HAND-EYE COORDINATION OF A ROBOT MANIPULATOR
BASED ON FUZZY LOGIC

by

SUKIR KUMARESAN, B.S.M.E.

A THESIS

IN

COMPUTER SCIENCE

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE

Approved

Accepted

August, 1994

805
T3
1994
No. 50
Cop. 2

ACKNOWLEDGEMENTS

AEI 9791
JAC 8/13/

I am greatly indebted to Dr. Hua Li, the chairperson of my thesis committee, for his encouragement, guidance and support extended in the course of my thesis work and in the preparation of this document. I also sincerely thank the other committee members, Dr. William Marcy and Dr. David Mehrl, for their valued suggestions. I would like to take this opportunity to thank the people at FSI International, Inc., for supporting the work.

Last but not the least, I am indebted to my parents for having been extremely supportive and patient with me.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	ii
LIST OF TABLES	v
LIST OF FIGURES	vi
CHAPTER	
I. INTRODUCTION	1
1.1 Introduction	1
1.2 Objectives Of This Study	2
II. LITERATURE SURVEY	5
2.1 Conventional Calibration Techniques	5
2.2 Calibration Technique With Adaptation Capability	7
III. MATHEMATICAL FORMULATION	11
3.1 Inverse Mapping Problem	11
3.2 Formulation Of The Inverse Mapping	12
3.2.1 Mathematical Algorithm	13
3.3 Mathematical Algorithm For The Environment With Random Disturbances	15
IV. FUZZY LOGIC ALGORITHM	19
4.1 Fuzzification	20
4.2 Fuzzy Control Rules	21
4.3 Defuzzification	22
V. EXPERIMENTS	26
5.1 Experimental Setup	26
5.2 Stationary Environment	29
5.2.1 Implementation of the Algorithm	29
5.2.2 Testing Algorithm Using Robot	30
5.3 Environment with Small Random Disturbances	31
5.3.1 Calibration at the Extreme Points	31
5.3.2 Implementation Of Fuzzy Logic Algorithm	32
5.3.3 Testing The Fuzzy Logic Algorithm	34
5.3.4 Implementation And Testing Of Linear Interpolation	35

VI. RESULTS AND OBSERVATIONS	43
6.1 Fuzzy Logic Algorithm	43
6.2 Linear Interpolation Algorithm	44
6.3 Comparative Study	44
VII. CONCLUSION	52
REFERENCES	53
APPENDIX: SOURCE CODE	55

LIST OF TABLES

6.1 Results of experimentation with fuzzy logic algorithm	47
6.2 Results of experimentation with linear interpolation	48

LIST OF FIGURES

3.1	Location determination of workpiece pose given the distances between feature points in 3D and the position of the points in the image plane	18
4.1	Fuzzy membership functions for various parameters	24
4.2	Fuzzy inference rules	25
5.1	Physical setup of system	36
5.2	Robot arm axes	37
5.3	Robot arm transmissions	37
5.4	Operating range with a gripper	38
5.5	Operating range without a gripper	38
5.6	Relative position between camera and robot	39
5.7	Boundaries reached by the end effector in the workarea	40
5.8	Image for camera at position A which is an upper position with the camera conventionally calibrated.	41
5.9	Image for camera at position B which is the lower position with the camera conventionally calibrated	42
6.1	Error in fuzzy logic algorithm	49
6.2	Error in linear interpolation algorithm	50
6.3	Comparison of fuzzy and linear interpolation algorithms	51

CHAPTER I

INTRODUCTION

In this chapter, a brief introduction to machine-vision systems and applications is given. The problems associated with the hand-eye coordination are described and the objective of this study is explained.

1.1 Introduction

Robots have become a major element in today's industrial world. They are used in repeated, hazardous and monotonous tasks like picking and placing tasks and basic assembly operations. By using robots, people are freed from such unfulfilling jobs. Also using robots will lead to an increase in autonomy and would increase productivity, reliability and precision of task execution.

Tasks in automated manufacturing systems involving robot operations are often guided by machine vision systems. In their normal mode of operation, these systems take a picture, analyze the image, extract position information and move the robot accordingly. It is estimated that by the year 2000, about 90% of the robots in use will use vision feedback.

To track random parts, a robot must have vision feedback. This feedback gives position information necessary for the end-effector to grab the object. Changes in the task or the environment involve the time and expense of adjusting and recalibrating the system. Therefore, there is a need to develop a faster and more

reliable camera calibration algorithm to cope with these changes, so that the flexibility of a programmable multiple degree-of-freedom manipulator guided by vision can be achieved to accomplish different tasks without kitting trays and precise part feeders in a flexible manufacturing environment. It also permits short production runs and easily changed designs.

1.2 Objectives Of This Study

For the hand-eye coordination of a robot, there should be integration of a camera or cameras into a robotic system to provide information of the relative position and orientation (pose) between robot's end effector and the workpiece. We would like to use a single camera to estimate the relative pose of the robot's gripper to the workpiece. This pose is then fed to the robot controller where control adjustments are made.

To analyze the visual feedback problem, the relationship between the robot's gripper, the stationary camera and the workpiece is needed. The vision system sends the estimated pose of workpiece with respect to the fixed base to the robot control system.

Camera calibration computes the transformation from the camera space to a given world space. Hand-eye calibration is completed by finding the transformation from the given world space to the robot manipulator space.

A monocular system will locate an object if it is on a plane perpendicular to camera's focal axis at a fixed distance from camera. The major difficulty is due to the fact that the projection of a three-dimensional object on to a two-dimensional plane results in the loss of depth information. Therefore, given a point (x,y) on an image plane, it is generally not possible to reconstruct its depth information from the projection plane without imposing additional constraints. This problem is called the inverse mapping problem which is ill-posed in the sense of Hadamard as: (1) no solution exists, (2) the solution may not be unique, or (3) for a small change in the conditions, there would be a significant change in the solution [Poggio and Koch 1985]. It is the purpose of this study to investigate, and to develop a relatively reliable algorithm to solve the hand-eye coordination problem.

In real-world applications, very often the camera position may not be fixed due to random disturbances. An example is that the camera could be used for different purposes and every time the camera is mounted for the hand-eye coordination system or its position changed, it has to be recalibrated. Therefore, another question related to this is how to develop an algorithm to cope with these changes, and in particular if two known positions are given, how to utilize this information to calibrate the camera when it is placed between these two points.

In particular, we would like to answer the following questions:

1. How do we solve the inverse mapping problem by imposing acceptable constraints?

2. Given the two known positions where the camera has been calibrated, how do we utilize this information for the calibration of the camera when it is placed between these two points?

We are interested in solving the above problems by using fuzzy logic to take care of changes in the position of the camera within certain limits.

CHAPTER II

LITERATURE SURVEY

This section explains the current trends in the research of hand-eye coordination of a robot manipulator. Initially the research in this area was focussed on conventional calibration techniques. Recently, the research emphasis has been placed on the area of developing new calibration techniques with learning or adaptation capability. This trend is discussed in this section.

2.1 Conventional Calibration Techniques

The problem of camera calibration has been extensively studied. Tsai [1986] classifies the calibration methods for a camera at a fixed position into five categories:

1. Techniques involving full-scale optimization: This technique allows easy adaptation of an arbitrarily accurate but complex model for imaging. The major disadvantage is that this technique is computationally-intensive and nonlinear [Faig 1975, Abdel Aziz and Karara 1971].
2. Techniques involving computation of perspective transformation and solving of linear equations: Here no nonlinear optimization is needed. Given the 3D world coordinates of a large number of points and the corresponding 2D image coordinates, the coefficients in the perspective transformation matrix

can be solved by least square solution of a system of linear equations

[Sutherland 1974, Yakimovsky and Cunningham 1978].

3. The two-plane method was developed by Martins et al [1987]. Linear equations need to be solved. The main disadvantage is that the transformation between image and object coordinates is empirically based only.
4. Geometric Technique, developed by Fischer and Bolles [1980], uses a geometric construction to derive a direct solution for the camera location and orientation. None of the camera intrinsic parameters need be computed. An important disadvantage is that lens distortion was not considered.
5. Tsai's method: Tsai [1986] developed a technique for three dimensional camera calibration for machine vision. This technique is aimed at efficient computation of camera external position, orientation relative to the object reference coordinate system, effective focal length, and radial lens distortion as well as image scanning parameters.

The process of calibrating points on a calibration block is utilized to uniquely define a one-to-one correspondence between image points and its 3D position [Lenz and Tsai 1987, Faugeras et al. 1987]. Using this technique, the three-dimensional position of the points in world coordinates can be determined. Tsai and Lenz [1987] showed that the transformation of the world coordinate system to the robot coordinate system is difficult to measure precisely.

Weiss has used an approach called the “look and move” approach [Weiss 1984]. It consists of two phases. In the “look” phase, the vision system identifies and locates the centroid of the workpiece in the camera’s field of view to obtain the x - y coordinate, specified in pixels. The pixel-based coordinate is transformed into a real world coordinate by applying a suitable algorithm. In the “move” phase, the robot controller evaluates a suitable trajectory comprised of a set of desired joint positions to move the end effector from its current position to the target position. The feedback is provided by a sensor mounted on each joint of the robot. The major drawback of this approach is the very high cost of executing a reliable algorithm for evaluating the set of joint positions.

Until recently, for most vision-based manipulator control systems, hand-eye calibration is carried out before the operation of the system. Liang et al. [1989] classified this type of calibration as static batch calibration.

2.2 Calibration Technique With Adaptation Capability

It is desired that a hand-eye system be adaptive and autonomous. Unexpected disturbances to a hand-eye system often occur during or after the operation of robot motion. An example is that the camera could be used for various purposes. Changes to the position of the camera need recalibration of the hand-eye system. Therefore, this effect has to be taken into consideration. In such cases, a spatial relation between the camera space and the robot space is not fixed, and the

intrinsic and extrinsic parameters are changing. The hand-eye system is expected to adapt to the working environment and task requirement in real time.

An approach based on a visual-feedback learning process, known as *adaptive calibration*, was first proposed by Liang et al. [1988]. In this approach, the hand-eye system calibration is performed in real-time while the system is operating. The hand-eye transformation is learned dynamically through repeated trials until a satisfactory result is obtained.

A dynamic learning process through an automatic repetition of operation trials has been discussed by Arimoto et al. [1985] and Miller [1987]. A simple system to intercept a ping-pong ball using a one-dimensional degree freedom of robot and a single camera has been demonstrated by Skaar et al. [1987]. Brooks et al. [1987] addressed the self-calibration problem of motion and stereo vision for mobile robots obstacle avoidance. Weiss et al. [1987] developed a strategy for dynamic image feature based servo control of robots. The goal of all the above research was to provide the capability of dynamically calibrating a hand-eye system in real time.

Recursive computation of the transformation between the camera space and the world space for a robot hand-eye system using Kalman filtering has been addressed by Izaguirre et al. [1987], Faugeras et al. [1987] and Liang et al. [1989].

Izaguirre et al. [1987] developed a calibration algorithm for a stereo vision system mounted on a platform with four degrees of freedom. The problem is

non-linear and approximation by linearization is used to compute the transformation recursively using a Kalman filter.

