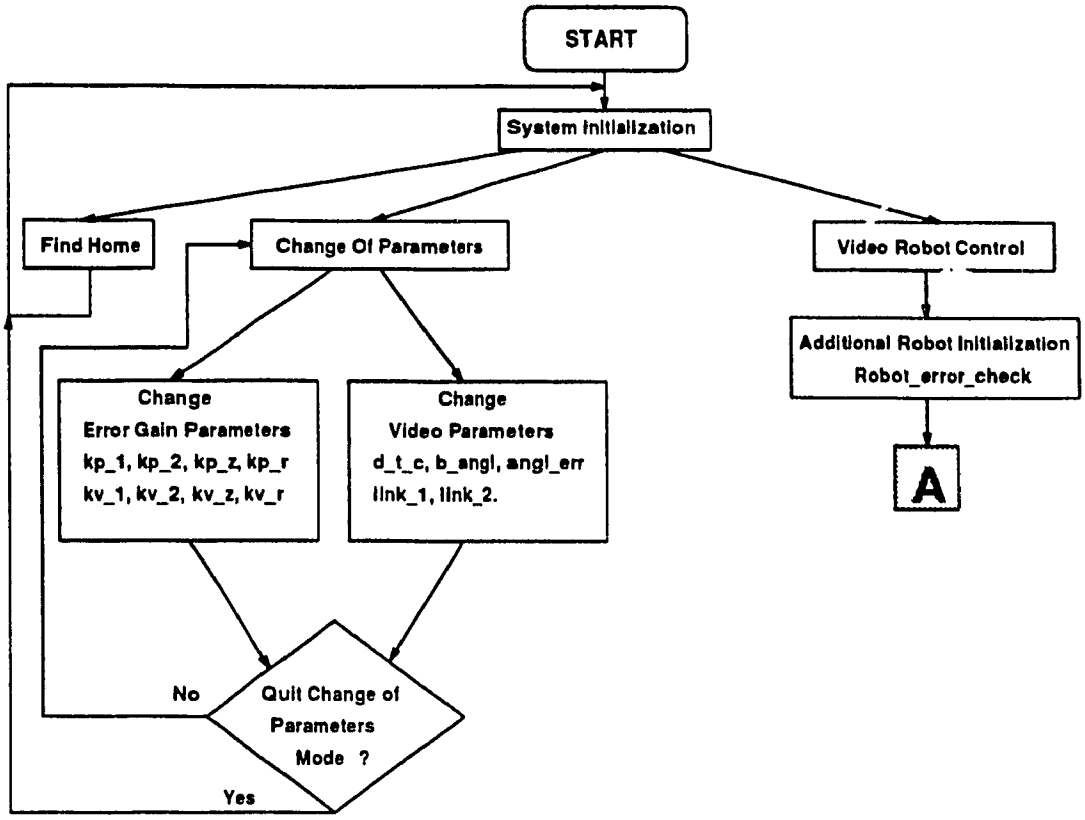

Figure 5.3: General flowchart of the robot modes (first part).

motions to avoid high torques at the beginning of a move. High torques may not be achievable in a short time and thus may cause tracking errors.

Flowcharts of the robot modes of operation are shown in Figure 5.3- 5.4.

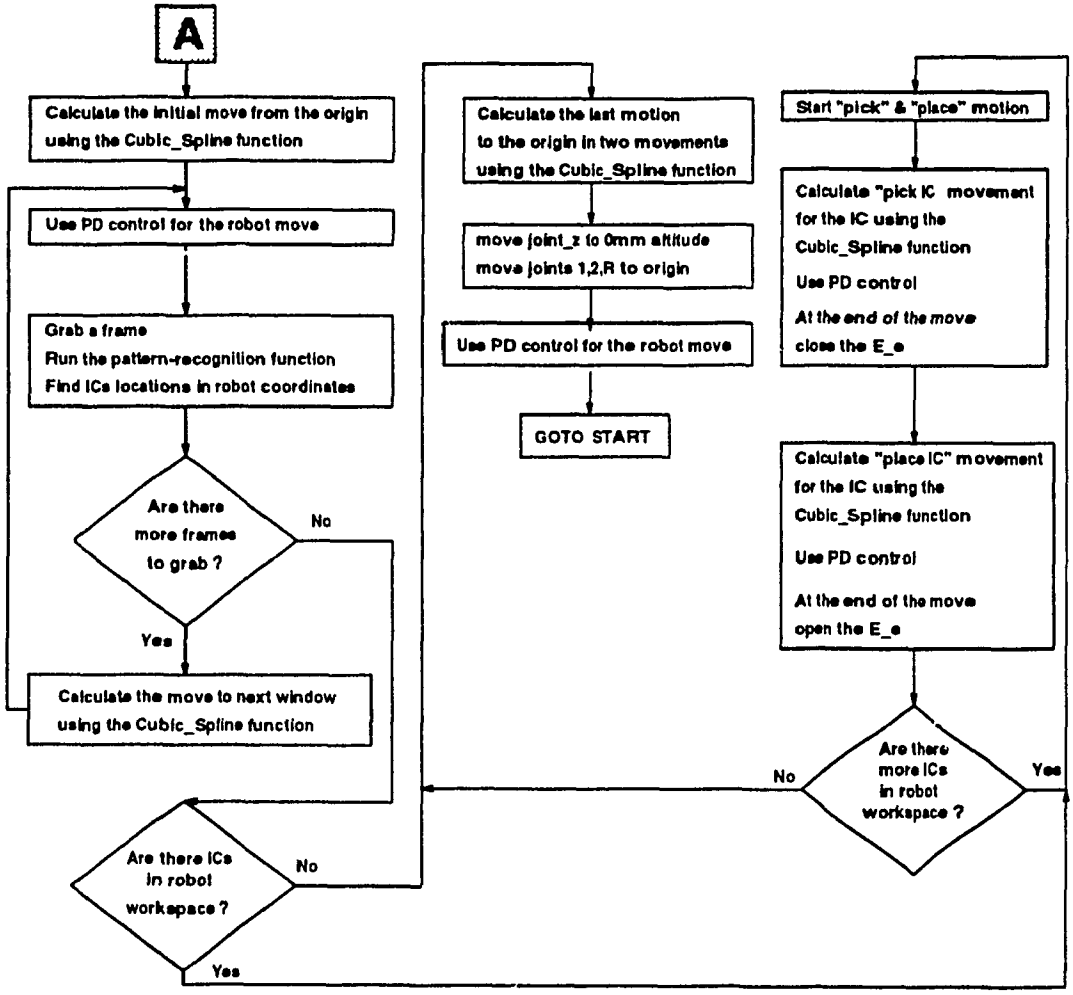

Figure 5.4: General flowchart of the robot modes (second part).

Chapter 6

CONCLUSIONS

An architecture for incorporating visual information in the operation of a robot has been presented. The goal of the thesis was to enhance an Advanced Robot Controller, based on a Master/Slave configuration which controls the IBM7545 with the capability to use input from a camera in its workspace. An important consideration was to create an efficient low cost effective environment. In the design of the system the following aspects were considered.

1. *High Speed System* - Vision information usually involves a huge amount of data (250,000 pixels) which requires fast processing and large amounts of memory. To address this problem within a limited budget, we used an INTEL-486 based PC (which replaced the slower INTEL-286 based PS2/50).

2. *Compatibility* - The PC-AT-BUS is fully compatible with the frame grabber used (OCULUS-300).

3. *Hardware Design* - Several hardware changes were carried out to implement the vision system on the robot:

(a) *New Master/Slave Interface* - Special care was taken in choosing the interfacing ICs in order to comply with the speed requirements of the faster INTEL-486 processor. It was necessary to ensure that the BUSY signal from the DPR met the timing constraints.

(b) *E_e Control* - New design of the Master/Slave control of the Robot E_e was carried out. This was not available in the earlier version.

(c) *Lighting Environment Control* - A special hardware design lets the INTEL-486 processor control the lighting environment for object recognition (fluorescent light) and to distinguish between colors (Halogen light).

4. *Software Design* :

(a) *Master Software Design* - The software consists of two main parts. The first is the Pattern-Recognition function based on the input from the frame grabber. The second part is in the form of continuous movement based on PD control and Cubic-Spline trajectory generation, to perform "pick and place" operations.

(b) *Slave Software Design* - Additional software was added to the Slave control procedure in order to obtain full control of the Robot E_e.

5. *Integration* - Full integration between all the components of the system to perform real-time tasks (Camera, Frame-Grabber, Master and Slave) was carried.

The theory developed for the pattern-recognition function was shown to work very effectively when implemented on (dual in line) ICs. The same method may be implemented for any electronic component that satisfies the main requirement of symmetry and shape (rectangle, disc, square and triangle), and is flat (the distance

of the object surface to the table should be small compared to the distance of the camera to the table).

When the image resolution is low and/or not much detail about an object is available, (i.e., the number of pixels/mm is small) in order to keep good accuracy of the pattern-recognition function, convolution should not be performed. This will tend to reduce the accuracy of important features (object corners), and enhance the effect of errors (due to IC pins, spots etc...). Our analysis and experimental results indicate that with the vision system incorporated with the IBM 7545, it can perform almost $\underline{real-time}$ "pick and place" operations. An important aspect of our design is that it is low-cost and can be easily adapted for other industrial robotic systems.

Up to three more cameras may be connected and controlled by the COREC'O frame grabber. A second camera can be incorporated to provide data for fine motion. This can be used to perform guarded motion or even achieve contact. If a force controller is incorporated as well, one can perform insertion of different types of IC's on a printed circuit board.

Another interesting application (possibly not in real-time) would be to enable the robot E_e to track an object moving in its workspace. Since the average time of recognition of simple objects is less than 0.2 sec. and path computation is 0.1 sec. the robot's response time with respect to object movement in its workspace is 0.3 sec.

Bibliography

[1] J. N. Brodkin, *The Design and Implementation of an Advanced Robot Controller.* M.A.Sc. Thesis, Dept. of Electrical & Computer Engineering, Concordia University, Montreal, Quebec, Canada. November 1990.

[2] C. Michaud, *Multi-Robot Workcell with Vision for Integrated Circuit Assembly.* M. Eng. Thesis, Dept. of Electrical Engineering, McGill University, Montreal, Quebec, Canada. July 1986.

[3] A. R. Mansouri and A. Malowany, "Using Vision Feedback in Printed-Circuit Board Assembly". *1985 IEEE Microprocessor Forum*, Atlantic City, New Jersey. pp. 115-122 April 2-4, 1985.

[4] J. S. Weszka, "A Survey of Threshold Selection Techniques". *Computer Graphics and Image Processing* vol. 7, pp. 259-265, 1978.

[5] J. S. Weszka and A. Rosenfeld, "Threshold Evaluation Techniques". *IEEE Transaction on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 8 Aug. 1978.

[6] A. R. Mansouri, A. F. Malowany, and M.D. Levine, "Line Detection in Digital Pictures: A Hypothesis Prediction/Verification Paradigm", *Computer Vision, Graphics, and Image Processing*, vol. 40, pp. 95-114 1987.

[7] "Robot with vision system picks parts and places them on hybrid substrate", *Electronics Magazine*, vol. 17, pp. 136-137 January 1984.

[8] N. B. Freeman, "Assembly- assisted by vision, robot helps build circuits", *American Machinist*, vol. 8, pp. 115-117, May 1985.

[9] D. Horn, "Machine Vision: The Guiding Light", *Mechanical Engineering*, vol. 19, pp. 40-43, June 1989.

[10] John J. Craig, *Introduction to Robotics Mechanics and Control*, Second Edition, Addison-Wesley. Reading, MA, 1989.

[11] Pamela McCorduck, *Artificial Intelligence-Machines Who Think*. W.H. Freeman, San Francisco, CA, 1989.

[12] *IBM 7545 Manufacturing System Hardware Library*. IBM Corporation, Boca Raton, Florida, 1984.

