Implementation of a Pick-and-Place Robot With Manipulator and Camera

Bernard Brochu

A Major Technical Report
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

August 1985

ISBN 0-315-30600-9

ABSTRACT

Implementation of a Pick-and-Place Robot
with Manipulator and Camera

Bernard Brochu

This major report describes a particular implementation of a low cost pick-and-place robot. Vision is used to determine the position of the manipulator instead of the common homogeneous transformations which require use of internal sensors. A discussion on a common robot programming approach is made starting with the introduction of homogeneous transformations which are then applied in a typical assembly task. The report describes in detail the physical and functional aspects of the manipulator and the camera that were used in the implementation. The task, physical installation, and restrictions are defined, and the control software exhaustively described.

ACKNOWLEDGEMENTS

I would like to acknowledge Dr. Fancott for his precious contribution and contagious enthusiasm during all phases of this project. I also take this opportunity to thank my sister Andrée who is responsible for most of the drawings related to the robot manipulator.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER 0

INTRODUCTION

The goal of the project described here, is to experiment on the implementation of a pick-and-place robot using a micro-computer and low-cost manipulator/camera. A particularity of the implementation is that it does not use homogeneous transformations as the primary tools for describing the position of the manipulator.

The most common representation for object positions in robotics and graphics is the homogeneous transform (Paul, 1981). An alternative approach to this world modeling method has been implemented. One of the objects involved in a robot task, is the manipulator itself and in the homogeneous transformation description of the world, its position is known only if one has an exact knowledge of the values of its joint variables. This approach requires internal sensors for each degree of freedom of the manipulator.

Humans don't control their moves in terms of their joint coordinates but rather continuously adjust their arms with what they see. The adjustment is made as a function of the relative position of the object to be picked, and the arm. In this project, the position of the manipulator is obtained from the camera, not from internal sensors in the manipulator. This approach has the advantage of using the full power of vision with less calculation on coordinate frame transformations, but requires more computing time in image processing. Another advantage is that the camera may be used to compensate for inexact devices. Using vision, less demands are put on the manipulator precision (repeatability) which is, with the pay load and the dimensions, the main justification for high prices of manipulators. The task implemented here could be repeated almost indefinitely if each degree of freedom could be "seen" by the camera.

As will be seen in chapter 1, the homogeneous transformation description requires an exact quantitative knowledge of the geometry (lengths and angles) of all links of the manipulator. Our method only requires qualitative descriptions on which the control software is based. The quantitative data are obtained by having the robot arm move in the camera field and make automatic computations on the images obtained. The arm will first learn to move in its own space, it will then perform tasks.

Chapter 1 introduces the homogeneous transformations which are then used, as an example, to describe our manipulator. A typical task is then described in terms of homogeneous transformation equations. Finally, use of sensors and other aspects of robot programming are discussed.

Chapter 2 gives a description of the manipulator used in the project.

Chapter 3 defines the project, its physical installation and its restrictions.

Chapter 4 describes in detail, the camera system for the project.

Chapter 5 gives a description of the internal functionality of the control program and of its external functionality as seen by the user.

CHAPTER 1

HOMOGENEOUS TRANSFORMATIONS AND ROBOT PROGRAMMING

1.1 Introduction

This chapter describes the most common method employed in describing the position of a robot manipulator (or arm) and the world with which it interracts (this is world modeling). The method uses homogeneous transformations which are tools to describe the relationships between objects. Homogeneous transforms are also used in other fields such as computer vision and computer graphics and will be used here to describe the manipulator (Paul, 1981).

Each degree of freedom of the manipulator will be specified by an homogeneous transformation matrix relating it to the previous one.

Notation for vectors and transformations are first introduced. As special cases, translation and rotation transformations are then defined and a geometrical interpretation is given for the product of such transformations. We will see that this representation may be used to represent rigid objects. The meaning of an inverse transformation will be explained. Transformations will be used to describe the manipulator of the project and a typical task will be described in terms of transformation equations. Finally, the use of sensors and other aspects of robot programming will be discussed.

1.2 Vectors

In the homogeneous coordinate notation, a point vector $\mathbf{v} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$ is represented as a column matrix :

$$\mathbf{v} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = [x, y, z, w]^T$$

where the superscript T represents the transpose of the row matrix and where $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are unit vectors along the x,y, and z coordinate axes, respectively and where

$$a = x/w$$
$$b = y/w$$
$$c = z/w$$

In this notation, a vector $[a, b, c, w]^T$ can be multiplied by any non-zero scalar n without changing the vector it represents:

$n[x, y, z, w]^T = [nx, ny, nz, nw]^T$ represents vector $(nx/nw)\mathbf{i} + (ny/nw)\mathbf{j} + (nz/nw)\mathbf{k} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$.