Faugeras et al. [1987] proposed using a Kalman filter for camera calibration to deal with the uncertainty of the parameters. The calibration was static and carried out using a calibration block before the operation of the system. They used a one-stage extended non-linear Kalman filter to calibrate all the parameters. Since the extended Kalman filter is not optimal in general and numerically unstable, Liang et al. [1989] proposed a linear Kalman filter. They used a statistical hypothesis test to detect large parameter changes. If a change is detected, the filter is reinitialized accordingly.

General drawbacks of these approaches include (1) intensive computational requirements for parameter identification, (2) high sensitivity to sensor noise, and (3) the selection of a system model. Due to this, several investigators have discussed possibilities for the application of neural networks.

Kuperstein [1987] developed an algorithm for visual-motor coordination in multijoint robots using a parallel architecture. There has been relatively little progress in the nature of vision-feedback experiments using an actual robot system. This is due to the computational speed and stability problems using complex neural networks.

Walter and Schulten [1993] implemented self-organizing neural networks for the visuo-motor control of an industrial robot. Martinez et al. [1990] developed a three-dimensional neural net for learning visuo-motor coordination of a robot arm.

As mentioned above, there has been successful implementation of neural nets for the calibration of hand-eye systems. There are some limitations of the neural network approach, which include a computationally intensive training process, and stability concern for the network. We would like to investigate the use of fuzzy logic for hand-eye calibration.

CHAPTER III

MATHEMATICAL FORMULATION

In this chapter, we will give the mathematical formulation based on the analysis of the hand-eye calibration problem, which is also referred to as the *inverse mapping problem*. Initially, we define the problem. In the next section, we formulate the problem, and give a brief description of the fuzzy logic algorithm for solving the problem.

3.1 Inverse Mapping Problem

A hand-eye system needs visual feedback for the robotic manipulator to go to the position of the workpiece. The camera takes the picture. This picture is then analyzed using image processing techniques to extract the position information. This information is sent to the robot for it to go and pick the workpiece. The main difficulty is that the depth of the object is not known. This problem in nature is called the *inverse mapping problem* with the formal definition given in the next section [Fischer and Bolles 1980].

Given the two-dimensional image positions of the workpiece's features, the model of the workpiece, the camera parameters, the pose of the robot's end-effector, and the pose of the camera, we have to determine the workpiece's pose with respect to the world (robot) coordinate system. In analyzing the visual

feedback of a hand-eye coordination system, the spatial relationships between the cameras, the workpiece, and the robot's gripper should be initially measured.

3.2 Formulation Of The Inverse Mapping

The inverse mapping problem can be described as follows. Given the two-dimensional image of the workarea, it is not possible to reconstruct the three-dimensional features without the knowledge of the depth information. The formulation is as follows. Assume that the coordinates of any three points (P_1, P_2, P_3) in world coordinates are known. We need three points to determine the pose of the workpiece. This is because a minimum of three points are required to define a plane. The plane equation is given by $Ax + By + Cz + D = 0$. The coefficients A , B , and C define the normal to the plane in a vector form (A, B, C) . The equation of the normal has three unknowns (A, B, C) . So we need a minimum of three points to compute the coefficients. Thus, we can use a geometric model to determine the pose of the workpiece.

For a given point, we can compute its image coordinate by using the linear homogeneous transformation method. We use a 3×3 homogeneous transformation matrix for this purpose. Using the coordinates of the already known points, we should be able to compute the image coordinates of any point using this method.

By using the inverse homogeneous transformation method, we should be able to determine the value of the (x, y, z) coordinates of a position from its image

coordinates. An assumption has to be made regarding the uniform depth of the workpiece. The assumption is that all the workpieces are of the same height. So we need to determine only the (x, y) coordinate of a position. This reduces the three-dimensional inverse transformation matrix to a two-dimensional matrix.

3.2.1 Mathematical Algorithm

Before we describe the algorithm, the camera coordinate system and the world coordinate system are defined. The camera (viewer) coordinate system is defined as the coordinate system of the mounted camera, whose origin coincides with the camera as illustrated in Figure 3.1 [Fischer and Bolles 1980]. The world (robot) coordinate system is defined as the coordinate system whose origin coincides with the origin of the robot as shown in Figure 3.1. The notations for describing the coordinates in various coordinate systems are as follows:

- A point in the world coordinate system is described as (x_i, y_i, z_i) .
- The same point, whose world coordinates are (x_i, y_i, z_i) has (x'_i, y'_i, z'_i) as its camera coordinates.
- The perspective projection of (x'_i, y'_i, z'_i) on the image plane is (x''_i, y''_i, z''_i) .

Let v'_i be a point $(x_i, y_i, z_i)^T$ of the workpiece and v''_i be its projection to image plane in the camera coordinate system, which is,

$$v''_i = (x''_i, y''_i, z''_i)^T. \quad (3.1)$$

The perspective projection can be described mathematically by

$$v_i'' = T_i * v_i', \quad (3.2)$$

where T_i is nonsingular 3×3 matrix.

Suppose

$$v_i = (x_i, y_i, z_i), \quad (3.3)$$

are the coordinates of v_i' in the world coordinate system, then the transformation from the world coordinate system to viewer coordinate system is given by,

$$v_i' = T_r * v_i + v_s, \quad (3.4)$$

where T_r is 3D rotation matrix and v_s represent shift. Then the relation between its perspective projection and its position in the world coordinate system is

$$v_i'' = T_i * (T_r * v_i + v_s) = T_i * T_r * v_i + v_f, \quad (3.5)$$

where $v_f = T_i * v_s$.

Now, given any three known points v_1 , v_2 and v_3 in world coordinates, their corresponding points v_1'' , v_2'' and v_3'' on image plane can be found by using equation (3.5). So, we have

$$v_j'' = T_i * T_r * v_j + v_f, \quad (3.6)$$

where $j=1,2,3$. By subtraction, we have

$$v_2'' - v_1'' = T_i * T_r * (v_2 - v_1), \quad (3.7)$$

and

$$v_3'' - v_1'' = T_i * T_r * (v_3 - v_1), \quad (3.8)$$

or,

$$(v_2'' - v_1'', v_3'' - v_1'') = T * (v_2 - v_1, v_3 - v_1)$$

where $T = T_i * T_r$ is a 3*3 matrix. Putting the above equations together in the form of x, y, and z components, we have

$$\begin{pmatrix} x_2'' - x_1'' & x_3'' - x_1'' \\ y_2'' - y_1'' & y_3'' - y_1'' \\ z_2'' - z_1'' & z_3'' - z_1'' \end{pmatrix} = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{pmatrix} * \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \\ z_2 - z_1 & z_3 - z_1 \end{pmatrix}. \quad (3.9)$$

Since all projected points have the same z value, $z_2'' - z_1'' = 0$ and $z_3'' - z_1'' = 0$.

Also assume these objects in the 3D world coordinate system have the same depth. Then, $z_2 - z_1 = 0$ and $z_3 - z_1 = 0$.

Let $v_1'' = 0$ and $v_1 = 0$. Finally, we have

$$T_1 = \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix}^{-1} = \begin{pmatrix} x_2 & x_3 \\ y_2 & y_3 \end{pmatrix} * \begin{pmatrix} x_2'' & x_3'' \\ y_2'' & y_3'' \end{pmatrix}^{-1} \quad (3.10)$$

Thus, for any point v_i'' in image plane, we can get the corresponding point v_i ,

by

$$v_i = T_1 * v_i''$$

3.3 Mathematical Algorithm For The Environment With Random Disturbances

In this section, we will describe the mathematical formulation for the calibration of the hand-eye system for an environment with random disturbances. For any change of camera position, there is a corresponding change in the image of the workarea. Depending on the position, the image may have increased or decreased in size. The position of the work piece in world coordinates would not change with a change in camera position. But there is a change in the value of the image coordinates and the coefficients of the inverse homogeneous transformation matrix.

Using the same notation for defining the coordinate systems as in Section 3.2.1, we have, for the change of camera position; the rotation matrix T_r changes to $T_r + \delta T_r$ and v_s changes to $v_s + \delta v_s$. The coordinate of the point in image plane is defined as

$$v_i'' = T_i * (T_r + \delta T_r) * v_i + T_i * (v_s + \delta v_s). \quad (3.11)$$

Three points in the world coordinates are v_1 , v_2 and v_3 , and their corresponding points on image are v_1'' , v_2'' and v_3'' . So, we have,

$$v_2'' - v_1'' = T_i * (T_r + \delta T_r) * (v_2 - v_1) \quad (3.12)$$

$$v_3'' - v_1'' = T_i * (T_r + \delta T_r) * (v_3 - v_1) \quad (3.13)$$

or,

$$(v_2'' - v_1'', v_3'' - v_1'') = T'' * (v_2 - v_1, v_3 - v_1) \quad (3.14)$$

where $T'' = T_i * (T_r + \delta T_r)$.

Using the distributive property of matrix multiplication,

$$T'' = T_i * T_r + T_i * \delta T_r = T + T'$$

where $T = T_i * T_r$ and $T' = T_i * \delta T_r$.

The coefficients of T_1 change to $T_1 + \delta T_1$. Also the image coordinates change to v_{i1}'' . So the position of the workpiece in world coordinates, when the camera is moved is given by

$$v_i = (T_1 + \delta T_1) * v_{i1}''. \quad (3.15)$$

Figure 3.1: Location determination of workpiece pose given the distances between feature points in 3D and the position of the points in the image plane

CHAPTER IV

FUZZY LOGIC ALGORITHM

It is difficult mathematically to establish a correlation between the change in camera position and change in the value of image coordinates. Also the calculation of the change in the value of the coefficients of the inverse homogeneous transformation matrix poses this difficulty. So the hand-eye system has to be calibrated with every change in camera position using conventional calibration. Thus it is important to make the system adaptive. We are using fuzzy logic for that purpose.

The hand-eye robotic system requires a fine tuning so that we can take care of small changes in the position of the camera. Knowing the image coordinates of the workpiece at two extreme positions of the camera, we should be able to determine the image coordinates of the workpiece at any location between these positions.

We calibrate the system at two extreme positions. These two positions are termed A and B. A is considered the reference point. So at A, there is a zero change in the value of the coefficients of T_1 . The maximum change in the value of the coefficients of T_1 occurs at B. These positions form the boundaries within which the camera is moved. By experimentation, the coefficients of significance are t_1 and t_4 . These two coefficients change when the camera position is changed. The changes in these two coefficients has to be determined.

We are using fuzzy logic to help fine tune the system. When the camera moves, there is a change in the size of the objects in the image. The squared distance between two known points on the image is used to define the change in the camera position in terms of fuzzy membership functions. With this information, the hand-eye system can be calibrated using fuzzy logic.

The design of the fuzzy logic algorithm consists of (1) determination and fuzzification of the input, (2) selection of fuzzy inference rules for reasoning, and (3) defuzzification and computation of outputs.

4.1 Fuzzification

Fuzzification is defined as a mapping from an observed input space to fuzzy sets in certain input universes of discourses. The fuzzification stage involves the following steps:

1. measuring the values of the input variables,
2. performing a scale mapping that transfers the range of values of input variables into corresponding universes of discourse,
3. performing the conversion of input data into suitable linguistic values.