[13] L. C. Eggebrecht, *Interfacing to the IBM Personal Computer*. PC AT System and Bus Architecture, Chapter 10 pp. 267-285, second edition 1990, Howard W. Sams & Company.

[14] *EV80C196KA Microcontroller Evaluation Board User's Manual*. Intel Corporation, Santa Clara, California, Release 001, March 20, 1988.

[15] *OC-300 Real-Time Image Digitizing and Processing Board*, User's Manual, Edition 1.00, Coreco Inc., Montreal, Canada, July 1988.

[16] *VK-C360 Color Camera*, Instruction Manual, HITACHI, Japan.

[17] *High Performance CMOS Data Book*, Integrated Device Technology, Santa Clara, California, 1988.

[18] *AC Solid State Relays*, GORDOS ARKANSAS, G Series, Rogers, Arkansas, USA.

[19] Shao Lejun, R. A. Volz, L. Conway, M. W. Walker, "Tele-Robot Control Involving Contact and Time-Delay", *Computer, Communication and Networking Systems: An Integrated Perspective. Proc. of the International Conference on Information Engineering - ICIE*, vol. 1, pp. 160-169, Elsevier, Amsterdam, Netherlands, 1991.

[20] Shao Lejun, R. A. Volz, "Robot Vision Calibration Under Constraint conditions", *Computer, Communication and Networking Systems: An Integrated Perspective. Proc. of the International Conference on Information Engineering - ICIE*, vol. 1 pp. 55-64, Elsevier, Amsterdam, Netherland, 1991.

[21] V. Graefe, K. Fleder, "A Powerful and Flexible Co-processor for Feature Extraction in a Robot Vision System", *Proc. IECON 1991, International Conference on Industrial Electronics, Control and Instrumentation*, vol. 3 pp. 2019-2024, IEEE, New York, NY, USA, 1991.

[22] R. A. Jarvis, "3D Shape and Surface Color Sensor Fusion for Robot Vision", *Robotica*, Vol. 10, pp. 389-396, 1992, UK.

[23] H. Kobayashi, K. Uchida, Y. Matsuzaki, "A Self-Learning Robot Vision System". *IEEE International Joint Conference on Neural Networks*. vol.3 pp. 2007-2012. IEEE, New york, NY, USA.

[24] R. Sharma, J. Y. Herve, P. Cucka, "A Framework for Vision-Guided Manipulator of a Moving Target", *Conference Proceedings IEEE International Conference on Systems, Man, and Cybernetics.*, vol. 1, pp. 213-218, IEEE, New York, NY, USA, 1991.

[25] A. Nagchaudhuri, M. Thint, D. P. Garg, "Camera-Robot Transform for Vision-Guided in a Manufacturing Work Cell", *Journal of Intelligent and Robotic systems: Theory and Applications*, vol. 5, pp. 283-298, Netherlands, 1992.

[26] L. G. Trabasso, C. Zielinski, "Semi-Automatic Calibration Procedure for the Vision-Robot Interface Applied to Scale Model Decoration", *Robotica*, vol. 10 pp.303-308, UK, 1992.

[27] C. C. Wang, "Extrinsic Calibration of o Vision Sensor Mounted on a Robot", *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 161-175, USA, 1992.

[28] M. J. Daily, "Self-Calibration of a Multi-Camera Vision System", *Conference Record. Twenty-Fourth Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 810-815, San Jose, CA, USA, 1990.

[29] P. K. Sinha, M. Maamri, "Use of TV Cameras for Robot Position Determination", *International Conference on Image Processing and Its Applications*, pp. 567-570, London, UK, 1992.

[30] A. J. Vayda, A. C. Kak, "A Robot Vision System for Generic Object Recognition", *Workshop on Directions in Automated CAD-Based Vision*, pp. 166-175, IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1991.

[31] M. Ito, "Robot vision Modelling-Camera Modelling and Camera Calibration", *Advanced Robotics*, Vol. 5, pp. 321-335, Netherlands, 1991.

[32] L. Annell, M. Torngren, "Transputer-Based Machine Vision System Research at the University of Oulu", *Nordic Transputer Applications. Proc. of the 1st and 2nd Nordic Transputer Seminars*, pp. 112-116, Amsterdam, Netherlands, 1991.

[33] N. R. Schofield, *Low Cost, Real Time, Robot Vision System, with a Cluster-Based Learning Capability*, Univ. Durham, UK, 1988.

[34] S. Sakane, T. Sato, M. Kakikura, "Automatic Planning of Light Source Placement for an Active Photometric Stereo System", *Proc. IROS. IEEE International Workshop on Intelligent Robots and Systems. Towards a New Frontier of Applications*, vol. 2, pp. 559-566, IEEE, New York, NY, USA, 1990.

[35] V. Graefe "The BVV-Family of Robot Vision Systems", *Intelligent Motion Control. Proc. IEEE International Workshop*, vol. 1, pp. 55-65, New York, NY, USA, 1990.

[36] H. Inoue, T. Tachikawa, M. Inaba, "Robot Vision System with a Correlation Chip for Real-Time Tracking, Optical Flow and Depth Map Generation", *Proc. IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1621-1626. Los Alamitos, CA, USA, 1992.

[37] A. K. C. Wong "Intelligent Robotics Research at Waterloo", *Proc. International Conference on Manufacturing Automation*, pp. 349-354, Univ. Hong Kong, Hong Kong, 1992.

[38] M. Rechsteiner, B. Schneuwly, W. Guggenbuhl, "Robots in Postal Service. A Fast and Robust Vision System for Parcel Separation", *Bulletin Des Schweizerischen Elektrotechnischen Vereins & Des Verbandes Schweizerischer Elektrizitaetswerke*, vol. 83, pp. 19-26, Switzerland, 1992.

[39] F. Leonard, G. Abba, E. Ostertag, D. Mehdi, "Closed Loop Robot Control by Real Time Visual Sensor", *Robot Control (SYROCO 1991). Selected Papers from the 3rd IFAC/IFIP/IMACS Symposium*, pp. 507-512, UK, 1991.

[40] A. T. de Almeida, U. C. Nunes, J. M. Dias, H. J. Araujo and J. Batista, "A Distributed Control Network for Sensory Robotics", *Microprocessors in Robotics and Manufacturing Systems*, Chapter 9, pp. 217-235, Kluwer Academic Publishers, Boston, MA, USA, 1992.

[41] E. Oliveira, R. Camacho and C. Ramos, "A Multi-Agent Environment in Robotics", *Robotica*, vol. 9, pp. 431-440, UK, 1991.

[42] J. H. Kim and S. C. Hyung, "Real-Time Determination of a Mobile Robot's Position by Linear Scanning of a Landmark", *Robotica*, vol. 10, pp. 309-319, UK, 1992.

[43] Z. Bien, H. Y. Known, J. Youn, "A Closed Form 3D Self-Positioning Algorithm for a Mobile Robot Using Vision and Guide-Marks", *Robotica*, vol. 9, pp. 265-274, UK, 1991.

[44] Theo A. G. Heeren, Frans E. Veldpaus, "An Optical System to Measure the End Effector Position for On-Line Control Purposes", *The International Journal of Robotics Research*, vol. 11 pp. 53-63, MA, USA, 1992.

Appendix A

Calibration of the Robot Vision System

Figures A.1 and A.2 illustrate the approach used to evaluate the fixed distance, *d_t_c*, the distance from E_e to the camera's closest corner, and the fixed angle, *b_angl*, the angle between the second link and the End-effector to Ref_Point line.

With reference to Figure A.1, the following steps are carried out to perform the calibration of the vision system.

Input: End-effector location relative to the origin (0,0).

Output: b_angle, d_t_c.

1. Turn on the system and initialize the robot by calling the *find_home* procedure.
2. Choose a *Ref_Point* in the corner closest to the E_e of the camera window.
3. *First Stage*,
   (a) Choose an arbitrary point *A* on the robot coordinates.
   (b) Enter continuous frame grabbing mode of the Camera and move the robot arm such that point *A* and the *Ref_Point* overlap exactly.

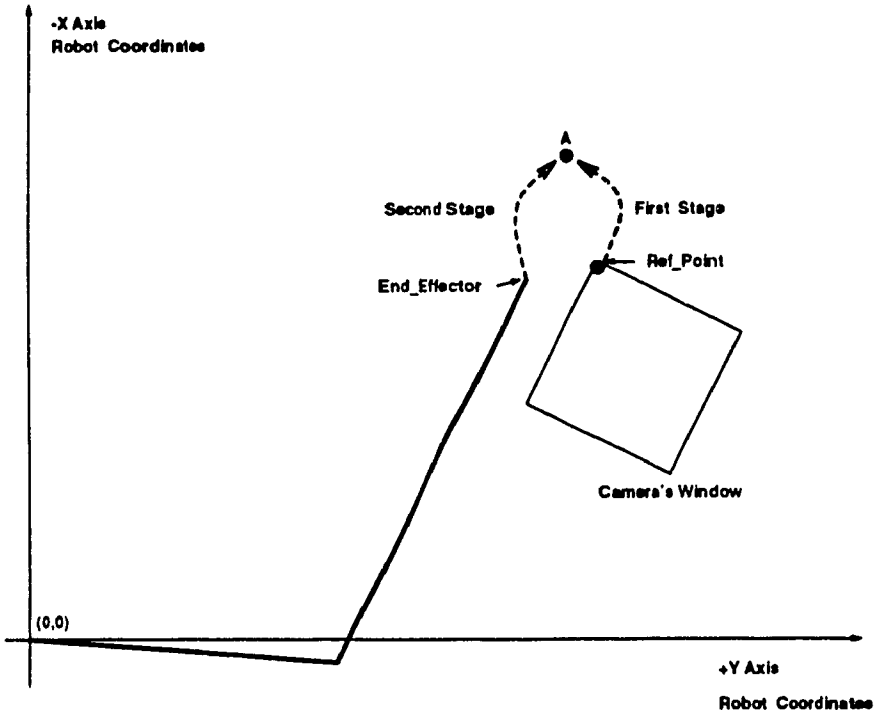

Figure A.1: Description of the two stage measurements when calibrating the vision system.