The vector at the origin, the null vector, is represented as $[0, 0, 0, n]^T$ where n is any non-zero scale factor. The vector $[0, 0, 0, 0]^T$ is undefined.

Vectors of the form $[a, b, c, 0]^T$ represent vectors at infinity and are used to represent directions; the addition of any other finite vector does not change their value in any way. To demonstrate this, lets take two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$

$$\mathbf{v}_1 = x_1\mathbf{i}/w_1 + y_1\mathbf{j}/w_1 + z_1\mathbf{k}/w_1$$

$$\mathbf{v}_2 = x_2\mathbf{i}/w_2 + y_2\mathbf{j}/w_2 + z_2\mathbf{k}/w_2$$

the vector sum $\mathbf{v}_1 + \mathbf{v}_2$ represented as a column matrix is

$$\begin{bmatrix} w_2x_1 + w_1x_2 \\ w_2y_1 + w_1y_2 \\ w_2z_1 + w_1z_2 \\ w_2w_1 \end{bmatrix}$$

now if $\mathbf{v}_1$ is a direction vector then $w_1 = 0$ and the sum reduces to

$$\begin{bmatrix} w_2x_1 \\ w_2y_1 \\ w_2z_1 \\ 0 \end{bmatrix} = w_2\mathbf{v}_1 = \mathbf{v}_1$$

which demonstrates that $\mathbf{v}_1$ has not been modified by the addition of a finite vector.

1.3 Transformations

A transformation $\mathbf{H}$ of the space, is a 4X4 matrix and can represent translation, rotation, stretching and perspective transformations. Only translation and rotation will be described here.

Given a point $\mathbf{u}$, its transformation $\mathbf{v}$ is represented by the matrix product

$$\mathbf{v} = \mathbf{H}\mathbf{u}$$

As we will see, there are two interpretations for this transformation. One interpretation sees $\mathbf{u}$ as a vector described in the reference coordinate frame. Applying the transformation $\mathbf{H}$ moves point $\mathbf{u}$ in space to a new position $\mathbf{v}$. The other interpretation sees $\mathbf{H}$ as the description of a coordinate frame made with respect to the reference coordinate frame, and $\mathbf{u}$ as a vector described in frame $\mathbf{H}$. The transformed vector $\mathbf{v}$ is the same vector but its description is now made with respect to the reference coordinate frame.

1.3.1 Translation and Rotation transformations

A general translation transformation is represented as

$$\mathbf{Trans}(a,b,c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and a general rotation transformation is represented by

$$\begin{bmatrix} n_x & o_x & a_x & 0 \\ n_y & o_y & a_y & 0 \\ n_z & o_z & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It may be shown that any combination of rotations is always equivalent to a single rotation about some vector **s** by some angle θ.

Below are the transformations corresponding to rotations about the x, y, or z axis by an angle θ.

$$\mathbf{Rot}(x,\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{Rot}(y,\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot(z,\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.3.2 Coordinate Frames

As already mentioned, a transformation can be interpreted as a description of a coordinate frame with respect to the reference coordinate frame: the last column of the matrix specifies the origin and the first three specify the orientations of the three axes.

As an example consider the following transformation

$$\begin{bmatrix} 0 & -1 & 0 & 7 \\ 1 & 0 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

which represents a translation in the XY plane and a rotation of +90° about the z axis. The first column is a direction vector which implicitly describes the unit vector $\mathbf{i}'$ of the x axis of the frame in terms of the units vectors $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ of the reference coordinate frame

$$\mathbf{i}' = 0\mathbf{i} + 1\mathbf{j} + 0\mathbf{k}$$

In the same manner

$$\mathbf{j}' = -1\mathbf{i} + 0\mathbf{j} + 0\mathbf{k}$$

$$\mathbf{k}' = 0\mathbf{i} + 0\mathbf{j} + 1\mathbf{k}$$

The last column tells us that the origin is at

$$7\mathbf{i} + 5\mathbf{j} + 0\mathbf{k}.$$

In the above example we have combined a translation $\mathbf{T}$

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 7 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and a rotation $\mathbf{R}$

$$\mathbf{R} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The final transformation is the matrix product $\mathbf{TR}$ which is different* from $\mathbf{RT}$ since matrix multiplication is not commutative. The interpretation of $\mathbf{TR}$ or $\mathbf{RT}$ may be clarified if we introduce the following general rule: if we postmultiply a transform $\mathbf{F}$ representing a frame by a second transformation $\mathbf{G}$ describing a rotation and/or translation, we must interpret that rotation and/or translation with respect to the frame axes described by the first transformation. If we premultiply the frame transformation **F** by **G** then the rotation and/or translation must be interpreted with respect to the reference coordinate frame.