The input d , is a squared distance between two known points on the image. The two known points are markers on the workarea. Using image processing, the (x,y)

value of the centroid of these markers in terms of pixels can be obtained. The squared distance can then be found out using the distance formula:

$$d = (x_2 - x_1)^2 + (y_2 - y_1)^2. \quad (4.1)$$

The value of d changes when the camera position changes. The values of d at positions A and B are measured beforehand by experimentation. The outputs are the changes in t_1 and t_4 , denoted as δt_1 and δt_4 respectively. Thus the system is a single-input-two-output system.

A fuzzification operator is used to convert the observed input data d into fuzzy data. A triangle is chosen to be the fuzzy membership function [Lee 1990]. The vertex of the triangle corresponds to the mean value of the data set. Using this membership function, fuzzification is done for the input and output variables.

The parameter d , is quantized into five levels. The linguistic variables used to describe each level are positive large L , positive (negative) medium $\pm M$, positive (negative) small $\pm S$. For the output parameters δt_1 and δt_4 , we quantize them into small S , medium M , and large L . Thus the fuzzy membership functions for each linguistic variable are defined. Figure 4.1 gives the membership functions for each parameter.

4.2 Fuzzy Control Rules

The next stage in the development of the fuzzy logic algorithm is the selection of fuzzy control rules. A fuzzy system is characterized by a set of linguistic

statements based on expert knowledge and operator's control actions. The expert knowledge is in the form of the "if-then" rules [Lee 1990]. Thus a fuzzy control rule is a fuzzy conditional statement in which the antecedent is a condition in its application domain and the consequent is a control action for the system under control. This collection of rules form the fuzzy control rule base.

The hand-eye system here is a single-input-two-output system. For example, fuzzy control rules have the form:

$$R_1: \text{if } x \text{ is } A_1 \text{ then } y \text{ is } B_1, z \text{ is } C_1,$$

... ..

$$R_n: \text{if } x \text{ is } A_n \text{ then } y \text{ is } B_n, z \text{ is } C_n,$$

where x , y and z are linguistic variables representing one process variable and two control variables. A_i , B_i , and C_i are linguistic are linguistic values of the linguistic variables x , y and z with $i = 1, 2, \dots, n$. For this system five rules have been developed. They are given in Figure 4.2.

4.3 Defuzzification

Defuzzification is the mapping from a space of fuzzy control actions defined over an output universe of discourse into a space of nonfuzzy control actions. This strategy is used in producing a nonfuzzy control action that best represents the possibility distribution of an inferred fuzzy control action.

The most widely used method is the Center of Area Method (COA) [Lee 1990]. This method generates the center of gravity of the possibility distribution of a control action. Thus in the case of a discrete universe, the nonfuzzy control action or crisp control action z_0 is given by

$$z_0 = \frac{\sum_{j=1}^n w_j \mu_z(w_j)}{\sum_{j=1}^n \mu_z(w_j)} \quad (4.2)$$

where n is the number of quantization levels of the input and w_j is the support value at which the membership function reaches the maximum value $\mu_z(w_j)$.

For the hand-eye system, the outputs are the δt_1 and δt_4 . The outputs are computed as linguistic variables for each set of input. Then defuzzification is done by the Center of Area Method as,

$$\delta t_1 = \frac{\sum_{j=1}^5 w_j \delta t_1}{\sum_{j=1}^5 w_j}, \quad (4.3)$$

$$\delta t_4 = \frac{\sum_{j=1}^5 w_j \delta t_4}{\sum_{j=1}^5 w_j}, \quad (4.4)$$

where w_j is the support value.

Figure 4.1: Fuzzy membership functions for various parameters

IF d IS NEGATIVE SMALL THEN δl_1 IS SMALL AND δl_4 IS SMALL.

IF d IS NEGATIVE MEDIUM THEN δl_1 IS MEDIUM AND δl_4 IS MEDIUM.

IF d IS LARGE THEN δl_1 IS MEDIUM AND δl_4 IS MEDIUM.

IF d IS POSITIVE MEDIUM THEN δl_1 IS MEDIUM AND δl_4 IS MEDIUM.

IF d IS POSITIVE SMALL THEN δl_1 IS LARGE AND δl_4 IS LARGE.

Figure 4.2: Fuzzy inference rules

CHAPTER V

EXPERIMENTS

This chapter explains the experiments conducted on the hand-eye system to test the conventional calibration algorithm for a stationary environment and the fuzzy logic algorithm for an environment with random disturbances.

In the first section, the hand-eye robotic manipulator system setup is described. In the second section, the experiments conducted using a stationary camera is described. In the third section, the experiments conducted for testing the fuzzy logic algorithm is described.

5.1 Experimental Setup

To test the algorithm, a hand-eye robotic manipulator system was used. The system consists of a single camera, a six degree-of-freedom robotic manipulator and a workpiece. The physical system setup is shown in Figure 5.1.

We use a Panasonic DIGITAL 5100 camera for taking the picture. The camera is mounted on a vertical stand in such a way that it is stationary with respect to the base coordinate frame of the robot. The camera can be moved up or down on the vertical stand. The base of the vertical stand forms the workarea for the experiments. The workpiece is placed in the workarea for conducting the experiments. The whole experiment is conducted under artificial lighting.

The picture of the workarea is captured in TGA file format by using a Targa Plus videographics adapter attached to the bus of a IBM compatible 486 personal computer. We use the TIPSPLUS software provided with the adapter for grabbing the image.

The TGA file is converted to an image file in IMG format. This conversion is done by a conversion routine in the PCDIP package. This file is the input to the program for the conventional calibration algorithm or to the fuzzy logic algorithm program.

A six degree-of-freedom SCORBOT-ER VII robot is used as a part of the hand-eye system. A controller controls the motion of the different axes of the robot. A command is sent to the controller which then makes the robot do the required action. Initially the commands were sent to the controller using the ATS software associated with the robot. The problem associated with this technique is that the position information has to be given within the ATS environment. So the calibration algorithm has to be run separately and the robot has to be moved independently. This slows down the computation. User developed programs written in C were used to integrate the image processing and robot motion computation, as described later in Section 5.2.1.

The operating range of the manipulator is shown in Figures 5.2-5.5. The optimal operating range in the X and Y coordinates is between 30 centimeters and 60 centimeters from the base of the robot. The optimum range of the pitch value

is from -85 degrees to +85 degrees. The robot controller, for every pitch angle, calculates its own work envelope. Any position that is not within the work envelope for that particular pitch value is determined as a bad position. The rule is that the angle between the shoulder and the elbow should never be greater than 180 degrees. This is the range within which the robot has the work envelope. Thus the angle between the shoulder and the elbow is less than 180 degrees. The relative distances between the camera, workarea and robot is shown in Figure 5.6.

An important consideration for the size of the workarea is the pitch angle of the robot. For every pitch angle of the manipulator, there is a corresponding work envelope which has to be determined by experimentation. We have determined the work envelope of the robot for a pitch angle of 69.2 degrees to ensure all the positions of the end effector be kept within the workarea. The boundaries of the work area within which the robot can reach a workpiece for the above pitch angle of is given as follows.

$$P1 : X = 800, Y = -6700, Z = 600, P = -692, R = -3$$

$$P2 : X = -1200, Y = -6700, Z = 600, P = -692, R = -3$$

$$P3 : X = -1200, Y = -5000, Z = 600, P = -692, R = -3$$

$$P4 : X = -3000, Y = -5000, Z = 600, P = -692, R = -3$$

$$P5 : X = -2800, Y = -3500, Z = 600, P = -692, R = -3$$

$$P6 : X = -1300, Y = -3500, Z = 600, P = -692, R = -3$$

$$P7 : X = 800, Y = -3500, Z = 600, P = -692, R = -3$$

$$P8 : X = 800, Y = -5000, Z = 600, P = -692, R = -3$$

where P_i , $i = 1, 2, \dots, 8$, is defined as the positions, the end effector could reach within the workarea, P is the pitch angle and R is roll angle of the wrist (see Figure 5.7).

5.2 Stationary Environment

Initially, we have kept the camera in a stationary environment. The homogeneous transformation algorithm is used to get the position information of the workpiece. This algorithm solves the inverse mapping problem, provided that the depth information is given. This section gives a brief description of the implementation and experiments conducted for the calibration of a hand-eye system for a stationary environment.

5.2.1 Implementation of the Algorithm

The implementation of the linear homogeneous transformation algorithm for calibration of a hand-eye system was done using Turbo C (Version 2.0) language. The IMG format file is given as input to the program. The program analyzes the image and extracts the position information of the workpiece, and this information is then used to calculate the motion trajectory for the robot to fetch the workpiece.

Initially, the robot was moved to the required position using its Advanced Terminal Software (ATS). Since the software is an independent environment, the position information from the calibration algorithm could not be passed on to the robot. This required manual typing of the position in the ATS environment.

An interface program is used which bypasses the ATS environment and directly communicates with the controller. This interface program is coded in Turbo C and Turbo assembly (Version 2.0). The position information from the calibration algorithm can now be sent directly to the controller for moving the robot to the position of the workpiece without any interruption.

5.2.2 Testing Algorithm Using Robot

The algorithm described in the previous section has been tested on the hand-eye system. Initially the robot and the camera coordinate system is calibrated so that the relative distances between the robot, the workpiece and the camera are measured. A workpiece is placed in the workarea within the robot's reachable work envelope (see Section 5.1).

The picture of the workpiece was taken using the Panasonic camera. The picture is displayed on a SONY monitor. This picture is grabbed in the form of a TGA file. This file is then converted into a IMG format file. This file is sent as input to the calibration program. The image has 64 gray levels.

The position of the workpiece is computed by the calibration algorithm. The robot is then moved to the required position. The experiments have been conducted for different positions of the workpiece on the workarea. These experiments help in determining the accuracy of the calibration algorithm

5.3 Environment with Small Random Disturbances

The second stage of our work is the implementation and testing of the fuzzy logic calibration algorithm for an environment with random disturbances. The main change in the environment is the change of the camera position.

Initially the calibration of the hand-eye system using the conventional algorithm is done at the two extreme points. These points form the boundary in which the fuzzy algorithm has been developed and implemented. The experiments conducted here are to test and verify the fuzzy logic algorithm.

5.3.1 Calibration at the Extreme Points

Two positions(A and B) are marked on the camera stand. The distance between the two positions is 10 inches. The camera is placed at the two positions and the hand-eye system is calibrated using the conventional technique. These two positions form the boundary within which the camera is moved. At positions A and B, the matrix associated with the positions is used to compute the position of the workpiece. In between A and B, there is a change in the value of the matrix and this change is determined using fuzzy logic. The position A is considered as a

reference point. The images, when the camera is at positions A and B, are shown in Figures 5.8 and Figure 5.9.

5.3.2 Implementation Of Fuzzy Logic Algorithm

When the camera position changes, there is a change in the size of the image of the workarea. This causes the position of the workpiece to change. The fuzzy logic algorithm is used to take care of those changes. In our hand-eye system, we assume that the camera is placed between positions A and B.