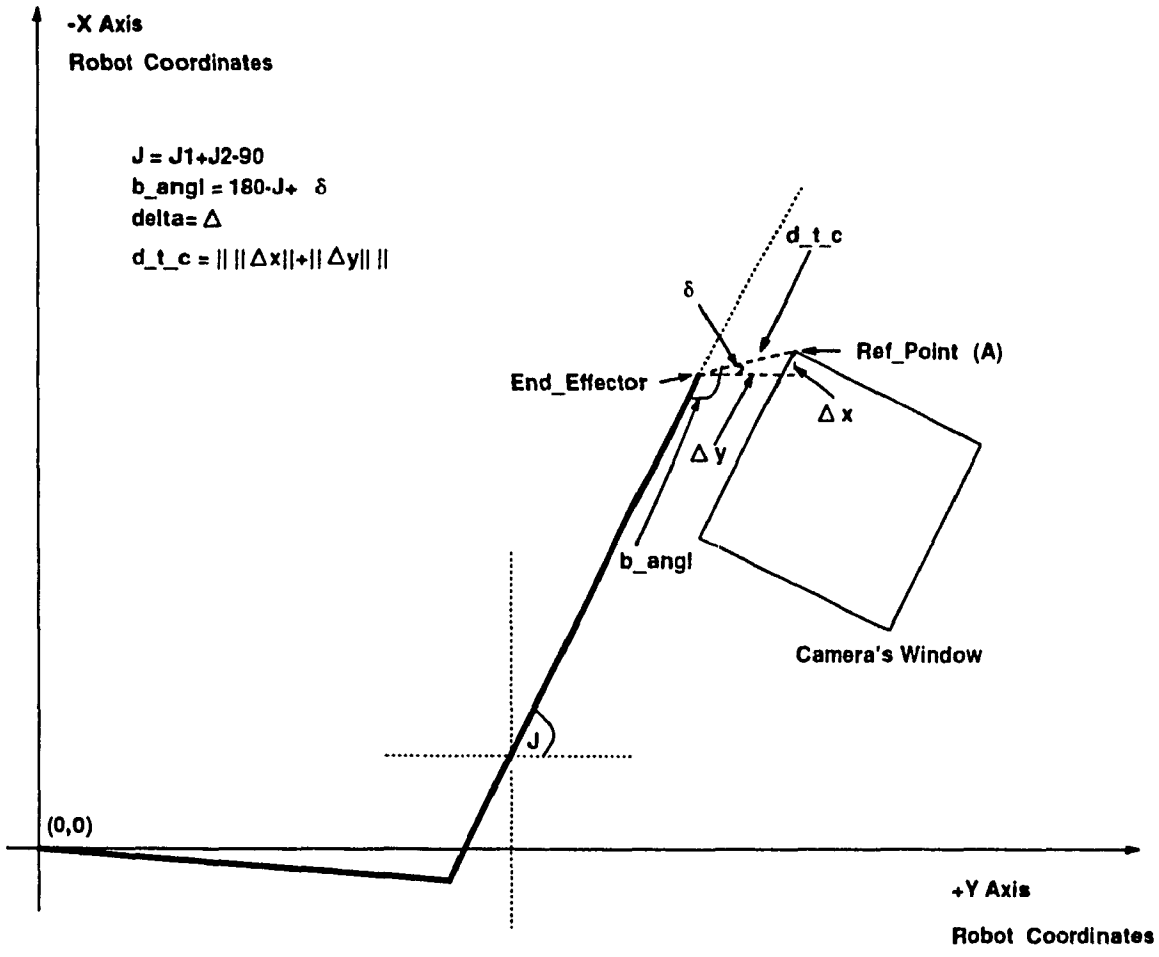

Figure A.2: Computation of d_t_c and b_angl.

(c) Exit the continuous frame grabbing mode and call the *nrobot* program which performs the robot PD control, and can be used to find the E_e position.

(d) Read the E_e location in robot coordinates (x_1, y_1) and the values of J1 (joint-1) and J2 (joint-2) in degrees.

4. *Second Stage*,

(a) Move the robot arm such that the center of the E_e is located precisely above point A.

(b) Read the new E_e location in robot coordinates (x_2, y_2).

5. From Figure A.2, we calculate Δx and Δy as

$$\Delta x = x_1 - x_2 \tag{A.1}$$

and,

$$\Delta y = y_1 - y_2 \tag{A.2}$$

This gives

$$\delta = \arctan(\frac{\Delta x}{\Delta y}) \tag{A.3}$$

and

$$d\_t\_c = \sqrt[2]{\Delta x^2 + \Delta y^2} \tag{A.4}$$

$$b\_angl = 180^0 - (J1 + J2 - 90^0) + \delta \tag{A.5}$$

6. End of algorithm

The calibration procedure is repeated 2 or 3 times and the average of the results are used to obtain values for *d_t_c* and *b_angl*.

After completing the above procedure, we find *d_t_c* to within an error of up to ± 0.5mm and *b_angl* to within an error of up to $\pm 0.5^0$. We Then perform the final *On Line Calibration* procedure to obtain more accurate values of *d_t_c*, *b_angl*, *angl_err*, *link_1* and *link_2*. It should be noted that the final value of *angl_err* is within $\pm 3^0$ and the values of the robot link lengths *link_1* and *link_2* are in the range of $[399 - 401]mm$ and $[249 - 251]mm$ respectively (based on the manufacturer's data).