After the hand-eye system is conventionally calibrated at these positions, two markers of almost equal size are placed in the workarea. These markers are used to determine the change in the size of image. The squared distance d between the markers is the measure for such a change. The change in d is used to determine how far is it away from the reference point A. The unit of d is in square of pixels.

A "C" program is coded to implement the fuzzy logic algorithm. The image of the work area in IMG file format is the input to this program. The IMG file is in 64 gray levels. This is then binarized. The binarization is done in such a way that all the pixels above a certain threshold is given a value 1 and all other pixels is given a value 0.

A region labelling algorithm developed by J.C. Lin is used to label each region in the binarized image. This algorithm scans the image from top-left to bottom-right. If the pixel is zero, then the next pixel is scanned. Non-zero pixels

are assigned a label. The pixels, which are connected, are assigned the same label. Thus each individual binary region is given a different label (color). Preprocessing is performed to remove random noise. It is assumed that the size of the workpiece is greater than that of the markers. The centroid of the two smaller patterns is determined and the squared distance(d) in pixels is computed.

The fuzzy logic algorithm is then used to determine the position of the workpiece in robot coordinates. d is the input parameter to the fuzzy logic algorithm. The outputs are the changes in t_1 and t_4 , are denoted as, δt_1 and δt_4 , respectively (see Section 4.1).

The fuzzification routine fuzzifies the value of the input parameter d . d is quantized into the following fuzzy levels: large L, positive (negative) medium $\pm M$, positive (negative) small $\pm S$. For the output parameters δt_1 and δt_4 , we quantize them into small S, medium M, and large L.

Depending on the value of d , it is assigned the corresponding fuzzy value. The fuzzy inference routine goes through the fuzzy control rule base and determine the appropriate fuzzy values of the output.

The defuzzification routine defuzzifies the fuzzy values of output into non-fuzzy values using the center of gravity method. The outputs are the changes in t_1 and t_4 , δt_1 and δt_4 , respectively. From these values, the (x,y) position of the centroid of the workpiece in world coordinates is computed.

The values of the position of the workpiece are sent to the robot through the interface program SEND.C.

5.3.3 Testing The Fuzzy Logic Algorithm

The algorithm is tested using the hand-eye system. Initially, the camera is kept at position A. The image is taken and the robot is made to move to the position of the workpiece using the conventional calibration. This value of the position is taken as the reference value for the experimentation.

The distance between A and B is 10 inches. The camera is moved at about 0.75 inches at a time. The criterion for this movement is that there is a significant change in the image. Overall, there are fourteen positions for the camera in the experiments.

Initially, the robot was moved physically using the teaching pendant to the workpiece and the values of the position of the workpiece is obtained. Then the experiments were conducted using fuzzy logic.

Since the hand-eye system is already calibrated at positions A and B, we assume that the values of the positions of the workpiece computed at these positions have no error.

At each position, the picture of the workarea is taken. Using the fuzzy logic algorithm, the position of the object is calculated and the robot is moved to that

position. The experiments are conducted in such a way that the validity of the algorithm over the entire experimental range is tested.

5.3.4 Implementation And Testing Of Linear Interpolation

We are interested in doing a comparative study of the fuzzy logic algorithm with a linear interpolation algorithm. This study helps us in determining the error criterion for both the methods. If the values of the parameters are known at two extreme points, then the values in between the points could be obtained by linear interpolation.

The values of t_1 and t_4 for any value of d could be computed by using the formula,

$$t_1 = t_{1A} + \frac{(t_{1B} - t_{1A})}{(d_B - d_A)} * (d - d_A), \quad (5.1)$$

$$t_4 = t_{4A} + \frac{(t_{4B} - t_{4A})}{(d_B - d_A)} * (d - d_A), \quad (5.2)$$

where A and B are the two extreme points where the hand-eye robot system has been calibrated. Using these values of t_1 and t_4 , the position of the workpiece could be computed by Equation 3.15.

The linear interpolation algorithm is coded in a separate program. This algorithm has been tested with the same set of experimental conditions used for the fuzzy logic algorithm, so we could obtain fairly accurate results for our comparative study.

Figure 5.1: Physical setup of system

Figure 5.2: Robot Arm Axes

(Source: ESHED ROBOTEC SCORBOT-ER VII User's Manual)

Figure 5.3: Robot Arm Transmissions

(Source: ESHED ROBOTEC SCORBOT-ER VII User's Manual)

Figure 5.4: Operating range with gripper
(Source: ESHED ROBOTEC SCORBOT-ER VII User's Manual)

Figure 5.5: Operating range without gripper
(Source: ESHED ROBOTEC SCORBOT-ER VII User's Manual)

Figure 5.6: Relative position between camera and robot

Figure 5.7: Boundaries reached by the end effector in the workarea

Figure 5.8: Image for camera at position A which is an upper position with the camera conventionally calibrated.

Figure 5.9: Image for camera at position B which is the lower position with the camera conventionally calibrated

CHAPTER VI

RESULTS AND OBSERVATIONS

Experiments were conducted for a set of fourteen camera positions using the fuzzy logic algorithm and the experiments were repeated using the linear interpolation algorithm. In this chapter, we describe the results and observations found during the testing of the fuzzy logic algorithm and the linear interpolation algorithm in separate sections. The results of the experiments conducted and their observations is described. Finally we give a comparative study of the results.

6.1 Fuzzy Logic Algorithm

The fuzzy logic algorithm was tested for the hand-eye robot system. At each position of the camera, the values of the coordinates of the workpiece is recorded. These values are compared with the reference values determined by moving the robot to the workpiece at position A.

At positions A and B, the system is conventionally calibrated and hence it is assumed that there is no error. The values of the coordinate of the workpiece at the reference point is (x_0, y_0) . The error criterion E for the position of the workpiece is given as the sum of the square of the error in the X coordinate and the sum of the square of error in the Y coordinate. This criterion gives a good

picture of the total error incurred. We could judge by this criterion, the error occurring in both X and Y coordinates. E is computed by

$$E = \delta_x^2 + \delta_y^2, \quad (6.1)$$

where $\delta_x = x - x_0$ is the error in the X extent and $\delta_y = y - y_0$ is the error in the Y extent of the workpiece.

Table 6.1 explains the results of the experiments in detail. For each position of the camera, d and the coordinates (x, y) of the workpiece obtained using the fuzzy logic algorithm is given. Also the error in X and Y coordinates are shown. The final column shows the value of E for each position of the camera. The graphical distribution of the error is shown in Figure 6.1.

From the results, it has been observed that the maximum error occurred around the region where the fuzzy value of d lies in two levels. In spite of this error, the robot was able to move to the position of the workpiece.

6.2 Linear Interpolation Algorithm

The linear interpolation algorithm was tested for the same set of experiments as that of the fuzzy logic algorithm. The error criterion E is the same as that defined for the fuzzy logic algorithm.

Table 6.2 explains the results of the experiments in detail. This table contains d , the coordinates (x, y) of the workpiece, errors in X and Y coordinates, and E. The graphical distribution of E with respect to d is given in Figure 6.2.

6.3 Comparative Study

A comparative study of the fuzzy logic algorithm and linear interpolation was conducted. A graphical representation of E for both fuzzy logic and linear interpolation is given in Figure 6.3. This figure gives a good representation of E in both cases.

It is found that the linear interpolation algorithm is generally more erroneous than that of the fuzzy logic algorithm. This may be due to the fact that for a linear change in the position of the camera, the change in the distance between the markers d is nonlinear.

Another criterion used for comparison is the cumulative error E_c . E_c is defined as the total error that has occurred over the whole set of experiments. It can be defined mathematically as

$$E_c = \sum_{j=1}^n E_j \quad (6.2)$$

where E_j is the error occurring in the X and Y coordinates for any j position (Equation 6.1) and n is the number of positions where the camera was placed.

E_c for the fuzzy logic algorithm was found to be 267.94 and for linear interpolation algorithm was found to be 632.63. From these values, it can be inferred that the fuzzy logic algorithm gives a smaller cumulative error than that of the linear interpolation algorithm. Thus fuzzy logic algorithm is more accurate.

It was also found that the error given by the fuzzy logic algorithm lies within the tolerable range for the robot to go and pick the workpiece. In the case of linear interpolation algorithm, at certain places, the error was more than the tolerable range of the robot and hence it could not pick the workpiece.

Table 6.1: Results of experimentation with fuzzy logic algorithm

Position	Squared Distance d	X	Y	δ_x	δ_y	E
A	40500	-53.7	-589.0	0.0	0.0	0.00
1	42800	-49.5	-587.0	-4.2	-2.0	21.64
2	44800	-50.0	-590.0	-3.7	+1.0	14.69
3	47000	-48.0	-592.0	-5.7	+3.0	41.49
4	49000	-51.7	-587.5	-2.0	-1.5	6.25
5	50100	-53.9	-535.0	-0.1	-4.0	16.01
6	51400	-58.0	-583.0	+4.3	-6.0	54.49
7	52700	-54.3	-585.9	+0.6	-3.1	9.97
8	54000	-55.0	-582.8	+1.3	-6.2	40.13
9	55300	-52.6	-585.2	-0.7	-3.8	14.93
10	56800	-54.8	-584.5	+1.1	-4.5	21.46
11	58280	-52.1	-586.4	-1.6	-2.6	9.32
12	60600	-53.9	-586.0	+0.2	-3.0	9.04
13	61800	-53.5	-587.0	-0.2	-2.0	4.04
14	63800	-58.0	-585.0	+4.3	-4.0	34.49
B	66790	-53.7	-589.0	+0.0	+0.0	0.00

Value of X and Y is in millimeters.

value of d is in square of pixels.

Cumulative Error for Fuzzy Logic $E_c = 267.94$

Table 6.2: Results of experimentation with linear interpolation

Position	Squared Distance d	X	Y	δ_x	δ_y	E
A	40500	-53.7	-589.0	0.0	0.0	0.00
1	42800	-50.0	-587.0	-3.7	-2.0	17.69
2	44800	-52.0	-588.0	-1.7	-1.0	3.09
3	47000	-56.4	-587.0	+2.7	-2.0	11.29
4	49000	-57.7	-583.0	+4.0	-6.0	52.0
5	50100	-56.2	-534.0	+2.5	-5.0	31.25
6	51400	-61.0	-583.0	+7.3	-6.0	89.29
7	52700	-59.8	-584.0	+6.1	-5.0	62.21
8	54000	-61.0	-583.2	+7.3	-5.8	86.93
9	55300	-58.5	-583.0	+4.8	-6.0	59.20
10	56800	-63.1	-584.0	+9.4	-5.0	113.36
11	58280	-59.5	-585.0	-5.8	-4.0	49.64
12	60600	-59.6	-587.0	+5.9	-2.0	38.81
13	61800	-55.1	-586.0	+1.4	-3.0	10.96
14	63800	-56.7	-588.0	+3.0	-1.0	10.0
B	66790	-53.7	-589.0	+0.0	+0.0	0.00

Value of X and Y is in millimeters.

value of d is in square of pixels.