Appendix B

Master Software Description

Some parts of the Master software subroutines which control the Master's movement and trajectory planning were taken from [1] (find_home(), robot_error_check(), cubic_spline(), joint_torques, prop_deriv_control(), ...).

```
/* UTILITY PROGRAMS AND FUNCTIONS FOR USE IN VISUAL ROBOT CONTROL */
#include <fg3.h>
#include <fg3demo.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <math.h>
#include <dos.h>

/* GLOBAL VARIABLES (begin with a capital letter) */
 double Duration; /* Duration (in sec.) of the move */

/* position error gains */
 double kp_1 = 60, kp_2 = 60, kp_Z = 55, kp_R = 60;
/* velocity error gains */
 double kv_1 = .1, kv_2 = .1, kv_Z = .1, kv_R = .1;
/* explanation for all the following variables is in chapter 4. sections 3-4. */
 double c_x=1/3.56, c_y=1/3.48, b_angl=137.7, d_t_c=81.8, d_err=0, angl_err=1.8;
 double link_1=400.00, link_2=250.00;
 int b_x=3, b_y=457;
    int Home_found = 0; /* A flag which is set to 1 when home is found */
 int Flag_a = 0, Flag_b = 0, fn_ics=0, Ini_step = 0, N=0, K=0, Stop;
```

```
 int C0 = 0, take_picture=0;
 int far *gripper = 0xc000802a;
 long int In_pos[10][3], Out_pos[10][3];
 double choose_new_gain(char *, double);

/*****************************************************************************/
main(argc, argv)
int argc;
char *argv[];
{ *gripper = 0x0000;
  _stklen=10000;
 if(fg3env(argc,argv)<0)
  {
 fg3free();
 exit(1);
  }
 for(;;) {
/* fn_ics=0;*/
Stop = 0;
clrscr();
controller_menu();

    }
}
/*****************************************************************************/
int check_home_found_flag(); /* function prototypes */
prop_control();
prop_deriv_control();
find_home();

controller_menu()
{
    char ch;

    printf("Choose controller\n\n");
    printf("[1]  FIND HOME\n");
 printf("[2]  VISUAL SERVOING OF ROBOT\n");
 printf("[3]  CHANGE OF PARAMETERS ?\n");
    printf("[0]  QUIT\n\n");
    printf("ENTER YOUR CHOICE: ");
```

```
    do {
switch(ch = getch()) {
    case '1': find_home();
      return;
 case '2': /* check_home_found_flag(); */
      prop_deriv_control();
return;
 case '3': change_parameters();
return;
 case '0': fg3free();
clrscr();
      exit(0);
    default : delline();
      printf("\rINVALID CHOICE. TRY AGAIN: ");
        }
 } while(ch != '1' && ch != '2' && ch != '3' && ch != '0');
}


/*******************************  ********************************************/

/* this function was established in order to allow calibration of robot
   and video " ON-LINE ". */
change_parameters()
{
  char ch;

clrscr();
printf("\t\t\tPD CONTROL & VIDEO PARAMETERS\n\n");
printf("The position error gains are:\n");
printf("kp_1 = %-5.2f  kp_2 = %-5.2f  kp_Z = %-5.2f  kp_R = %-5.2f\n\n"
       ,kp_1,kp_2,kp_Z,kp_R);
printf("The velocity error gains are:\n");
printf("kv_1 = %-5.2f  kv_2 = %-5.2f  kv_Z = %-5.2f  kv_R = %-5.2f\n\n"
       ,kv_1,kv_2,kv_Z,kv_R);
printf("link_1 = %-6.2f link_2 = %-6.2f\n\n\n",link_1, link_2);
printf("The number of VIDEO pixels per one mm. in the work space are:\n");
printf("X axis = %-4.2f pixels   Y axis = %-4.2f pixels\n\n",1/c_x,1/c_y);
printf("The location of the reference point is:\n");
printf("pixel num.  %d  on X axis and pixel num.  %d  on Y axis\n\n",b_x,b_y);
```

```
printf("The value of measured b_angl is  %-5.2f  degrees\n\n",b_angl);
printf("The distance of measured tip to corner of picture is  %-5.2f  mm.\n\n"
       ,d_t_c);
printf("The value of angl_err is  %-5.2f degrees\n\n\n",angl_err);
printf("Change any of the parameters? [y/n]:");
if(getche()=='y') {
clrscr();
printf("\tChoose group:\n\n");
printf("[1]  ERROR GAIN AND LINK_1(2) PARAMETERS\n");
printf("[2]  VIDEO PARAMETERS\n");
printf("[0]  QUIT\n\n");
printf("ENTER YOUR CHOICE: ");
do {
  switch(ch = getch())  {
  case  '1': gain_change();
 return;
  case  '2': video_change();
 return;
  case  '0': return;

  default  : delline();
 printf("\rINVALID CHOICE. TRY AGAIN: ");
 }
  } while(ch != '1' && ch != '2' && ch != '0' );
}

getch();

}
/*************************************************************************/
gain_change()
{
printf("\n\n");
kp_1 = choose_new_gain("Kp_1", kp_1);
kp_2 = choose_new_gain("Kp_2", kp_2);
kp_Z = choose_new_gain("Kp_Z", kp_Z);
kp_R = choose_new_gain("Kp_R", kp_R);
printf("\n");
kv_1 = choose_new_gain("Kv_1", kv_1);
kv_2 = choose_new_gain("Kv_2", kv_2);
```

```
kv_Z = choose_new_gain("Kv_Z", kv_Z);
kv_R = choose_new_gain("Kv_R", kv_R);
        printf("\n");
        link_1 = choose_new_gain("link_1", link_1);
        link_2 = choose_new_gain("link_2", link_2);

}
/*****************************************************************************/
video_change()
{
printf("\n\n");
b_angl = choose_new_gain("b_angl", b_angl);
d_t_c  = choose_new_gain("d_t_c", d_t_c);
angl_err = choose_new_gain("angl_err", angl_err);
}
/*****************************************************************************/
/* change the value of a gain or parameter */

double choose_new_gain(char *name, double gain)
{
printf("Change %s? [y/n]: ", name);
if(getch() == 'y') {
printf("\tFrom %lf to: ", gain);
scanf("%lf", &gain);
}
else printf("\n");

return(gain);
}

/*****************************************************************************/

find_home()   /* tells 196 to find home */
{
    int far *command = 0xC0008006;
    unsigned int far *error = 0xC0008008;

    *error = 0x0000; /* clear the error register */
 *command = 0x0080; /* give 'find home' command */
    clrscr();
```

```
    printf("\t\t\tFINDING HOME, PLEASE WAIT");
    while(*command); /* wait for 196 to finish the move */
    Home_found = 1;
    delline();
}

/*****************************************************************************/
/* check for joint overuns */

robot_error_check()
{
    int far *error_pointer = 0xC0008008;
    int e;

    *error_pointer = *error_pointer & 0x00ff;
    if(*error_pointer) {
do {
    e = *error_pointer;
    clrscr();
    printf("\t\t\tROBOT ERRORS\n\n");
            if(e & 0x0001) printf("Joint 1 in positive overun\n");
    if(e & 0x0002) printf("Joint 1 in negative overun\n");
    if(e & 0x0004) printf("Joint 2 in positive overun\n");
    if(e & 0x0008) printf("Joint 2 in negative overun\n");
    if(e & 0x0010) printf("Joint z in positive overun\n");
    if(e & 0x0020) printf("Joint z in negative overun\n");
    if(e & 0x0040) printf("Joint r in positive overun\n");
    if(e & 0x0080) printf("Joint r in negative overun\n");
    printf("\nMANUALLY MOVE THESE JOINTS BACK INTO THE WORKSPACE");
    while(e == *error_pointer);
} while(*error_pointer);
        printf("\n\nAll overun errors fixed.  Press any key to continue ");
        while(!getch());
        return;
    }
}

/*****************************************************************************/
/* display errors */
```

```
display_errors()
{
    unsigned int far *error = 0xC0008008;
    unsigned int temp;

    temp = *error;
    if(temp & 0x0001) printf("\tjoint 1 entered positive overun\n");
    if(temp & 0x0002) printf("\tjoint 1 entered negative overun\n");
    if(temp & 0x0004) printf("\tjoint 2 entered positive overun\n");
    if(temp & 0x0008) printf("\tjoint 2 entered negative overun\n");
    if(temp & 0x0010) printf("\tjoint z entered positive overun\n");
    if(temp & 0x0020) printf("\tjoint z entered negative overun\n");
    if(temp & 0x0040) printf("\tjoint r entered positive overun\n");
    if(temp & 0x0080) printf("\tjoint r entered negative overun\n");
    if(temp & 0x0100) printf("\tToo much torque was applied to joint 1\n");
    if(temp & 0x0200) printf("\tToo much torque was applied to joint 2\n");
    if(temp & 0x0400) printf("\tToo much torque was applied to joint z\n");
    if(temp & 0x0800) printf("\tToo much torque was applied to joint r\n");
    if(temp & 0x1000) printf("\tPS/2 did not provide torques in time\n");
    if(temp & 0x2000) printf("\tPS/2 did not finish its loop in time\n");
}

/*****************************************************************************/
/* compare two numbers (double precision).
 * return 0 if their signs are the same.
 * return 1 if their signs are different or if at least one number is zero. */

cmp_sign(double num_1, double num_2)
{
    if(num_1 && num_2){  /* if neither number is zero */
if( (num_1 > 0 && num_2 > 0) || (num_1 < 0 && num_2 < 0) ) return 0;
    }
    else return 1;
}

/*****************************************************************************/
```

```
/*************************************************************************
 * pat_rec.c file is a basic general pattern-recognition function       *
 * which checks the data from the frame grabbed by the camera and       *
 * then separates the data into partial groups(partial-objects).        *
 * In the next stage the skeleton of each partial group is extracted    *
 * and the function joins skeleton partial groups into skeleton         *
 * groups that share the same data(object). In the last stage the       *
 * function locates the external rectangle of each object and compute   *
 * the center of each external rectangle before calling the library     *
 * functions.                                                           *
 *************************************************************************/
#include <fg3.h>
#include <fg3demo.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <math.h>

extern int fn_ics;
int edgp[20][4][2], f_cntr[20][2], fg_p=-1, *fp[20][5], box[20][6];
double jt, jt1, jt2;
/* edgp[][][] - collects data about 20 rectangles and the value(x,y)
                of each of the four corners in camera's coordinates.
   f_cntr[][] - the center (x,y) of each ICs in camera's
                coordinates.
   fg_p       - final number of groups(ICs) start from 0(first group).
   *fp[][5]   - 5 pointers for each final group pointer 0 is always
                the first address of the group poiters 1,2 usually
                found within the group and pointers 3,4 are always
                at the end of the object's group.
   box[][]    - contains the external rectangle values. box[][0,1],
                contain the center of the external rectangle and
                box[][2-5] contain the side's values.
   jt1        - the value of the first joint in degrees.
   jt2        - the value of the second joint in degrees.          */
pattern_recognition()
   {
byte far  *strt;
   long int huge *pr = 0xc0008010;
long int j1 = *pr;
```

```
long int j2 = *(pr+1);
int *dstn, *s_dstn, *p[30][5], *l_p, *m_p, *h_p;
int x, y, n=-1, i, j=0, g_p=0, n_ics=0, scnd_b=0;
int y_glvl, x_glvl, gp_f=0, tmp_g_p;
int k, tmp[4][2], break_1;

/*  *strt     - points to the adress of the frame buffer.
  *pr & *(pr+1) - point to joint1 and joint2 addresses in DPR.
 *s_dstn     - points to the first data address where all
               the groups are mixed all together.
   *dstn     - points to the last address where all the
               groups are mixed all together.
   *p[][5]   - 5 pointers for each partial group with the same
               separation as in *fp[][5].
   *l,m,h_p  - point to three consecutive values of "y".
     n       - number of words collected by *dstn.
       g_p   - number of partial groups.
      n_ics  - equal to (g_p-1).


 fn_ics=0;
 jt1=j1/872.2222;
 jt2=j2/444.444;
fg3roa(35,35);/* offset and rage initialization */
convo_ini( 0);
/* convo(0); Perform horizontal sobel covolution */
convo_end();
frop_1(); /* grab a frame */
dstn = (int  *) malloc(60000) ; /* allocates 60000 bytes for*/
                                    /* data collected from the  */
                                    /* frame buffer.            */
if(!dstn)   {
  printf("OUT OF MEMORY");
  exit(1);  }
s_dstn=dstn;
if((strt = fg3omap(0,BYTE)) == NULL)  { fg3cmap();
                                        exit(1); }
/*  the data is collected in four stages each stage depends on the
    board limit of 64kbytes.*/
```

```
for(y=0;y<=127   ;y++)  {/* first stage */
  for(x=0;x<=511  ;x++)  {
    if((y!=0) && (*strt<150)) {/* if the value of the pixel is*/
      (*dstn)=y;              /* less than 150(graylevel)    */
      dstn++;                 /* keep the index values in    */
      (*dstn)=x;              /* the allocated area.         */
      dstn++;
      n=n+2;
}
    strt++;
  }
 }

if(fg3romap(1) != 0)  {/* second stage */
 fg3cmap();
 exit(1);             }
 for(y=128;y<=255  ;y++)  {
   for(x=0;x<=511  ;x++)  {
     if(*strt<150)        {
       (*dstn)=y;
       dstn++;
       (*dstn)=x;
       dstn++;
       n=n+2;
  }
     strt++ ;
  }
  }

if(fg3romap(2) != 0)  {/* third stage */
  fg3cmap();
  exit(1);            }
for(y=256;y<=383  ;y++)  {
  for(x=0;x<=511  ;x++)  {
    if(*strt<150)        {
      (*dstn)=y;
      dstn++;
      (*dstn)=x;
      dstn++;
      n=n+2;
```

```
 }
    strt++ ;
 }
 }

if(fg3romap(3) != 0)  {/* fourth stage*/
fg3cmap();
exit(1);              }
  for(y=384;y<=480  ;y++)  {
    for(x=0;x<=511  ;x++)  {
      if(*strt<150)        {
(*dstn)=y;
dstn++;
(*dstn)=x;
dstn++;
n=n+2;
   }
      strt++ ;
   }
   }
fg3cmap();
/*****22222222222222222222222222222222222222222222222222222222222222222********/
  dstn=s_dstn;
  printf("\n%i" ,n);

  if(n != -1)   { /* if the workspace is not empty open the first partial group*/
    p[g_p][0] = (int *) malloc(10000);/* locate 10,000 bytes four each partial */
    if(!p[g_p][0]) {                  /* group.                                */
      printf("OUT OF MEMORY");
      exit(1);      }
    p[g_p][4] = p[g_p][3] = p[g_p][2] = p[g_p][1] = p[g_p][0];
    n_ics=1;
    p[g_p][4]++;
    *p[g_p][3]=*dstn;
    dstn++;
    *p[g_p][4]=*dstn;
    dstn++;

    j=1;
    for(i=2;i<=n;i+=2)    {
```

```
      y_glvl=*dstn;
      dstn++;
      x_glvl=*dstn;
      dstn++;
      gp_f=1;

/* check if the current pixel belongs to the first group saying, if the
   distance to the last pixel or the pixels before is less than 20 pixels */
      if(sqrt(pow(*p[g_p][3]-y_glvl,2)+pow(*p[g_p][4]-x_glvl,2))<=20)   {
j++;
p[g_p][3]=p[g_p][3]+2;
p[g_p][4]=p[g_p][4]+2;/* compare to the last pixel in first group*/
*p[g_p][3]=y_glvl;    /* joint the pixel to the first group if   */
*p[g_p][4]=x_glvl;    /* less than 20 pixels distance.           */
gp_f=0;                                                          }


      else                                        {
/* compare the current pixel to the last 34 pixels in the first group and
   joint if the distance is less than 20 pixels. */
if((p[g_p][3]-p[g_p][0]) < 132)     {
  p[g_p][1] = p[g_p][0];
  p[g_p][2] = p[g_p][0]+1;          }

else                                {
  p[g_p][1] = p[g_p][3] - 132;
  p[g_p][2] = p[g_p][4] - 132;       }
for(;;)        {
  if(sqrt(pow(*p[g_p][1]-y_glvl,2)+pow(*p[g_p][2]-x_glvl,2))<=20)     {
    j++;
    p[g_p][3]+=2;
    p[g_p][4]+=2;
    *p[g_p][3]=y_glvl;
    *p[g_p][4]=x_glvl;
    gp_f=0;
    break;                                                          }
    if(p[g_p][1]==p[g_p][3])  break;
    p[g_p][1]+=2;
    p[g_p][2]+=2;
  }
```

```
      }
 /* if no match found for the first partial group than open a new partial
    group and place the index values at the beginning of the next partial
    group. for each additional pixel compare the distance of the pixel in
    following order:
    1) compare to the last pixel in the current partial group.
    2) compare to the last 43 pixels in the current partial group.
    3) compere to the last 43 pixels in the other partial groups.
    4) open a new partial group. */

 /*    compare to the last 43 pixels in the other partial groups */
  if(gp_f==1)    {
    tmp_g_p=g_p;
    for(g_p=0;g_p<=(n_ics-1);g_p++)      {
    if(g_p!=tmp_g_p)                      {
      if((p[g_p][3]-p[g_p][0])<132)          {
p[g_p][1] = p[g_p][0];
p[g_p][2] = p[g_p][0] +1;            }
      else                                   {
       p[g_p][1] = p[g_p][3] - 132;
       p[g_p][2] = p[g_p][4] - 132;          }
      for(;;)   {
if(sqrt(pow(*p[g_p][1]-y_glvl,2)+pow(*p[g_p][2]-x_glvl,2))<=20)    {
p[g_p][3]+=2;
p[g_p][4]+=2;
*p[g_p][3]=y_glvl;
*p[g_p][4]=x_glvl;
gp_f=0;
scnd_b=1;
break;                                              }
      if(p[g_p][1]==p[g_p][3])   break;
      p[g_p][1]+=2;
      p[g_p][2]+=2;
  }
 }
      if(scnd_b==1)  {
scnd_b=0;
break;       }
}
 /* open a new partial group. */
```

```
      if(gp_f==1)    {
n_ics++;
g_p=n_ics-1;
p[g_p][0]=(int *) malloc(10000);
if (!p[g_p][0])            {
  printf("\nOUT OF MEMORY");
  exit(1);                  }
p[g_p][4]=p[g_p][3]=p[g_p][2]=p[g_p][1]=p[g_p][0];
*p[g_p][3]=y_glvl;
p[g_p][4]++;
*p[g_p][4]=x_glvl;
gp_f=0;
       }
    }
 }

/****333333333333333333333333333333333333333333333333333333333333333333*****/
 free(s_dstn);
/* having the all partial groups we get rid of the filled data keeping only
   the skeleton. the skeleton is found by keeping the first and last pixel's
   indexes of each line in the camera's coordinates. */
for(g_p=0;g_p<=n_ics-1;g_p++)   {
  l_p=p[g_p][0];
  p[g_p][1]=l_p+1;
  m_p=l_p+2;
  h_p=m_p+2;
  for(;;)   {
 if(l_p==p[g_p][3])  break;
 if(m_p==p[g_p][3])          {
p[g_p][1]++;
*p[g_p][1]=*m_p;
p[g_p][1]++;
m_p++;
*p[g_p][1]=*m_p;
p[g_p][4]=p[g_p][1];
p[g_p][3]=p[g_p][1]-1;
break;                     }

 if((*l_p!=*m_p) || (*m_p!=*h_p))     {
p[g_p][1]++;
```

```
*p[g_p][1]=*m_p;
p[g_p][1]++;
m_p++;
*p[g_p][1]=*m_p;
m_p++;                                  }
 else  m_p+=2;
 l_p+=2;
 h_p+=2;
 }
 }
/* having the skeleton of partial groups we compare each partial group
   to another if one pixel in one skeleton group is in distance of less than
   20 pixels to the other skeleton group we join the second group to the
   first skeleton partial group and cancelling the second by putting
   the value -1 in the first address location.
   at the end of the process we formed skeleton's objects. */
for(g_p=0;g_p<=n_ics-1;g_p++)     {
  for(i=g_p+1;i<=n_ics-1;i++)       {
 break_1=0;
 if((*p[g_p][0]!=-1) && (*p[i][0]!=-1))     {
p[g_p][1]=p[g_p][0];
p[g_p][2]=p[g_p][0]+1;
p[i][1]=p[i][0];
p[i][2]=p[i][0]+1;
for(;;)                    {
if(sqrt(pow(*p[g_p][1]- *p[i][1],2)+pow(*p[g_p][2]- *p[i][2],2))<=20) {
  p[i][1]=p[i][0];
  p[i][2]=p[i][0]+1;
  for(;;)                    {
 p[g_p][3]+=2;
 p[g_p][4]+=2;
 *p[g_p][3]=*p[i][1];
 *p[g_p][4]=*p[i][2];
 p[i][1]+=2;
 p[i][2]+=2;
 if(p[i][1] > p[i][3])  {
*p[i][0]=-1;
break_1=1;
break;               }
 }
```

```
  }
  if(break_1==1) break;
  else            {
 p[g_p][1]+=2;
 p[g_p][2]+=2;
 if(p[g_p][1]>p[g_p][3])      {
p[g_p][1]=p[g_p][0];
p[g_p][2]=p[g_p][0]+1;
p[i][1]+=2;
p[i][2]+=2;
if((p[i][1]>p[i][3]) || ((p[i][1]-p[i][0])>20))  break;
  }
  }
  }
  }
  }
  }
 /* assign the final skeleton's objects to fp[][[] and discard skeleton
    groups with less than 6 pixels(dirt). */
 for(g_p=0;g_p<=n_ics-1;g_p++)       {
if((*p[g_p][0]!=-1) && (p[g_p][3]-p[g_p][0] >20))     {
  fg_p++;
  for(i=0;i<=4;i++)    { fp[fg_p][i]=p[g_p][i];}
  fn_ics++;                                          }
  }


 printf("\n%i",n_ics);
 printf("\n%i",fn_ics);

/* find the external rectangle to each object by looking for min,max of {x},{y}
   and then calculate the center of the external rectangle. */

 for(i=0;i<=19;i++)       {
box[i][2]=511;
box[i][3]=0;
box[i][4]=480;
box[i][5]=0;          }

 for(fg_p=0;fg_p<=fn_ics-1;fg_p++)      {
```

```
fp[fg_p][1]=fp[fg_p][0];
fp[fg_p][2]=fp[fg_p][0] + 1;
for(;;)                                  {
  if(*fp[fg_p][1]<box[fg_p][4])  box[fg_p][4]=*fp[fg_p][1];
  if(*fp[fg_p][1]>box[fg_p][5])  box[fg_p][5]=*fp[fg_p][1];
  if(*fp[fg_p][2]<box[fg_p][2])  box[fg_p][2]=*fp[fg_p][2];
  if(*fp[fg_p][2]>box[fg_p][3])  box[fg_p][3]=*fp[fg_p][2];