Cumulative Error for Linear Interpolation $E_c = 632.63$

Figure 6.1: Error in fuzzy logic algorithm

Figure 6.2: Error in linear interpolation algorithm

Figure 6.3: Comparison of fuzzy and linear interpolation algorithms

CHAPTER VII

CONCLUSION

In this thesis, we have analyzed the effect of the change in the camera position and its impact on the calibration of the hand-eye system. This causes a recalibration of the system which involves a lot of time.

We have described a fuzzy logic algorithm for taking care of changes in the position of a camera without the need of actual calibration every time the camera is moved. The fuzzy logic algorithm is able to take care of changes in the position of the camera within certain limits. This algorithm saves time since there is no need of calibrating the hand-eye system every time the camera is moved. This algorithm could be used in places where the camera is utilized for different purposes and need not worry about recalibrating the hand-eye system.

We have also successfully integrated the image processing and the robot systems into one single system which makes the whole system faster.

As future research, we will highlight the enhancements that could be made to the system. The system could be enhanced to handle more than one workpiece at a time by incorporating more sophisticated computer vision algorithms. Now the system is only handling the vertical change in the position of the camera. The next step would be to modify the system to take care of changes in the other two directions in 3D space. Another enhancement of the system could be the integration of the picture capturing system and the fuzzy calibration system.

REFERENCES

- [1] John T. Feddema, George Lee, Robert Mitchell, Model-Based Visual Feedback Control for a Hand-Eye Coordinated Robotic System, *IEEE Transactions on Computers*, page 21-29, Aug 1992.
- [2] M.A. Fischer and R.C. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, *Comm. ACM*, vol. 24, no. 6, June 1981, pp. 381-395.
- [3] R. Y. Tsai, An efficient and accurate camera calibration technique for 3D machine vision, *Proc. IEEE Computer Vision and Pattern Recognition*, 1986, pp. 364-374.
- [4] A. Izaguirre, P. Pu, and J. Summers, A new development in camera calibration: Calibrating a pair of mobile cameras, *Int. J. Robotics Res.*, vol 6, no.3, pp. 104-116, 1987.
- [5] R. K. Lenz and R. Y. Tsai, Techniques for calibration of the scale factor and image center for high accuracy 3-D machine vision metrology, *Proc. IEEE Int. Conf. Robotics and Automation*, 1987, pp. 68-75.
- [6] R. Y. Tsai and R. K. Lenz, A new technique for fully autonomous and efficient 3D robotics hand/eye calibration, *Proc. 4th Int. Symp. Robotics Research*, Santa Cruz, CA, Aug. 1987.
- [7] O. D. Faugeras and G. Toscani, Camera Calibration for 3-D computer vision, *Proc. Int. Workshop Industrial Applications of Machine Vision and Machine Intelligence*, Tokyo, Japan, Feb. 1987, pp. 240-247.
- [8] P. Liang, Y. L. Chang, and S. Hackwood, Adaptive self-calibration of vision-based robot systems, *IEEE Trans. on Systems, Man, and Cybernetics*, vol 19, no. 4, July 1989, pp. 811-824.
- [9] P. Liang, J. F. Lee, and S. Hackwood, A general framework for robot hand-eye coordination, *Proc. IEEE Int. Conf. on Robotics and automation*, 1988.

- [10] S. Arimoto, S. Kawamura, and F. Miyazaki, Can mechanical robots learn by themselves?, *Robotics Research: 2nd Int. Symp.* Cambridge, MA: MIT Press, 1985, pp. 127-134.
- [11] W. T. Miller III, Sensor-based control of robotic manipulators using general learning algorithm, *IEEE J. Robotics Automation*, vol. RA-3, pp. 157-165, Apr. 1987.
- [12] S. B. Skaar, W. H. Brockman, and R. Hanson, Camera-space calibration, *Int. J. Robotics Res.*, vol 6, no. 4, pp. 20-32, 1987.
- [13] R. A. Brooks, A. M. Flynn, and T. Marill, Self calibration of motion and stereo vision for mobile robots, *Proc. 4th Int. Symp. Robotics Research*, Santa Cruz, CA, Aug. 1987.
- [14] A.C. Sanderson and L. E. Weiss, Image-based servo control of Robots, *Proc. 26th Ann. SPIE Technical Symp.*, Aug. 1987.
- [15] M. Kuperstein, Adaptive visual-motor coordination in multijoint robots using parallel architecture, *Proc. 1987 IEEE Int. Conf. Robotics and Automation*, Raleigh, NC, 1987, pp. 1595-1602.
- [16] J. A. Walter, K. J. Schulten, Implementation of Self-Organizing Neural Networks for Visuo-Motor Control of an Industrial Robot, *IEEE transactions on neural networks*, Jan 01 1993 vol 4 no 1 page 86.
- [17] T. M. Martinez, H. J. Ritter, K. J. Schulten, Three-dimensional Neural Net for Learning Visuomotor Coordination of a Robot Arm, *IEEE transactions on neural networks*, Mar 01 1990, vol 1, no 1, Page 131.
- [18] C. C. Lee, Fuzzy Logic in Control Systems: Fuzzy Logic Controller-Part I, *IEEE Transactions on Systems, Man, Cybernetics*, vol 20, no 2, March/April 1990, pp 404-418.
- [19] C. C. Lee, Fuzzy Logic in Control Systems: Fuzzy Logic Controller -Part II, *IEEE Transactions on Systems, Man, Cybernetics*, vol 20 , no 2, March/April 1990, pp 419-434.
- [20] T. Poggio and C. Koch, Ill-posed Problems in early vision: From computational theory to analog networks, *Proc. of Royal Society, London* B-226, pp 303-323, 1985.

- [21] H. Li and S. S. Kumaresan, Feasibility Study of Fuzzy Logic in calibration of Hand-eye Systems, *17th IEEE International Conference on Semiconductor Manufacturing Technology*, to be presented in Sept 1994.

APPENDIX

SOURCE CODE

```
/******  
**  
** COPYRIGHT NOTICE : COPYRIGHT © 1994. *  
** THIS CODE IS PROPERTY OF Dept. Of Comp.Sc. *  
** *  
** *  
** TITLE : IMPPLEMENTATION OF FUZZY LOGIC ALGORITHM FOR *  
** HAND-EYE SYSTEM. *  
** AUTHOR : SUKIR S. KUMARESAN *  
** DATE-WRITTEN : 04/01/94. *  
** VERSION : 4.0 *  
** SYSTEM REQ : 386 PC RUNNING MS-DOS, TURBO-C version 3.0 *  
** or higher. *  
** *  
** COMMENTS : This is the main module for the calibration *  
** of the hand-eye system using fuzzy logic. *  
*****/
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>
```

```
/******GLOBAL VARIABLES*****  
d is an unsigned long integer where the values of the distance  
between the markers is stored. (xi,yi) gives the value of the  
position of the workpiece.  
*****/
```

```
unsigned long d;  
int xi,yi;
```

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

This is the main program for the calibration of the hand-eye system using fuzzy logic. This module calls two other functions in different modules. pos() determines the position of the workpiece in the image. This values are then used by the fuzzy() to determine the world coordinates of the workpiece using fuzzy logic.

*****/

```
main()
{
    pos();
    fuzzy();
    sleep(5);
    system("send move hom"); /* Move the robot to the home position*/
}
```

```

/*****
**
** COPYRIGHT NOTICE : COPYRIGHT © 1994.
**
**          THIS CODE IS PROPERTY OF Dept. Of Comp.Sc.
**
**
** TITLE      :    POS.C      *
** AUTHOR     :    SUKIR S. KUMARESAN
** DATE-WRITTEN :    04/01/94.
** VERSION    :    4.0
** SYSTEM REQ  :    386 PC RUNNING MS-DOS, TURBO-C version 3.0
**
**          or higher.
**
** This is the module for the computing of the image coordinates
** for the position of the workpiece from the image. This module
** gets the image file as input. The image is binarized. If the
** pixel is above a threshold value, it is displayed. The
** binarized image is then labelled using the region labelling
** algorithm developed by J.C. Lin. This algorithm labels the
** different patterns in the binary image. The markers and the
** workpiece are identified. The distance between the markers is
** calculated and so is the centroid of the workpiece.
*****/

```

```

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <math.h>
#include "pcdip.h"
#include "dimple.h"
#define false 0
#define true 1

```

```

/*****GLOBAL DECLARATIONS*****/
indata is a pointer pointing to the two dimensional buffer which
stores the binarized image for labelling the different patterns
in the object. This is dynamically allocated since a 256x200
size image is used. The extern unsigned long d is the value for

```

the distance between the markers. This is computed in this module and is used by the module FUZZY.C. (xi,yi) gives the position of the workpiece in image coordinates. This value is sent to the module FUZZY.C to compute the world coordinates of the position of the centroid of the workpiece using fuzzy logic.

```
u_char **indata;
extern unsigned long d;
extern int xi,yi;
```

*****FUNCTION PROTOTYPES*****

Label() is the function to label different patterns in the image using a region labelling algorithm developed by J.C. Lin

```
void Label();
```

```
pos()
```

```
{
    int      row, column, light;          /* Loop variables */
    int      row1,column1;                /* Original row and col */
    int      srow,scolumn;               /* The saved row, col */
    int      key,thres=55,i,val;
    int      colct=0,col_1,col_2,col_3;
    unsigned long  count1x=0,count2x=0,count3x=0;
    unsigned long  count1y=0,count2y=0,count3y=0;
    unsigned long  area1=0,area2=0,area3=0;
    int      liroco1=0,licol1=0,lirow1=0,lifin;
```

```
    /* For storing the values of the position of markers
and workpiece*/
```

```
float  x1bar,y1bar,x2bar,y2bar,x3bar,y3bar;
```

```

u_char  buffer[256],buffer1[256];

int  save_row[7];
int  save_col[7];

char    in_name[20];
FILE    *infile;

/*----- get the image file name -----*/

strncpy(buffer1,"0",256); /* Initialising the buffer */
printf("\n Enter your 256x256-image file name please \n");
scanf("%s", in_name);
if ((infile = fopen(in_name,"rb")) == NULL)
{
printf("Error: can not open input file");
exit(1);
}

/***** Initialize the system *****/
The system is initialized. The buffer for the regional labelling
algorithm is dynamically allocated. The PC is set to graphics
mode for display of 64 gray level IMG file
*****/

indata = (u_char **) malloc(200 * sizeof(u_char *));
for (key=0;key <= 200;key++) {
    indata[key] = (u_char *) malloc(256 * sizeof(u_char ));
}
pc_attach(); /* set graphic mode (PC_DIP) */
pc_gray_shades(); /* set 64 gray level mode (PC_DIP) */

/*****

```


The image data is read from the input file in .IMG format. This is converted into a 64 gray level PC_DIP format. This image is then binarized. If the intensity of the pixel is above that of the threshold value, it is given a value 1. Otherwise it is given a value zero. The value of threshold is 55.

*****/

```
for(row=0;row<=200;row++){
    fread(buffer,1,256,infile);
    for(column=0;column<=255;column++) {
```

/*convert to 64-bit grey level*/

```
light = buffer[column] / 4;
pc_write_pixel(row,column,light); /* write a pixel (PC_DIP) */
if (column != 0) licol1 = buffer[column - 1] /4;
else licol1 =0;
if(row != 0)
{
    lirow1 = buffer1[column] /4;
    liroco1 = buffer1[column -1] /4;
}
else
    { lirow1 = 0;
      liroco1 = 0;
    }

    lifin = (liroco1 +licol1 + lirow1 +light) / 4;
```

If the value of pixel is greater than the threshold value assign value of 1 to the corresponding pixel in the buffer. Otherwise assign a value of zero.

*****/

```
if( lifin > thres)
{
    lifin = 63;
    indata[row][column] = 1;
}
```

```

else
{
light = 0;
indata[row][column] = 0;
}
}
for (i=0;i<=255;i++)
{
buffer1[i] = buffer[i];
}
}
fclose(infile);

```

```

/*****
The different patterns in the image is labelled using the
regional labelling algorithm. The patterns are assigned a
different color by the algorithm. There are three patterns
in the image, two markers and one workpiece.
*****/

```

```

Label();

```

```

/*****
Here we calculate the centroid position of the different
patterns in the labelled image. The position of the centroid
of the patterns is computed by going through the whole image
adding the number of pixels having the same label. Then the
X and Y positions in terms of pixels is obtained by

X = Sum of row number of pixels having same color
-----
    number of pixels having same color

Y = Sum of column number of pixels having same color
-----
    number of pixels having same color
*****/

```

```

for(row=0;row<=199;row++){
    for(column=0;column<=255;column++) {
        if ((val = indata[row][column]) > 0)
        { if (colct == 0)
        { col_1 = val;
          colct++;
          }

          else
          { if ((colct == 1) && (val != col_1))
          { col_2 = val;
            colct++;
            }

            else
            { if ((colct == 2) && (val != col_2))
            { col_3 = val;
              colct++;
              }
            }
          }
        }

        if (val == col_1)
        { count1x += row;
          count1y += column;
          area1++;}
        else {
            if (val == col_2)
            { count2x += row;
              count2y += column;
              area2++;}
            else {
                if (val == col_3)
                { count3x += row;
                  count3y += column;
                  area3++;}
                }
            }
        }
    }
}

```

```
/* Computing of Centroid for all the binary patterns
   in the image */
```

```
x1bar = count1x / area1;
y1bar = count1y /area1;
x2bar = count2x / area2;
y2bar = count2y /area2;
x3bar = count3x / area3;
y3bar = count3y /area3;
```

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

We have assumed that the size of the object is greater than that of the markers. Therefore we check the size of patterns and the largest pattern is designated the workpiece. The value of the centroid of the workpiece is send to the fuzzy logic algorithm. The other two patterns are the markers. The distance between them is computed by the distance formula.

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

```
if(area1 < area3 && area2 < area3)
{ xi=x3bar; yi= y3bar;
  d = (x1bar- x2bar) * (x1bar - x2bar) + (y1bar - y2bar)
    * (y1bar -y2bar);}
if(area1 < area2 && area2 > area3)
{ xi=x2bar; yi= y2bar;
  d = (x1bar- x3bar) * (x1bar - x3bar) + (y1bar - y3bar)
    * (y1bar -y3bar);}
if(area1 > area3 && area2 < area1)
{ xi=x1bar; yi= y1bar;
  d = (x2bar- x3bar) * (x2bar - x3bar) + (y2bar - y3bar)
    * (y2bar -y3bar);}
```

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

The image is displayed on the monitor. The centroid of the workpiece is shown by a cursor. The following is the code for the display of the cursor.

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



```

**
** This function takes the binary data in a buffer as input
** The image is scanned from top left to bottom-right. If the pixel
** is zero, then scan the next pixel. If the pixels are connected
** they are assigned the same label. Otherwise a new label is
** assigned
**
*****/

```

```

void Label()
{
int    i, k, j, l, ix, iy, icolor;
int    label, match[200];
int    new, A[4], scale;
int    count, flag, flag1, order[200];

new=0;
/* Start of phase 1 */
for (i=1; i<=199; i++) {
    for (j=1; j<=255; j++) {
        if (indata[i][j] != 0) {
A[0]=indata[i-1][j-1];
A[1]=indata[i-1][j];
A[2]=indata[i-1][j+1];
A[3]=indata[i][j-1];
        if ((A[0]+A[1]+A[2]+A[3]) == 0) {
new++;
indata[i][j]=new;
label=new;
match[new]=new;
        }
        else {
flag1=0;
for (k=0; k<3; k++) {
            if (A[k] != 0 && flag1 == 0) {
label=A[k];
            for (l=k+1; l<=3; l++) {
                if (A[l] != label && A[l] != 0) {
flag1=1;
                if (A[l] < label) {
                    match[label]=A[l];
                    label=A[l];
                }
            }
        }
    }
}
}
}
}
}
}

```

```

}
    if (A[l]>label) match[A[l]]=label;
}
}
}
}
indata[i][j]=label;
}
}
}
}

        /* End of phase 1 */

/*----- Start of phase 2 -----*/

for(k=new;k>=1;k--)                                /* Forward merging */
{
    if(k != match[k]) {
        for(i=0;i<=199;i++) {
            for(j=0;j<=255;j++) {
                if(indata[i][j]==k)
                    indata[i][j]=match[k];
            }
        }
    }
}
for(k=1;k<=new;k++)                                /* Backward merging */
{
    if(k != match[k]) {
        for(i=0;i<=199;i++) {
            for(j=0;j<=255;j++) {
                if(indata[i][j]==k)
                    indata[i][j]=match[k];
            }
        }
    }
}

        /* End of phase 2 */

```

```

/*****
**
** COPYRIGHT NOTICE : COPYRIGHT © 1994.
**                THIS CODE IS PROPERTY OF Dept. Of Comp.Sc.
**
**
** TITLE          :    FUZZY.C    *
** AUTHOR         :    SUKIR S. KUMARESAN
** DATE-WRITTEN   :    04/01/94.
** VERSION        :    4.0
** SYSTEM REQ     :    386 PC RUNNING MS-DOS, TURBO-C version 3.0
**                or higher.
**
** This is the module for the computation of the position of the
** workpiece in world coordinates using fuzzy logic algorithm and
** moving the robot to that position. The input to the program
** is the distance between the markera and the position of the
** centroid of the workpiece in image coordinates. The distance
** between the markers d is the input to the fuzzy logic algorithm*
** This value is fuzzified. The fuzzified value of d is then
** inferred over the fuzzy control rule base. The output of the
** fuzzy control algorithm is the change in the value of the
** coefficients of the linear homogenous transformation matrix.
** The values of the position of the workpiece is thus calculated *
** and the robot is moved to that position
*****/

```

```

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

```

```

/*****Global Variables *****/
t1,t2,t3,t4 are the coefficients of the linear homogenous
transformation matrix.
*****/

```

```

float t1=0,t2=0.000, t3=-.000,t4 = 0;

```



```

float mu_NS=0,mu_NML=0,mu_NMR=0,mu_LL=0,mu_LR=0;
float mu_PML=0,mu_PMR=0,mu_PS=0;
float  delt1_NS,delt1_NM,delt1_L,delt1_PM,delt1_PS,delt1;
float med_t1 = 0.475, end_t1 = 0.95;
float  delt4_NS,delt4_NM,delt4_L,delt4_PM,delt4_PS,delt4;
float med_t4 = 0.54, end_t4 = 1.08;
extern unsigned long d;
extern int xi,yi;
int xs=0,ys=0,xso=0,yso=0;
float x0,y0,f1,f2;

```

```

/*****Function Prototypes*****/

```

```

void get_position( int,int);
void fuzzify(unsigned long);
void neg_small(float);
void neg_med(float);
void large(float);
void pos_med(float);
void pos_small(float);
float cal_delt_small();
float cal_delt_medl(float );
float cal_delt_medr(float );
float cal_delt_large();
float cal_delt4_small();
float cal_delt4_medl(float );
float cal_delt4_medr(float );
float cal_delt4_large();
void defuzzify();
void calc_xy(float);

```

```

fuzzy()
{ char ch,bufx[200],bufy[200],bufz[100];
  char  buf1[100],buf2[100],buf3[100],buf4[100];
  int z;
  float xt,yt;

```

```

/*****
The input parameter d to the fuzzy logic algorithm is
quantized into a fuzzy level. The corresponding fuzzified
output parameters is obtained by searching the fuzzy control
rules. The fuzzified output parameters is then converted
back to a nonfuzzified output using the defuzzify(). Then the
value of the position of workpiece is obtained using the
calc_xy().
*****/

fuzzify(d);
defuzzify();
calc_xy(d);

/*****
The values of the position of the workpiece in image
coordinates is initialised. The position of the workarea
with respect to the robot coordinate system is initialised.
*****/

xi=xi-xs;yi=yi-ys;
get_position(xi,yi);
xt= -7000 + x0 * 300;
yt= 900 - y0 * 300;

/* Calculation of coordinates of points with respect to
workpiece */

printf("x0=%f y0=%f\n",x0-xso,y0-yso);
printf("\n Y coord = %f",xt);
printf("\n X coord = %f",yt);

/*****
The following section of the code involves the movement of
the robot to the position of the workpiece. The position
information is sent to the robot using the SEND program
which is used to send messages to the controller of the

```

robot

*****/

```
printf("\n Enter the position for the movement of the robot:");
scanf("%d",&z);
sprintf(bufx,"send %d ",(int ) yt);
sprintf(bufy,"send %d ",(int ) xt);
sprintf(bufz,"send defp %d",z);
sprintf(buf1,"send teach %d",z);
sprintf(buf2,"send listpv %d",z);
sprintf(buf3,"send move %d",z);
sprintf(buf4,"send undef %d",z);
system("send speed 10");
system(bufz);
system(buf1);
system(bufx);
system(bufy);
system("send 720");
system("send -692");
system("send -3");
system(buf2);
system(buf3);
system(buf4);
```

}

/*
 This function is to get the position of the X and Y coordinates
 of the position in world coordinates from image coordinates
 */

void get_position(int xi,int yi)

```
{
    f1=t1*(float)xi+t2*(float)yi;
    f2=t3*(float)xi+t4*(float)yi;
    x0=f1;
    y0=f2;
```

```
}
```

```

/*****
The folloing function fuzzifies the input parameter d into any
one of the following five levels: large L, negative small NS,
negative medium NM, positive small PS, positive medium PM.
Depending on the fuzzified value of d, the corresponding fuzzy
control rule is applied and the fuzzified values of the output
parameters, deltat1 and deltat4 are obtained.
*****/

```

```

void fuzzify(unsigned long dum)
{
    if ((dum >= 40456) && (dum <= 47041))
        neg_small(dum);
    if ((dum >= 40456) && (dum <= 53625))
        neg_med(dum);
    if ((dum >= 47041) && (dum <= 60210))
        large(dum);
    if ((dum >= 53625) && (dum <= 66794))
        pos_med(dum);
    if ((dum >= 60210))
        pos_small(dum);
}