  if(fp[fg_p][1]==fp[fg_p][3])    {
 box[fg_p][1]=(box[fg_p][2]+box[fg_p][3])/2;/*X*/
 box[fg_p][0]=(box[fg_p][4]+box[fg_p][5])/2;/*Y*/
 printf("\n%d %d %d",fg_p+1,box[fg_p][0],box[fg_p][1]);
 break;                          }

  fp[fg_p][1]+=2;
  fp[fg_p][2]+=2;                        }
  }

 rectangle_corners(); /* library function */
 rectangle_pos_orient();/* library function */



  }

for(g_p=0;g_p<=(n_ics-1);g_p++)  {free(p[g_p][0]); } /* free all pointers */

}
```

```
/*****************************************************************************/
/* rec_corner.c locates the four corners of the rectangle from a given       */
/* skeleton in camera coordinates in multi-object(ICs) input.                */
/*****************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>

extern int fg_p, fn_ics, *fp[20][5], box[20][6], edgp[20][4][2], f_cntr[20][2];
rectangle_corners()
{
float gdst, ldst, nr, fr;
/* gdst-the longest distance from the center of the external rectangle to any */
/*      point given by the skeleton of an object.                            */
/* ldst-a variable of gdst, nr, fr.                                          */
/* nr- When locating the 4 corners we have to put these points in order. "nr" */
/*     is the closest corner to the first corner and,                        */
/* fr-  is the farthest corner.                                              */
int   ymx, xmx, t_l_l, t_l_m, k, trd_b, tmp[4][2], i, y0=241, x0=255;
/* ymx- hold the corner with the highest y value(first corner)               */
/* xmx- Is the x axis value of ymx                                           */
/* t_l_l- The index value of the closest corner                              */
/* t_l_m- The index value of the farthest corner                             */
/* tmp[4][ ]-tmp[0][] holds the corner which comply with "ymx", tmp[1][]     */
/* holds the value of the corner in the same side of the first corner.       */
/* tmp[2][ ]-holds the corner of the other side of the rectangle saying,     */
/* the nearest corner to tmp[1][ ] and tmp[3][ ] holds the last corner.      */
/* (y0,x0)- hold the value of the center pixel of the camera.                */
double i_c_h=8.5, c_i_d=420;
/* i_c_h-distance of IC face to table in mm.                                 */
/* c_i_d-distance of camera to table in mm.                                  */
/************FINDING***FOUR*****EDGE*****POINTS*******************************/
for(fg_p=0;fg_p<=fn_ics-1;fg_p++)   {
  fp[fg_p][1]=fp[fg_p][0];/* initializing beginning of groups */
  fp[fg_p][2]=fp[fg_p][0]+1;/* fg[fg_p][1]=y & fg[fg_p][2]=x */
  gdst=0;
  i=0;/* corners' index */
    for(;;)  {
```

```
      trd_b=0;
      ldst=sqrt(pow(*fp[fg_p][1]-box[fg_p][0],2)+
                pow(*fp[fg_p][2]-box[fg_p][1],2));
if(ldst>gdst)    {
  if((i>0) && (i<=3))   { /* If the first longest distance
                                    point(corner) found then check that
                                    the other three points are in 14 pixels
                                    distance of one to the next one */
    for(k=0;k<=i-1;k++)   {
      if(sqrt(pow(*fp[fg_p][1]-edgp[fg_p][k][0],2)+
                     pow(*fp[fg_p][2]-edgp[fg_p][k][1],2))<=14)  {
      trd_b=1;
      break;     }
  }
}

  if(trd_b==0)   {
    gdst=ldst;
    edgp[fg_p][i][0]=*fp[fg_p][1];
    edgp[fg_p][i][1]=*fp[fg_p][2];   }
 }

  if(fp[fg_p][1]==fp[fg_p][3]) { /*If the end of group then starts from
                                       the beginning for the next corner */
    i++;
    if(i>3)    { /* If all the four corners found then calculate the center
                          of IC in camera coordinates */
      f_cntr[fg_p][0]=0;
      f_cntr[fg_p][1]=0;
       for(k=0;k<=?;k++)  {
edgp[fg_p][k][0]=floor(edgp[fg_p][k][0]+(y0-edgp[fg_p][k][0])*i_c_h/c_i_d+.5);
edgp[fg_p][k][1]=floor(edgp[fg_p][k][1]+(x0-edgp[fg_p][k][1])*i_c_h/c_i_d+.5);
 f_cntr[fg_p][0]+=edgp[fg_p][k][0];
 f_cntr[fg_p][1]+=edgp[fg_p][k][1];
  }
      f_cntr[fg_p][0]=f_cntr[fg_p][0]/4;
      f_cntr[fg_p][1]=f_cntr[fg_p][1]/4;
      break;  }

  else      {
```

```
   fp[fg_p][1]=fp[fg_p][0];
   fp[fg_p][2]=fp[fg_p][0]+1;
   gdst=0;
   continue;  }
}
     fp[fg_p][1]+=2;  /* check next pixel */
     fp[fg_p][2]+=2;
   }
      }
/*****PUTTING******POINTS********IN***********ORDER*****************************/
      for(fg_p=0;fg_p<=fn_ics-1;fg_p++)     {
t_l_m=0;
ymx=edgp[fg_p][0][0];/* Find the corner with the highest value of y */
for(i=1;i<=3;i++)     {
  if(ymx<edgp[fg_p][i][0])     {
  ymx=edgp[fg_p][i][0];
  t_l_m=i;                }/* t_l_m contains the index of "ymx" */
      }
xmx=edgp[fg_p][t_l_m][1];
edgp[fg_p][t_l_m][0]=edgp[fg_p][0][0];
edgp[fg_p][t_l_m][1]=edgp[fg_p][0][1];
edgp[fg_p][0][0]=ymx;
edgp[fg_p][0][1]=xmx;

nr=150.0;
fr=0.0;
for(i=1;i<=3;i++)          { /* Finding the farthest corner(fr) and the closest
                                    corner(nr) to "ymx" corner */
  ldst=sqrt(pow(edgp[fg_p][0][0]-edgp[fg_p][i][0],2)+
                   pow(edgp[fg_p][0][1]-edgp[fg_p][i][1],2));
  if(ldst>fr)   {
    fr=ldst;
    t_l_m=i;    }
    if(ldst<nr)   {
      nr=ldst;
      t_l_l=i;    }
 }
      for(i=1;i<=3;i++)   {
tmp[i][0]=edgp[fg_p][i][0];
tmp[i][1]=edgp[fg_p][i][1];  }
```

```
edgp[fg_p][2][0]=tmp[t_l_l][0]; /* assigning the order which is
                                                    specified above */
edgp[fg_p][2][1]=tmp[t_l_l][1]; /* into edgp[ ][ ][ ] */
edgp[fg_p][3][0]=tmp[t_l_m][0];
edgp[fg_p][3][1]=tmp[t_l_m][1];
for(i=1;i<=3;i++)     {
  if((i!=t_l_l) && (i!=t_l_m))    {
    edgp[fg_p][1][0]=tmp[i][0];
    edgp[fg_p][1][1]=tmp[i][1];   }
      }
printf("\n");
  for(k=0;k<=3;k++)  {
    printf("%d %d   ",edgp[fg_p][k][0],edgp[fg_p][k][1]);
     }
 }
  }
```

```
/*************************************************************************
 * rec_po.c calculate the (x,y) center location and orientation of ICs   *
 * (rectangles)in robot's coordinates. full explanation of the           *
 * procedure and variables are given in chapter 4. sections 3 and 4.     *
 *************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>

extern int fn_ics, edgp[20][4][2], f_cntr[20][2];
extern double jt1, jt2, jt, pi;
extern long int In_pos[10][3];
extern int b_x, b_y;
extern double c_x, c_y, b_angl, angl_err, d_t_c, d_err;
extern double link_1, link_2;

rectangle_pos_orient()
{
int  fg_p;
double dy1, dx1, dy2, dx2, alfa[20], jf, y_angl, beta_err, gama;
double beta_1, phi, X, Y, w, jnt1, jnt2, temp;
double d_c_ic, d_t_ic, theta;



/****FINDING******IC'S*******ORIENTATION***IN****ROBOT***COORDINATES********/
for(fg_p=0;fg_p<=fn_ics-1;fg_p++)      {
dy1=(edgp[fg_p][0][0]-edgp[fg_p][1][0])*(c_y/c_x);
dx1=edgp[fg_p][0][1]-edgp[fg_p][1][1];
dy2=(edgp[fg_p][2][0]-edgp[fg_p][3][0])*(c_y/c_x);
dx2=edgp[fg_p][2][1]-edgp[fg_p][3][1];
  if(dy1==0) dy1=0.01;
  if(dy2==0) dy2=0.01;

  temp=(atan(dx1/dy1)*pi)-(atan(dx2/dy2)*pi);
  if((temp>150) || (temp<-150))   alfa[fg_p] = 90;
  else  alfa[fg_p]=(atan(dx1/dy1)*pi + atan(dx2/dy2)*pi)/2;
```

```
  alfa[fg_p]=-(alfa[fg_p] +jt1+jt2-90-angl_err);
  if(alfa[fg_p]>90) alfa[fg_p]=alfa[fg_p]-180;
  if(alfa[fg_p]<-90) alfa[fg_p]=180+alfa[fg_p];
      }
/*FINDING***X**Y**IC'S**CENTERS**IN**ROBOT**COORDINATES**********************/
jt=jt1+jt2;
d_t_c=d_t_c-d_err;
theta=(180-b_angl)-angl_err;
  for(fg_p=0;fg_p<=fn_ics-1;fg_p++)      {
    if((f_cntr[fg_p][1]-b_x) ==0)    gama=theta +90;
    else  gama=theta+
          atan(c_y*abs(f_cntr[fg_p][0]-b_y)/(abs(f_cntr[fg_p][1]-b_x)*c_x))*pi;
    d_c_ic=sqrt(pow((f_cntr[fg_p][1]-b_x)*c_x,2)+
                pow((f_cntr[fg_p][0]-b_y)*c_y,2));
    d_t_ic=sqrt(d_t_c*d_t_c+d_c_ic*d_c_ic-2*d_t_c*d_c_ic*cos(gama/pi));
    beta_1=acos((d_t_c*d_t_c+d_t_ic*d_t_ic-d_c_ic*d_c_ic)/(2*d_t_ic*d_t_c))*pi;
    phi=180-(b_angl-beta_1)-jt;
         X=link_2*cos((jt)/pi)+link_1*cos(jt1/pi);
         Y=link_2*sin((jt)/pi)+link_1*sin(jt1/pi);


    X=X+d_t_ic*cos(phi/pi);
    Y=Y-d_t_ic*sin(phi/pi);
/****PUTTING****X***Y****R****INSIDE***THE****GLOBAL***TABLE***In_pos[fg_p][i]**/

/* There are three restrictions imposed to allow inverse kinematics */
  if(X<-650 || X>650)    {   /* -1- */
    printf("\nX out of space for ics No. %d",fg_p+1);
    exit(1);
 }
  if(Y<-386 || Y>650)    {   /* -2- */
  printf("\nY out of space for ics No. %d",fg_p+1);
  exit(1);
 }
w=(X*X+Y*Y+97500.0)/800;
  if(X*X+Y*Y>+w*w)       {   /* -3- */
   jnt1=atan2(Y,X)+atan2(-sqrt(X*X+Y*Y-w*w),w);/* Inverse kinematics for joint1*/
    if(jnt1<0)    jnt1=jnt1+ 2*180/pi;

     /* Inverse kinematics for joint2 */
```

```
        jnt2=atan2(-X*sin(jnt1)+Y*cos(jnt1),X*cos(jnt1)+Y*sin(jnt1)-link_1);
    In_pos[fg_p][0]=(long int) (jnt1*pi*872.2222+0.5);
    In_pos[fg_p][1]=(long int) (jnt2*pi*444.4444+0.5);
}
 else    {
    jnt1=-1;
    jnt2=-1;
 }
 if(jnt1<0 || jnt1>174445 ||jnt2<0 || jnt2>60001)  {
    printf("\nOut of space in X-Y plane");
    exit(1);
 }
 In_pos[fg_p][2]=(long int) (alfa[fg_p]*227.5556);

}
    }
```

```
/* subroutine to perform proportional derivative control
   and gripper control. */

#include <alloc.h>
#include <stdio.h>

long int huge *cubic_spline(double); /* prototype */
extern double kp_1, kp_2, kp_Z, kp_R;
extern double kv_1, kv_2, kv_Z, kv_R;
extern double Duration; /* Global variables */

extern int Flag_a, Flag_b, Ini_step, Stop, CO;
extern int far *gripper;

prop_deriv_control()
{
    double kv1, kv2, kvz, kvr;
    double e_1=0, e_2=0, e_Z=0, e_R=0; /* joint position error */
    double el_1, el_2, el_Z, el_R; /* previous joint position error */
    static double sample_per;
    char ch;
    long int huge *pp;  /* pointer to desired (reference) path points */
    long int huge *s_pp; /* a place to save this address */
    unsigned int far *sample_timer = 0xC0008000;
    int far *error = 0xC0008008;