```

```

/*****
This function computes the value of output deltat1 and deltat4
if the fuzzified value of input d is negative small NS.
*****/

```

```

void neg_small(float dum)
{
    mu_NS = 1-((dum - 40456) /6584);
    delt1_NS = cal_delt_small();
    delt4_NS = cal_delt4_small();
}

```

```

/*****
This function computes the value of output deltat1 and deltat4
if the fuzzified value of input d is negative medium NM.
*****/

```

```

void neg_med(float dum)
{
    if (dum <= 47041)
    {
        mu_NML = (dum - 40456) / 6584;
        delt1_NM = cal_delt_medl(mu_NML);
        delt4_NM = cal_delt4_medl(mu_NML);
    }
    else
    {
        mu_NMR = 1 - ((dum - 47041)/6584);
        delt1_NM = cal_delt_medr(mu_NMR);
        delt4_NM = cal_delt4_medr(mu_NMR);
    }
}

```

```

/*****
This function computes the value of output deltat1 and deltat4
if the fuzzified value of input d is large L.
*****/

```

```

void large(float dum)
{
    if (dum <= 53625)
    {
        mu_LL = (dum - 47041) / 6584;
        delt1_L = cal_delt_medl(mu_LL);
        delt4_L = cal_delt4_medl(mu_LL);
    }
    else
    {

```

```

    mu_LR = 1 - ((dum - 53625)/6584);
    delt1_L = cal_delt_medr(mu_LR);
    delt4_L = cal_delt4_medr(mu_LR);
}
}

```

```

/*****
This function computes the value of output deltat1 and deltat4
if the fuzzified value of input d is positive medium PM.
*****/

```

```

void pos_med(float dum)
{
    if (dum <= 60210)
    {
        mu_PML = (dum - 53625) / 6584;
        delt1_PM = cal_delt_medl(mu_PML);
        delt4_PM = cal_delt4_medl(mu_PML);
    }
    else
    {
        mu_PMR = 1 - ((dum - 60210)/6584);
        delt1_PM = cal_delt_medr(mu_PMR);
        delt4_PM = cal_delt4_medr(mu_PMR);
    }
}

```

```

/*****
This function computes the value of output deltat1 and deltat4
if the fuzzified value of input d is positive small PS.
*****/

```

```

void pos_small(float dum)
{
    mu_PS = (dum - 60210) / 6584;

```

```

delt1_PS = cal_delt_large();
delt4_PS = cal_delt4_large();
}

```

```

float cal_delt_small()
{
    return(mu_NS * (1 - mu_NS) * med_t1);
}

```

```

float cal_delt_medl(float dummy)
{
    return(dummy * dummy * med_t1);
}

```

```

float cal_delt_medr(float dummy)
{
    return(dummy * (med_t1 + (1-dummy) * med_t1));
}

```

```

float cal_delt_large()
{
    return(mu_PS * (med_t1 + mu_PS * med_t1));
}

```

```

float cal_delt4_small()
{
    return(mu_NS * (1 - mu_NS) * med_t4);
}

```

```

float cal_delt4_medl(float dummy)
{
    return(dummy * dummy * med_t4);
}

```

```
float cal_delt4_medr(float dummy)
{
    return(dummy * (med_t1 + (1-dummy) * med_t4));
}
```

```
float cal_delt4_large()
{
    return(mu_PS * (med_t4 + mu_PS * med_t4));
}
```

```

/*****
  This function defuzzifies the fuzzy outputs of delta_t1 and
  delta_t4. The defuzzification is done by the center of area(COA)
  method. The values of coefficients t1 and t4 is then computed.
  *****/
```

```
void defuzzify()
{
    delt1 = delt1_NS +delt1_NM +delt1_L +delt1_PM +delt1_PS;
    printf("\n Value of delta1 = %f", delt1/100);
    delt4 = delt4_NS +delt4_NM +delt4_L +delt4_PM +delt4_PS;
    printf("\n Value of delta4 = %f", delt4/100);
    t1 = (.0446 - delt1/100);
    t4 = .0479 - delt4/100;
    printf("\n Value of t1 = %f", (.0446- delt1/100));
    printf("\n Value of t4 = %f",(.0479 - delt4/100));
}
```

```

/*****
  This function is used to initialize the coordinate system, every
  time the camera is changed.
  *****/
```



```
void calc_xy(float dum)
{
    if (dum >= 40494 && dum < 43070)
        { xs = 9; ys = 0;}
    if (dum >= 43070 && dum < 45700)
        { xs = 7; ys = -4;}
    if (dum >= 45700 && dum < 48300)
        { xs = 4; ys = -9;}
    if (dum >= 48300 && dum < 50950)
        { xs = -2; ys = -13;}
    if (dum >= 50950 && dum < 53570)
        { xs = -3; ys = -15;}
    if (dum >= 53570 && dum < 56210)
        { xs = -4; ys = -15;}
    if (dum >= 56210 && dum < 58840)
        { xs = -5; ys = -15;}
    if (dum >= 58840 && dum < 61470)
        { xs = -5; ys = -15;}
    if (dum >= 61470 && dum < 64100)
        { xs = -6; ys = -15;}
    if (dum >= 64100 && dum < 66800)
        { xs = -6; ys = -20;}
}
```

```

/*****
**
** COPYRIGHT NOTICE : COPYRIGHT © 1994.
**                      THIS CODE IS PROPERTY OF Dept. Of Comp.Sc.
**
**
** TITLE      :    LINEAR.C
** AUTHOR     :    SUKIR S. KUMARESAN
** DATE-WRITTEN :    04/01/94.
** VERSION    :    4.0
** SYSTEM REQ  :    386 PC RUNNING MS-DOS, TURBO-C version 3.0
**                      or higher.
**
** This is the module for the computation of the position of the
** workpiece in world coordinates using Linear interpolation and
** moving the robot to that position. The input to the program
** is the distance between the markera and the position of the
** centroid of the workpiece in image coordinates. The distance
** between the markers d is the input to the fuzzy logic algorithm*
** The system is calibrated at the two extreme positions. At these*
** positions the distance of the markers is measured and also the *
** coefficients of transformation matrix t1 and t4 are computed. *
** The values of the coefficients in between the positions is *
** computed by using the the linear interpolation formula. *
** The values of the position of the workpiece is thus calculated *
** and the robot is moved to that position *
*****/

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

/*****Global Variables *****/
t1,t2,t3,t4 are the coefficients of the linear homogenous
transformation matrix.
*****/

float t1=0,t2=0.000, t3=.000,t4 = 0;

```

```
int xs=0,ys=0,xso=0,yso=0;
float x0,y0,f1,f2;
```

```
/******Function Prototypes******/
```

```
void get_position( int,int);
void calct(float);
void calc_xy(float);
```

```
void main()
{ char ch,bufx[200],bufy[200],bufz[100];
  char buf1[100],buf2[100],buf3[100];
  int xi,yi,z;
  float xt,yt;
  float d;
```

```
/* Query for the value of distance between markers d */
```

```
printf("\n Please enter the value D:  ");
scanf("%f",&d);
calct(d);
system("position");
printf("input the coordinates (xi yi) :\n");
scanf("%d%d",&xi,&yi);
calc_xy(d);
xi=xi-xs;yi=yi-ys;
get_position(xi,yi);
xt= -7000 + x0 * 300;
yt= 900 - y0 * 300;
```

```
/*Calculation of coordinates of points with respect to workpiece*/
```

```
printf("x0=%f y0=%f\n",x0-xso,y0-yso);
printf("\n Y coord = %f",xt);
printf("\n X coord = %f",yt);
```

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

The following section of the code involves the movement of

the robot to the position of the workpiece. The position information is sent to the robot using the SEND program which is used to send messages to the controller of the robot

```

*****/

printf("\n Enter the position for the movement of the robot:");
scanf("%d",&z);
sprintf(bufx,"send %d ",(int ) yt);
sprintf(bufy,"send %d ",(int ) xt);
sprintf(bufz,"send defp %d",z);
sprintf(buf1,"send teach %d",z);
sprintf(buf2,"send listpv %d",z);
sprintf(buf3,"send move %d",z);
sprintf(buf4,"send undef %d",z);
system("send speed 10");
system(bufz);
system(buf1);
system(bufx);
system(bufy);
system("send 720");
system("send -692");
system("send -3");
system(buf2);
system(buf3);
system(buf4);

}

/*****
This function is to get the position of the X and Y coordinates
of the position in world coordinates from image coordinates
*****/

void get_position(int xi,int yi)

{
    f1=t1*(float)xi+t2*(float)yi;
f2=t3*(float)xi+t4*(float)yi;

```

```

x0=f1;
y0=f2;

}

```

```

/*****
This function is to compute the values of the coefficients of the
transformation matrix using the linear interpolation method.
*****/

```

```

void calct(float dum)
{
    t1 = .0446 - .000000361 * (dum - 40456) ;
    t4 = .0479 - .000000412 * (dum - 40456);
    printf("\n Value of t1 = %f", t1);
    printf("\n Value of t4 = %f",t4);
}

```

```

/*****
This function is used to initialize the coordinate system, every
time the camera is changed.
*****/

```

```

void calc_xy(float dum)
{
    if (dum >= 40494 && dum < 43070)
        { xs = 9; ys = 0;}
    if (dum >= 43070 && dum < 45700)
        { xs = 7; ys = -4;}
    if (dum >= 45700 && dum < 48300)
        { xs = 4; ys = -9;}
    if (dum >= 48300 && dum < 50950)
        { xs = -2; ys = -13;}
    if (dum >= 50950 && dum < 53570)
        { xs = -5; ys = -18;}
    if (dum >= 53570 && dum < 56210)
        { xs = -7; ys = -22;}
    if (dum >= 56210 && dum < 58840)

```

```
    { xs = -10; ys = -27;}  
if (dum >= 58840 && dum < 61470)  
    { xs = -12; ys = -30;}  
if (dum >= 61470 && dum < 64100)  
    { xs = -14; ys = -32;}  
if (dum >= 64100 && dum < 66800)  
    { xs = -16; ys = -35;}  
}
```

```

/*****
**
** COPYRIGHT NOTICE : COPYRIGHT © 1994.
**
** THIS CODE IS PROPERTY OF Dept. Of Comp.Sc.
**
**
** TITLE      :   STATENV.C
** AUTHOR     :   SUKIR S. KUMARESAN
** DATE-WRITTEN :   12/20/93
** VERSION    :   3.0
** SYSTEM REQ  :   386 PC RUNNING MS-DOS, TURBO-C version 3.0
**
**               or higher.
**
** This program is for the calibration of the hand-eye system
** for stationary environment (when the camera is stationary)
** using the linear homogeneous transformation algorithm. The
** program takes the image file as input, analyzes the position
** of the workpiece and moves the robot to the position of the
** workpiece. This program could be used either the user giving
** the values of the coefficients of the linear transformation
** matrix or by computing it at a particular position
*****/

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

/*****Global Variables *****/
t1,t2,t3,t4 are the coefficients of the linear homogeneous
transformation. They have been obtained by experimentation. xs,
ys, xso and yso are the variables used in the initializing the
system.
*****/

float t1=-1670.;
float t2=-25.;
float t3= 22.;
float t4= -1771.;