    long int far *act_pos_1 = 0xC0008010;
    long int far *act_pos_2 = 0xC0008014;
    long int far *act_pos_Z = 0xC0008018;
    long int far *act_pos_R = 0xC000801C;
    int far *t_1 = 0xC0008040;
    int far *t_2 = 0xC0008042;
    int far *t_Z = 0xC0008044;
    int far *t_R = 0xC0008046;
    int far *timing_a = 0xC0008002;
    int far *timing_b = 0xC0008004;
    int far *command  = 0xC0008006;


/* DISPLAY CURRENT SAMPLING PERIOD AND PROMPT FOR A CHANGE */
 *sample_timer=1000;
```

```
    sample_per = (double)(*sample_timer)/1000000;

/* CHECK ROBOT FOR OVER RUN ERRORS */
    robot_error_check();

/* GET TRAJECTORY */
 pp = cubic_spline(sample_per);
    s_pp = pp;



/* BEGINNING OF THE MOVE */
    clrscr();
    kv1 = kv_1 / sample_per;
    kv2 = kv_2 / sample_per;
    kvz = kv_Z / sample_per;
    kvr = kv_R / sample_per;
    printf("\t\t\tEXECUTING");
    *command = 0x0001; /* have 196 begin its i_o loop */
    while( (*pp != 0xffff0000) && (*error == 0) ) {

    /* wait to for start of sampling period */
while(!(*timing_a & 0x0001));

    /* update joint position errors */
el_1 = e_1, el_2 = e_2, el_Z = e_Z, el_R = e_R;

    /* wait for joint positions */
while(!(*timing_a & 0x0002));

    /* compute joint torques */
*t_1 = (int)(0x0800 + kp_1*(e_1 = (*pp++ - *act_pos_1)) + kv1*(e_1-el_1));
*t_2 = (int)(0x0800 + kp_2*(e_2 = (*pp++ - *act_pos_2)) + kv2*(e_2-el_2));
*t_Z = (int)(0x0800 + kp_Z*(e_Z = (*pp++ - *act_pos_Z)) + kvz*(e_Z-el_Z));
*t_R = (int)(0x0800 + kp_R*(e_R = (*pp++ - *act_pos_R)) + kvr*(e_R-el_R));
*timing_b = 0x0001;  /* tell 196 that torques are ready */


    /* wait for torques to be accepted */
while(*timing_b && !(*error));
```

```
    /* indicate end of sample period */
*timing_a = 0x0000;
    }

/* INDICATE END OF COMMAND */
    *command = 0x0000;
    *timing_b = 0x0000;

/* DISPLAY SUCCESS OF EXECUTION */
    delline();
    if(*error) {
printf("\n\n\tCommand terminated prematurely because:\n\n");
display_errors();
*error = 0; /* clear the error semaphores */
while(!getch());
Ini_step = 0 ;
Flag_a = 0;
Flag_b = 1;
Stop = 1;
 }
/* while(!getch());*/

/* FREE ALLOCATED MEMORY */
terminate:
   farfree(s_pp);
   if(Stop == 1) return;
   if(C0 !=0)     { /* do not change gripper position */
     if(C0 == 1)          {
       *gripper = 0x0080;/* instruct the slave to close gripper */
       while(*gripper);
       C0 = 0;             }
     else                  {
       *gripper = 0x0001;/* instruct the slave to open the gripper */
       while(*gripper);
       C0 = 0;              }
    }
 prop_deriv_control();
   }
```

```
/* A cubic spline path generator that incorporates via points.
 * Velocity at the via points is automatically choses.
 * See Craig, pp. 198-199. for details.
 * the function generates gripper control mechanism as well. */

#include <alloc.h>
#include <stdio.h>
#include  <math.h>
#include <dos.h>

void get_cartesian_position(); /* prototypes */
void get_joint_position();
void get_time();
cmp_sign(double, double);

extern double Duration;
extern int Flag_a, Flag_b, fn_ics, Ini_step, N, K, Stop, C0, take_picture;

/* In_pos[][] keeps the input location
   and orientation of the ICs in pulses in
   robot's coordinates.
   Out_pos[][] keeps the output location and
   orientation of the ICs in pulses in
   robot's coordinates.*/
extern long int In_pos[10][3], Out_pos[10][3];
long int huge *cubic_spline(double sample_per)

/* const_z1 = -200mm when the robot moves in its workspace.
   const_z2 = -227.7mm when the robot "pick & place" ICs */
{   const double const_z1 = 200*380.96, const_z2 = 227.7*380.96;
    char ch;
 int i, j, l, error, num_via_points;
    double s1, s2; /* slope 1, slope 2 */
 long int pos[4][4]; /* A two dimensional position array.
 * A row for each specified path point.
 * column 0 = joint 1 pulse count
 *   "    1 =   "   2   "     "
 *   "    2 =   "   z   "     "
 *   "    3 =   "   r   "     "   */
    long int position[4];
```

```
 double vel[4][4] = {   /* A two dimensional velocity array */
0,0,0,0, /* A row for each specified path point */
0,0,0,0,     /* column 0 = joint 1 velocity */
0,0,0,0,        /*    "   1 =   "   2    "     */
0,0,0,0 };         /*    "   2 =   "   z    "     */
/*    "   3 =   "   r    "     */
    double a[4][4]; /* two dimensional array of coefficients (Craig p. 198) */
 double time[4];
 double cnst[4] = {
 872.2222,444.4444,380.96,227.5556 };
 int v[4] =    {

/* constant velocities of joint1=45 degrees per second,
                          joint2=50 degrees per second,
                          joint_z = 100mm per second and
                          joint_r 70 degrees per second. */
 45,50,100,70  };
 double t1[4][4];
    double t, t2, t3, tf, tf2, tf3;
    long int huge *path_point; /* pointer to starting address of path */
    long int huge *s_path_point;  /* a place to save this value */
    unsigned long int num_bytes_mem_req;
    long int far *current_pos = 0xC0008010;

/* GET CURRENT POSITION */
 clrscr();
    pos[0][0] = *(current_pos++);
    pos[0][1] = *(current_pos++);
    pos[0][2] = *(current_pos++);
    pos[0][3] = *(current_pos);
    time[0] = 0;

/* SIMULATING OUTPUT POSITIONS OF UP TO 5 ICs */
    Out_pos[0][0] = 84496;
    Out_pos[0][1] = 52967;
    Out_pos[0][2] = 0;
    Out_pos[1][0] = 82262;
    Out_pos[1][1] = 50705;
    Out_pos[1][2] = 0;
    Out_pos[2][0] = 80578;
```

```
   Out_pos[2][1] = 48281;
   Out_pos[2][2] = 0;
   Out_pos[3][0] = 79411;
   Out_pos[3][1] = 45701;
   Out_pos[3][2] = 0;
   Out_pos[4][0] = 78726;
   Out_pos[4][1] = 42948;
   Out_pos[4][2] = 0;

   if(take_picture==1)  {
     take_picture=0;
     sleep(1);
     pattern_recognition();/* call pattern_recognition function */
     getch();
}

   if (fn_ics == K)    {
     Ini_step = 0;
     K = 0;
     Flag_a = 0;       }

   if ((Ini_step == 1) && (fn_ics > K ))   {
     Flag_a = 1;
     num_via_points = 2 ;                  }

/* if there are more ICs then put the In_pos and Out_pos values for the
   next " pick & place " move */
    if ((fn_ics> K) && (Flag_a == 1))        {

/* CO=2 open gripper CO=1 close gripper CO=0 do not change gripper
   position */
      if (N == 0)                {
pos[2][0] = In_pos[K][0];
pos[2][1] = In_pos[K][1];
pos[2][3] = In_pos[K][2];
N = 1;
CO = 1;            }
      else                        {
pos[2][0] = Out_pos[K][0];
pos[2][1] = Out_pos[K][1];
```

```
pos[2][3] = Out_pos[K][2];
N = 0;
K = K + 1;
CO = 2;                    }

      pos[1][0] = pos[0][0];
      pos[1][1] = pos[0][1];
      pos[1][2] = const_z1;
      pos[1][3] = pos[0][3];
      pos[2][2] = pos[1][2];
      pos[3][0] = pos[2][0];
      pos[3][1] = pos[2][1];
      pos[3][2] = const_z2;
      pos[3][3] = pos[2][3];
     }

 if (Flag_a == 0) {
   if (Flag_b == 0)    {

/* first move to the initial window for grabbing a frame */
     num_via_points=0;
     pos[1][0] = 0x00009951;
     pos[1][1] = 0x0000b646;
     pos[1][2] = 0;
     pos[1][3] = 0;
     Flag_b    = 1;
     Ini_step = 1;
     take_picture=1;    }
   else                    {

/* After completing the job return to origin */
     num_via_points=1;
     pos[1][0] = pos[0][0];
     pos[1][1] = pos[0][1];
     pos[1][2] = 0;
     pos[1][3] = pos[0][3];
     pos[2][0] = 0;
     pos[2][1] = 0;
     pos[2][2] = 0;
     pos[2][3] = 0;
```

```
    Flag_b    = 0;
    Ini_step = 0;
    Stop = 1;              }
    }

  for (i=1 ; i <= num_via_points + 1 ; i++)    {
    for (j=0 ; j <=3 ; j++)                                    {
      t1[i][j] = (double) ((pos[i][j] - pos[i-1][j])/(v[j]*cnst[j])) ;
      if (t1[i][j] < 0 ) t1[i][j] = -t1[i][j];
      t1[i][j] = t1[i][j] + 0.2; /* minimum move between two via points */
                                 /* is 0.2 sec.                          */
      if (t1[i][j] > t1[i][0]) t1[i][0] = t1[i][j] ;        }
      time[i] = time[i-1] + t1[i][0] ;
}


   do {
Duration = time[num_via_points+1];
num_bytes_mem_req=(unsigned long int)(4*4*Duration/sample_per+20+100);
    /* 4 = four joints
     * 4 = four bytes per joint per sample
     * Duration/sample_per = total # of samples
     * 20 = four bytes extra per joint + 4 byte flag
     * 100 = safety margin */
        path_point=(long int huge *)farmalloc(num_bytes_mem_req);
        if(!path_point) {
    printf("Insufficient memory for a %lf",Duration);
    printf("second move.\nEnter a shorter duration: ");
}
    } while (!path_point);


/* COMPUTE THE VELOCITIES AT THE VIA POINTS */
    for(i=1; i<=num_via_points; i++) {  /* for each via point */
for(j=0; j<=3; j++){          /* for each joint */
    s1 = (double)(pos[i+1][j]-pos[i][j]) / (time[i+1] - time[i]);
    s2 = (double)(pos[i][j]-pos[i-1][j]) / (time[i] - time[i-1]);
    if(cmp_sign(s1,s2)) vel[i][j] = 0;
    else vel[i][j] = (s1 + s2)/2;  /* the average of the two */
}
```

```
    }

/* COMPUTE AND STORE THE PATH */
    printf("\n\n\t\tCOMPUTING THE PATH.  PLEASE WAIT");
    s_path_point = path_point;
    t = 0.0;
    tf = 0;
    for(i=0; i<=num_via_points; i++) { /* for all points (beginning at 0) */
t = t - tf;
tf = time[i+1] - time[i];
tf2 = tf * tf;
tf3 = tf2 * tf;
for(j=0; j<=3; j++) { /* for each joint */
   a[j][0] = pos[i][j];
   a[j][1] = vel[i][j];
   a[j][2] =  3*(pos[i+1][j]-a[j][0])/tf2 - (2*a[j][1]+vel[i+1][j])/tf;
   a[j][3] = -2*(pos[i+1][j]-a[j][0])/tf3 + (vel[i+1][j]+a[j][1])/tf2;
}
        for(; t <= tf+sample_per/2; t += sample_per) {
    t2 = t * t;
    t3 = t2 * t;
    *path_point++ =(long int)(a[0][0]+a[0][1]*t+a[0][2]*t2+a[0][3]*t3+0.5);
    *path_point++ =(long int)(a[1][0]+a[1][1]*t+a[1][2]*t2+a[1][3]*t3+0.5);
    *path_point++ =(long int)(a[2][0]+a[2][1]*t+a[2][2]*t2+a[2][3]*t3+0.5);
    *path_point++ =(long int)(a[3][0]+a[3][1]*t+a[3][2]*t2+a[3][3]*t3);
}
    }
    *path_point = 0xffff0000;  /* flag end of path */

    return(s_path_point);
}
```

Appendix C

Slave Software Description

The Slave software subroutine used is from [1], the only part added to it is the gripper control which appears in the Slave software under section 'gripper mode'.

```
;;      slave.a96 - 80C196KA assembly code for AVRS slave processor.

$GE ; Expand all MACROs in slave.lst listing
$include(8096.inc) ; Include symbolic definitions from file 8096.inc

;;
;; Storage Reservation for Program Variables and Pointers in the 80C296KA's
;; 232-byte Register File
;;
rseg  at 40h

TEMP1: dsw 1 ; Temporary registers
TEMP2: dsw 1

N0001: dsw 1 ; Registers to hold frequently used
N0002: dsw 1 ; numbers.
N0003: dsw 1
N0004: dsw 1
N0007: dsw 1
N0008: dsw 1
N000F: dsw 1
N0010: dsw 1

SAMPLE_PERIOD: dsw 1 ; Registers to hold pointers to
TIMING_A: dsw 1       ; dual-port RAM registers
```

```
TIMING_B: dsw 1
COMMAND: dsw 1
ERROR: dsw 1
POS: dsw 1
VEL: dsw 1
ACC: dsw 1
TOR: dsw 1
GRP:            dsw     1
CL_GRP:         dsw     1

        LIMIT_SENSORS: dsw 1   ; Register to hold pointer to joint
; overrun sensors and encoder index
; pulses

POS_1:   dsl 1 ; Registers to hold current joint
POS_2: dsl 1 ; positions
POS_Z: dsl 1
POS_R: dsl 1
POS_1L: dsl 1 ; Registers to hold last joint positions
POS_2L: dsl 1
POS_ZL: dsl 1
POS_RL: dsl 1
VEL_1: dsw 1   ; Registers to hold current joint
VEL_2: dsw 1 ; velocities
VEL_Z: dsw 1
VEL_R: dsw 1
VEL_1L: dsw 1   ; Registers to hold last joint velocities
VEL_2L: dsw 1
VEL_ZL: dsw 1
VEL_RL: dsw 1
TORQUE_1: dsw 1 ; Registers to hold joint torques
        TORQUE_2:     dsw 1
TORQUE_Z: dsw 1
TORQUE_R: dsw 1

PCZ2_R14_CS: dsw 1 ; Pointers to position counters
PCZ2_R13_CS: dsw  1 ; Z2 or 1R = joints Z & 2 or 1 & R
PCZ2_R12_CS: dsw 1 ; R12, R13, R14 = register 12, 13, 14
PCZ2_R14_OE: dsw 1 ; CS = chip select
PCZ2_R13_OE: dsw 1 ; OE = output enable
```

```
PCZ2_R12_OE: dsw 1
PC1R_R14_CS: dsw 1
PC1R_R13_CS: dsw 1
PC1R_R12_CS: dsw 1
PC1R_R14_OE:    dsw 1
PC1R_R13_OE: dsw 1
PC1R_R12_OE: dsw 1

PC_1_PC: dsw 1 ; Registers to hold pointers to
PC_2_PC: dsw 1       ; Position counters' program counters
PC_Z_PC: dsw 1
PC_R_PC: dsw 1
PC_1_RESET: dsw 1 ; Registers to hold values to be to
PC_2_RESET: dsw 1 ; sent to position counters'
PC_Z_RESET: dsw 1       ; program counters
PC_R_RESET: dsw 1

DAC_1: dsw 1 ; Registers to hold pointers to the
DAC_2: dsw 1 ; first rank registers of the DACS
DAC_Z: dsw 1
DAC_R: dsw 1
DACS_OUT: dsw 1 ; Register to hold pointer to the
; second rank register of each DAC
CE_5:          dsw     1       ; Gripper address

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;  MAIN_PROGRAM                                ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;
;; Initialization of Program Variables and Pointers
;;
cseg  at 2080h

LD DAC_1, #0AFFCH ; DAC first rank addresses
LD DAC_2, #0AFFAH
LD DAC_Z, #0AFF6H
LD DAC_R, #0AFEEH
LD DACS_OUT, #0AFDEH ; All DACS second rank address
```

```
LD PCZ2_R14_CS, #09494H    ; Position counter addresses
LD PCZ2_R13_CS, #09392H
LD PCZ2_R12_CS, #09292H
LD PCZ2_R14_OE, #09414H
LD      PCZ2_R13_OE, #09312H
LD PCZ2_R12_OE, #09212H
LD PC1R_R14_CS, #094D4H
LD PC1R_R13_CS, #093D2H
LD PC1R_R12_CS, #092D2H
LD PC1R_R14_OE, #09454H
LD PC1R_R13_OE, #09352H
LD PC1R_R12_OE, #09252H
LD PC_1_PC, #085C4H ; Address of program counter register
LD PC_2_PC, #08584H         ;   on the position counters.
LD PC_Z_PC, #08584H ;   1 & R, 2 & Z are equal
LD PC_R_PC, #085C4H ;   since they are accessed in pairs
LD      PC_1_RESET, #0100H      ; Value to reset counter 1 & idle counter R
LD PC_2_RESET, #0001H ; Value to reset counter 2 & idle counter Z
LD PC_Z_RESET, #0100H ; Value to reset counter Z & idle counter 2
LD PC_R_RESET, #0001H ; Value to reset counter R & idle counter 1
LD      CE_5, #0F000H           ; Gripper address
LD      CL_GRP, #0001H          ; Generate 1 to d.f.f. to close the Gripper

LD N0001, #0001H
LD N0002, #0002H
LD N0003, #0003H
LD N0004, #0004H
LD N0007, #0007H
LD N0008, #0008H
LD N000F, #000FH
LD N0010, #0010H

LD     SP, #100H ; Init stack at top of reg. file
        LDB IOC2, #00000001B ; Put TIMER2 into fast increment mode

LD SAMPLE_PERIOD, #0E000H ; Addresses of dual-port RAM registers
LD TIMING_A, #0E002H
LD TIMING_B, #0E004H
LD COMMAND, #0E006H
LD ERROR, #0E008H
```

```
LD POS, #0E010H
LD VEL, #0E020H
LD ACC, #0E030H
LD TOR, #0E040H
LD LIMIT_SENSORS, #0B000H
LD      GRP, #0E02aH            ; the address Gripper in the DPR

ST 0, [ERROR] ; Clear error register
ST 0, [PC_1_PC] ; Software reset all position counter by
ST 0, [PC_2_PC] ;   writing 0 to their program counters
ST      0, [CE_5]       ; Initial to open Gripper

main_loop:
ST 0, [COMMAND] ; Clear Command register
ST      0, [GRP]                ; Clear Gripper register in DPR
wait_for_command:
CALL zero_dacs ; Turn off all servo motors
CALL get_robot_state ; get state of robot
CALL    overrun_check
CMP 0, [ERROR] ; If there is a joint overrun error
JNE wait_for_command ;   keep waiting until it's corrected

LD TEMP1, [COMMAND] ; Read Command register
BBS TEMP1, 0, control_mode  ; If bit 0 is set, enter control_mode
BBS TEMP1, 7, find_home_mode; If bit 7 is set, enter find_home mode
LD      TEMP1, [GRP]
BBS     TEMP1, 0, grp_o         ;If bit 0 is set, enter grp_open mode
BBS     TEMP1, 7, grp_c         ;If bit 7 is set, enter grp_close mode
BR wait_for_command        ; Otherwise keep waiting

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;                          GRIPPER MODE                          ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

grp_o:
 CALL   delay
    ST     0, [CE_5]    ; open Gripper
    CALL   delay
    BR     main_loop
```

```
grp_c:
 CALL   delay
    ST     CL_GRP, [CE_5]    ; close Gripper
    CALL   delay
    BR     main_loop
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;; FIND HOME MODE ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

FH MACRO   joint, positive_torque, negative_torque, home_bit, index_bit
LOCAL nh, fh_1, fh_2, fh_3
LD TEMP1,[LIMIT_SENSORS]
JBS TEMP1, home_bit, fh_1   ;; If HOME sensor is not set, goto fh_1
LD TEMP2, #positive_torque ;;   else move joint away from HOME
ST TEMP2, [DAC_&joint]
ST  0, [DACS_OUT]
nh: LD TEMP1, [LIMIT_SENSORS]  ;; If HOME sensor is still set, goto nh
JBC  TEMP1, home_bit, nh ;; Else keep moving away from HOME
CALL delay                    ;;   during 'delay' period (~2 seconds)
CALL zero_dacs                ;; Remove the positive torque
fh_1: LD TEMP2, #negative_torque ;; Move joint toward home,
ST TEMP2, [DAC_&joint]
ST 0, [DACS_OUT]
fh_2: LD TEMP1, [LIMIT_SENSORS] ;; Wait for home sensor to set,
JBS TEMP1, home_bit, fh_2 ;;   then continue.
fh_3: LD TEMP1, [LIMIT_SENSORS]   ;; Wait for index pulse,
JBC TEMP1, index_bit, fh_3   ;;   then
ST      PC_&joint&_RESET,[PC_&joint&_PC] ;; Set position counter to zero and
CALL zero_dacs ;;   remove the torque
ENDM

find_home_mode:
FH      Z, 900H, 640H, 4, 0
FH 1, 900H, 700H, 7, 2
FH 2, 900H, 700H, 6, 1
FH R, 920H, 6A0H, 5, 3

        BR main_loop

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;  CONTROL MODE ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

control_mode:
CLR TIMER2
ST 0, [TIMING_A] ; Clear the timing registers
ST 0, [TIMING_B]

control_loop:
    ST N0001, [TIMING_A] ; Signal master to begin its servo loop
SCALL get_robot_state ; Get robot state

        SCALL overrun_check ; Check for overruns and end if found
CMP 0,[ERROR]
BNE main_loop

wait_for_torque:
CMP TIMER2, [SAMPLE_PERIOD] ; Wait for torques but end
JGT torque_too_late ;   (with error) if too late.
CMP 0, [COMMAND] ; End (without error) if master
  BE main_loop ;   clears command register.
AND TEMP1, N0001, [TIMING_B]; Check for arrival of torques.
JE wait_for_torque

SCALL check_torques ; Get and check torques.
CMP 0, [ERROR] ; End (with error) if
BNE main_loop ;   torques are out of range.

ST 0, [TIMING_B] ; Tell master that torques were accepted
SCALL torque_out ; Output torques

wait_for_master:
CMP TIMER2, [SAMPLE_PERIOD] ; Wait for master to finish loop but end
JGT master_too_slow ;   (with error) if master is too slow.
CMP 0, [TIMING_A]
JNE wait_for_master
wait_next_period:
CMP TIMER2, [SAMPLE_PERIOD] ; wait for next sample period
JLT wait_next_period
SUB TIMER2, [SAMPLE_PERIOD] ; Reset the sample period timer
```

```
BR control_loop

torque_too_late:
        LD TEMP1, [ERROR]
OR TEMP1, #1000H
ST TEMP1, [ERROR]
BR main_loop

master_too_slow:
LD TEMP1, [ERROR]
OR TEMP1, #2000H;
ST TEMP1, [ERROR]
BR main_loop

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;  GET ROBOT STATE ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
get_robot_state:
;; Read Joint Position

LD  TEMP1, [PC1R_R14_CS] ; 1st read of R14 of position counters 1&R
LD TEMP1, [PCZ2_R14_CS] ; 1st read of R14 of position counters Z&2
ST POS_1, POS_1L ; Store 'last' positions - low word
ST POS_2, POS_2L         ;   and kill > than 1.8 usec.
ST POS_Z, POS_ZL
ST POS_R, POS_RL
LD POS_1, [PC1R_R14_OE] ; 2nd read of R14 of position counters 1&R
LD POS_Z, [PCZ2_R14_OE] ; 2nd read of R14 of position counters Z&2
ST POS_1+2, POS_1L+2 ; Store 'last' positions - high word
ST POS_2+2, POS_2L+2

LD TEMP1, [PC1R_R13_CS] ; 1st read of R13 of position counters 1&R
LD TEMP1, [PCZ2_R13_CS] ; 1st read of R13 of position counters Z&2
STB POS_1+1, POS_R ; Sort data and kill > 1.8 usec.
STB POS_Z+1, POS_2
LD TEMP1, [PC1R_R13_OE] ; 2nd read of R13 of position counters 1&R
STB TEMP1, POS_1+1 ; Sort data
STB TEMP1+1, POS_R+1
LD TEMP1, [PCZ2_R13_OE] ; 2nd read of R13 of position counters Z&2
ST POS_Z+2, POS_ZL+2 ; Store 'last' positions - high word
```

```
ST POS_R+2, POS_RL+2

LD TEMP2, [PC1R_R12_CS] ; 1st read of R12 of position counters 1&R
LD TEMP2, [PCZ2_R12_CS] ; 1st read of R12 of position counters Z&2
STB TEMP1, POS_Z+1          ; Sort data and kill > 1.8 usec.
STB TEMP1+1, POS_2+1
LD TEMP1, [PC1R_R12_OE] ; 2nd read of R12 of position counters 1&R
STB TEMP1, POS_1+2 ; Sort the data
STB TEMP1+1, POS_R+2
LD TEMP1, [PCZ2_R12_OE] ; 2nd read of R12 of position counters Z&2
STB TEMP1, POS_Z+2 ; Sort data
STB TEMP1+1, POS_2+2

CLRB POS_1+3 ; Clear the most significant byte
CLRB POS_2+3
CLRB POS_Z+3
CLRB POS_R+3
        CMPB  POS_1+2, #0FFH ; Set most sig. byte to FF if 2nd most
JNE pos2 ;    significant byte is FF.
LDB POS_1+3, #0FFH
pos2: CMPB POS_2+2, #0FFH
JNE posZ
LDB POS_2+3, #0FFH
posZ: CMPB POS_Z+2, #0FFH
JNE posR
LDB POS_Z+3, #0FFH
posR: CMPB POS_R+2, #0FFH
JNE pos_end
LDB POS_R+3, #0FFH
pos_end:
ST POS_1, [POS] ; Save positions in dual-port RAM
ST POS_1+2, 2[POS]
ST POS_2, 4[POS]
ST POS_2+2, 6[POS]
ST POS_Z, 8[POS]
ST POS_Z+2, 0AH[POS]
ST POS_R, 0CH[POS]
ST POS_R+2, 0EH[POS]

ST N0003, [TIMING_A] ; Positions saved.
```

```
;; Compute Joint Velocity

VELO MACRO JOINT, OFFSET
ST VEL_&JOINT, VEL_&JOINT&L ;; Store Prev. velocity
SUB VEL_&JOINT, POS_&JOINT, POS_&JOINT&L ;; Compute velocity
ST VEL_&JOINT, OFFSET[VEL] ;; Save in DPR
ENDM

VELO 1, 0
VELO 2, 2
VELO Z, 4
VELO R, 6

ST N0007, [TIMING_A] ; Signal to master: velocities ready.

;; Compute Joint Acceleration

ACCEL MACRO JOINT, OFFSET
SUB TEMP1, VEL_&JOINT, VEL_&JOINT&L ;; Compute accelerations
ST TEMP1, OFFSET[ACC] ;; Save in DPR
ENDM

ACCEL 1, 0
ACCEL 2, 2
ACCEL Z, 4
ACCEL R, 6

ST N000F, [TIMING_A] ; Accelerations saved.

RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;; DELAY ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

delay:  LD TEMP2, #2H     ; Generate about a 2 sec. delay
delay0: LD TEMP1, #0FFFFH        ;    by counting down from FFFFH
delay1: DEC TEMP1 ;   8H times
CMP  TEMP1, 0
```

```
JNE delay1
DEC TEMP2
CMP TEMP2, 0
JNE delay0
RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;; ZERO DACS ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

zero_dacs:                     ; Reset all DACS to 0 Vdc (offset = 800H)
        LD TEMP1, #800H
ST TEMP1, [DAC_1] ; Load first rank registers of all DACS
ST TEMP1, [DAC_2]
ST TEMP1, [DAC_Z]
ST TEMP1, [DAC_R]
ST 0, [DACS_OUT] ; load 2nd rank registers of each DAC
RET ;   simultaneously

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;  CHECK JOINT POSITIONS FOR OVERRUN ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

OVERRUN MACRO   JOINT, MIN, MAX, NEG_OVR_FLAG, POS_OVR_FLAG, NEXT_JOINT, MSB
        LOCAL or_a, or_b
CMP POS_&JOINT+2, #0FFFFH ;; If joint is not in negative region
JNE or_a                    ;;   goto or_a
CMP POS_&JOINT, #MIN ;; Else compare its position against
JC or_&NEXT_JOINT ;;   the specified MIN.
OR TEMP1, #NEG_OVR_FLAG    ;;   If position is < MIN, make note of
BR or_&NEXT_JOINT ;;     neg ovrrun error and check next joint
or_a: CMP POS_&JOINT+2, #MSB ;; Compare 3rd byte against MSB:
        JLT or_&NEXT_JOINT ;;   IF position < MSB, check next joint
JGT or_b ;;   IF position > MSB, goto or_b
CMP POS_&JOINT, #MAX ;;   Else compare position against MAX.
JNH or_&NEXT_JOINT ;;     IF < MAX, check next joint
or_b: OR TEMP1, #POS_OVR_FLAG ;; Make note of positive overrun error.
ENDM

overrun_check:
```

```
CLR TEMP1
or_1: OVERRUN 1, OEEF7H, 0B03CH, 2H, 1H, 2, 2H
or_2:   OVERRUN 2, 0F752H, 0EB3EH, 8H, 4H, Z, 0H
or_Z: OVERRUN Z, 0FE83H, 07585H, 20H, 10H, R, 1H
or_R: OVERRUN R, 05F1DH, 0A0E3H, 80H, 40H, end, 0H
or_end: LD TEMP2, [ERROR]  ; Update Error Register
STB TEMP1, TEMP2
ST TEMP2, [ERROR]
RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;         CHECK TORQUES ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CT MACRO JOINT, OFFSET, NEXT_JOINT, ERROR_FLAG
LD TORQUE_&JOINT, OFFSET[TOR] ;; Get torque from DPR register
AND TEMP1, TORQUE_&JOINT, #0F000H   ;; Mask-out all but most sig. byte
JE ct_&NEXT_JOINT ;; If result is zero, check next joint
OR TEMP2, #ERROR_FLAG ;; Else make a note of the error
ENDM

check_torques:
CLR  TEMP2
ct_1: CT 1, 0, 2, 100H
ct_2: CT 2, 2, Z, 200H
ct_Z: CT Z, 4, R, 400H
ct_R: CT R, 6, end, 800H
ct_end: OR TEMP2, [ERROR] ; Update Error Register
ST TEMP2, [ERROR]
RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;; OUTPUT TORQUES ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

torque_out:
ST TORQUE_1, [DAC_1] ; Load first rank registers of DACs.
        ST TORQUE_2, [DAC_2]
ST TORQUE_Z, [DAC_Z]
ST TORQUE_R, [DAC_R]
```

```
ST 0, [DACS_OUT] ; Load second rank register of DACs.
RET

END
```