```

```
double t5= -33615.;
int xs=3,ys=13,xso=0,yso=0;
float x0,y0,f1,f2;
```

```
/******Function Prototypes*****
get_t() gets the positions from the user to compute the values of
the linear homogeneous transformation matrix. get_position() takes
two integers corresponding to X and Y coordinates in image
coordinates and compute the world coordinates of the workpiece.
*****/
```

```
void get_t(void);
void get_position( int,int);
```

```
void main()
{ char ch,bufx[200],bufy[200],bufz[100];
  char buf1[100],buf2[100],buf3[100];
  int xi,yi,z,no;
  float xt,yt;
```

```
/******
Query the user whether to use the default values for the
coefficients of the matrix or to compute the values. If the
user wants to compute the values, four points are chosen from
the image for computation.
*****/
```

```
printf("define new corresponding points?\n");
ch=getch();
if(ch=='y')
  get_t();
```

```
do{
  printf("\n Enter the position (1 or 2)");
  scanf("%d",&no);
  system("position"); /* Get the position of point from image */
```



```

    printf("input the coordinates (xi yi) :\n");
    scanf("%d%d",&xi,&yi);
    xi=xi-xs;yi=yi-ys;

    /*******
    After the computation of the coefficients, the position of the
    workpiece in world coordinates is computed.
    *****/

    get_position(xi,yi);
    xt= -6800 + (x0-xso) * 300;
    yt= 1050 - (y0-yso) * 300;

    /* Calculation of coordinates of points with respect to workpiece*/

    printf("x0=%f    y0=%f\n",x0-xso,y0-yso);
    printf("\n Y coord = %f",xt);
    printf("\n X coord = %f",yt);
    printf("play again? (y/n)\n");
    ch=getch();
    }while(ch=='y');

    /*******
    The robot is given the position information of the workpiece. The
    robot needs the values of the X,Y,Z, value of pitch P and roll R.
    The pitch and roll values are defined for the whole set of
    experiments. The robot is then moved to that position
    *****/

    printf("\n Enter the position for the movement of the robot:");
    scanf("%d",&z);
    sprintf(bufx,"send %d ",(int ) yt);
    sprintf(bufy,"send %d ",(int ) xt);
    sprintf(bufz,"send defp %d",z);
    sprintf(buf1,"send teach %d",z);
    sprintf(buf2,"send listpv %d",z);
    sprintf(buf3,"send move4 %d",z);
    system("send yes");*/
    system(bufz);
    system(buf1);
    system(bufx);

```

```

system(bufy);
system("send 550");
system("send -600");
system("send -3");
system(buf2);
system(buf3);
}

/*****
This is the function to get the coefficients of the homogeneous
transformation matrix. Four points is necessary to determine these
values.
*****/

void get_t()
{ int i,xi1,yi1,xi2,yi2,xo1,yo1,xo2,yo2;

    system("position");
    printf("input the 1st coordinates of corresponding points
           please.\n");
    printf("xi1 yi1 xo1 yo1:\n");
    scanf("%d%d%d%d",&xs,&ys,&xso,&yso);

    system("position");
    printf("input the 2nd coordinates of corresponding points
           please.\n");
    printf("xi2 yi2 xo2 yo2:\n");
    scanf("%d%d%d%d",&xi1,&yi1,&xo1,&yo1);

    system("position");
    printf("input the 3th coordinates of corresponding points
           please.\n");
    printf("xi3 yi3 xo3 yo3:\n");
    scanf("%d%d%d%d",&xi2,&yi2,&xo2,&yo2);

    xi1-=xs;yi1-=ys;xi2-=xs;yi2-=ys;
    xo1-=xso;yo1-=yso;xo2-=xso;yo2-=yso;

    t5=-((float)xi2)*((float)yi1)+((float)xi1*(float)yi2);
    i=xi1;xi1=yi2;yi2=i;yi1=-yi1;xi2=-xi2;

```

```

    t1=((float)xo1*(float)xi1)+((float)xo2*(float)yi1);
    t2=(float)xo1*(float)xi2+(float)xo2*(float)yi2;
    t3=(float)yo1*(float)xi1+(float)yo2*(float)yi1;
    t4=(float)yo1*(float)xi2+(float)yo2*(float)yi2;

    printf("t1=%f t2=%f t3=%f t4=%f t5=%lf",t1,t2,t3,t4,t5);
}

/*****
This is the function to get the values of the world coordinates
of the workpiece.
*****/

void get_position(int xi,int yi)

{
    f1=t1*(float)xi+t2*(float)yi;
    f2=t3*(float)xi+t4*(float)yi;
    x0=f1/t5;
    y0=f2/t5;
}

```

```

/*****
**
** COPYRIGHT NOTICE : COPYRIGHT © 1994.
**
** THIS CODE IS PROPERTY OF Dept. Of Comp.Sc.
**
**
** PROGRAM      :   SEND.C
** MODIFIED BY   :   SUKIR S. KUMARESAN
** DATE-MODIFIED:   12/20/93
** SYSTEM REQ    :   386 PC RUNNING MS-DOS, TURBO-C version 3.0
**
**                or higher.
**
**      *
** This program transmit and receive messages to the robotic
** controller from a program or a command line. This is useful in
** transmitting commands to the robot form the C program itself.
** Thus the Advanced Terminal Software (ATS) of the robot need
** not be used. This helps in the integration of the hand-eye
** system and its faster running.
**
*****/

```

```

#include <stdio.h>
#include <time.h>
#define com_port 1

```

```

/** Close communication and exit routine **/

```

```

fin()
{
    close_rs(com_port);
    exit(0);
}

```

```

main(argc,argv)

```

```

int argc;
char *argv[];
{char buff[300]; /* Transmit/Receive buffer */
  int i;

  /* Usage at the command line argument*/

  if (argc<=1)
    {printf("Send: syntax: send word1 word2 ... wordn\n");
      exit(1);
    }

  st_rs(com_port,9600,0,1,8,17); /* Open communication */

  /* Install control-break handler (optional). This helps in the
    termination of the program by pressing Ctrl-Break */

  ctrlbrk(fin);

  /* Make sure robot controller is ready for sending commands.
    If the controller is not ready, then the program closes
    the communication channel and exits. */

  if (!sync1())
    {printf("Send: Error in sync.\n");
      fin();
    }

  /* transmission buffer for sending commands.*/

  buff[0]='\0';
  for(i=1; i<argc; ++i)
    {strcat(buff,argv[i]);
      strcat(buff," ");
    }

```

```

    strcat(buff, "\r"); /* 'return' character at the end of line */

/* Transmission of the buffer to the controller
   for executing a command */

send_l(com_port, buff, 0, 0);

/* There is a waiting till the reception of a reply back from the
   controller. */

for(;;)
    {if (rec_l(com_port, buff, 290, '\r') < 0)
        if (rec_l(com_port, buff, 290, '\r') < 0)
if (rec_l(com_port, buff, 290, '\r') < 0)
        break;
    }

/* The ports are synchronized again for the sending of the next
   command */

sync1();

/* Exit after command is executed. */

fin();
}

/*****
   This function does synchronization. '\r' is transmitted and
   waited for '\r', '\n' and '>' response from the controller to
   complete the synchronization.
*****/

sync1()
{int ci;
  clock_t tm;

```

```
clear_b(com_port);
transmit(com_port, '\r');
tm=clock();
while((ci=read_b(com_port))!=(unsigned int) '\r')
    if (ci==-5 && (clock()-tm)>(clock_t) 18)
        return(0);
tm=clock();
while((ci=read_b(com_port))!=(unsigned int) '\n')
    if (ci==-5 && (clock()-tm)>(clock_t) 18)
        return(0);
tm=clock();
while((ci=read_b(com_port))!=(unsigned int) '>')
    if (ci==-5 && (clock()-tm)>(clock_t) 18)
        return(0);
return(1);
}
```

```

/*****
**
** COPYRIGHT NOTICE : COPYRIGHT © 1994.
**                      THIS CODE IS PROPERTY OF Dept. Of Comp.Sc.
**
**
** PROGRAM      :    TRSC.C
** MODIFIED BY   :    SUKIR S. KUMARESAN
** DATE-MODIFIED:    12/20/93
** SYSTEM REQ    :    386 PC RUNNING MS-DOS, TURBO-C version 3.0
**                      or higher.
**
**
** This module contains the function definitions to interface
** between the C and assembly routines. These routines are used
** to send and receive communication between the ports
*****/

#include <dos.h>

/*****
    This function does the opening of the communication port. The
    parameters passed are the port number, the baud rate, parity bit,
    size of length.
*****/

int far st_rs(com,brate,parity,st_bits,w_length,mod)
int com,brate,parity,st_bits,w_length,mod;
{
    _DI=1; /* Force di save */
    _SI=com-1;
    _BX=brate;
    _CL=(parity&0xff);
    _CH=(st_bits&0xff);
    _DH=(w_length&0xff);
    _DL=(mod&0xff);
    _a_st_rs();
    return(-(unsigned int) _AH);
}

/*****

```


This function does the closing of the communication port. The parameters passed is the port number and the success or failure is the return value.

*****/

```
int far close_rs(com)
int com;
{ _DI=1; /* Force di save */
  _SI=com-1;
  _a_cls_rs();
  return(-(unsigned int) _AH);
}
```

/*****/

This function does the closing of the buffer associated with the communication port.

*****/

```
int far clear_b(com)
int com;
{ _DI=1; /* Force di save */
  _SI=com-1;
  _a_clr_rs();
  return(-(unsigned int) _AH);
}
```

/*****/

This function reads a byte from the controller for the synchronization of the communication ports.

*****/

```
int far read_b(com)
int com;
{ _DI=1; /* Force di save */
  _SI=com-1;
  _a_receive();
  if (_AH==0)
    return(_AX);
}
```

```

    return(-(unsigned int) _AH);
}

```

```

/*****
    This function transmits a byte to the controller for the
    synchronization of the communication ports.
*****/

```

```

int far transmit(com,ci)
int com,ci;
{
    _DI=1; /* Force di save */
    _SI=com-1;
    _AL=(ci&0xff);
    _a_trans();
    return(-(unsigned int) _AH);
}

```

```

/*****
    This is the function used in the transmission of the command to
    the controller. The input parameters are the port number, the
    address of the buffer which stores the command for the controller
*****/

```

```

int far send_l(com,buffer,len,c)
int com,len;
char c;
char far *buffer;
{
    _DI=1; /* Force di save */
    _SI=com-1;
    _ES=FP_SEG(buffer);
    _BX=FP_OFF(buffer);
    _CX=len;
    _AL=c;
    _a_send_l();
    return(-(unsigned int) _AH);
}

```

```

/*****

```

This is the function used in the receiving of messages from the controller.

```
int far rec_l(com,buffer,len,c)
int com,len;
char far *buffer;
char c;
{_DI=1; /* Force di save */
 _SI=com-1;
 _ES=FP_SEG(buffer);
 _BX=FP_OFF(buffer);
 _CX=len;
 _AL=c;
 _a_rec_l();
 return(-(unsigned int) _AH);
}
```