If we interpret our example as frame **T** being postmultiplied by transformation **R**, we must consider the rotation with respect to frame **T**. If the interpretation is that frame **R** is premultiplied by transformation **T**, then we must interpret the translation with respect to the reference coordinate frame. Both interpretations lead to the same result. Figures 1.1 and 1.2 illustrate the two interpretations of **TR** and **RT** respectively.

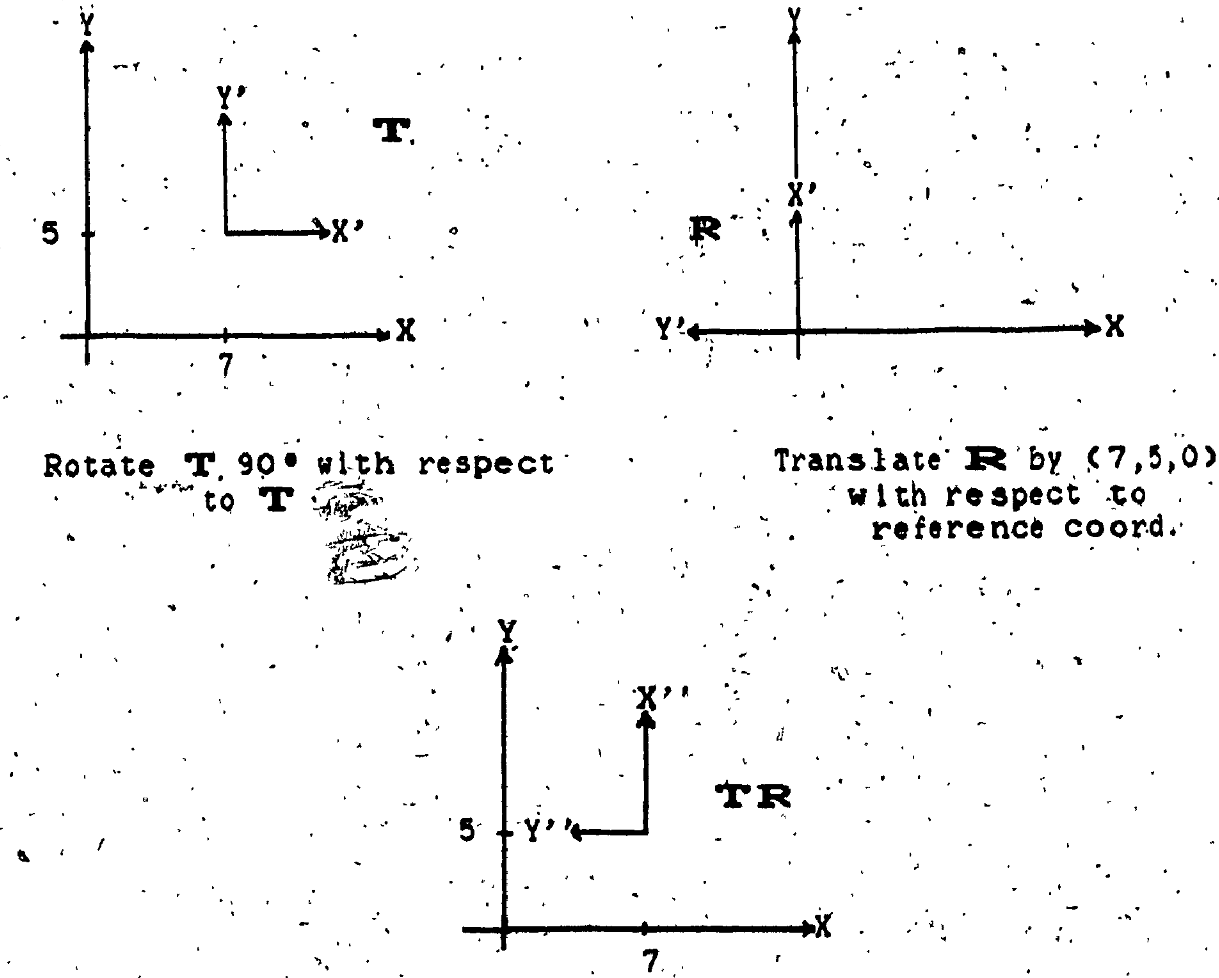
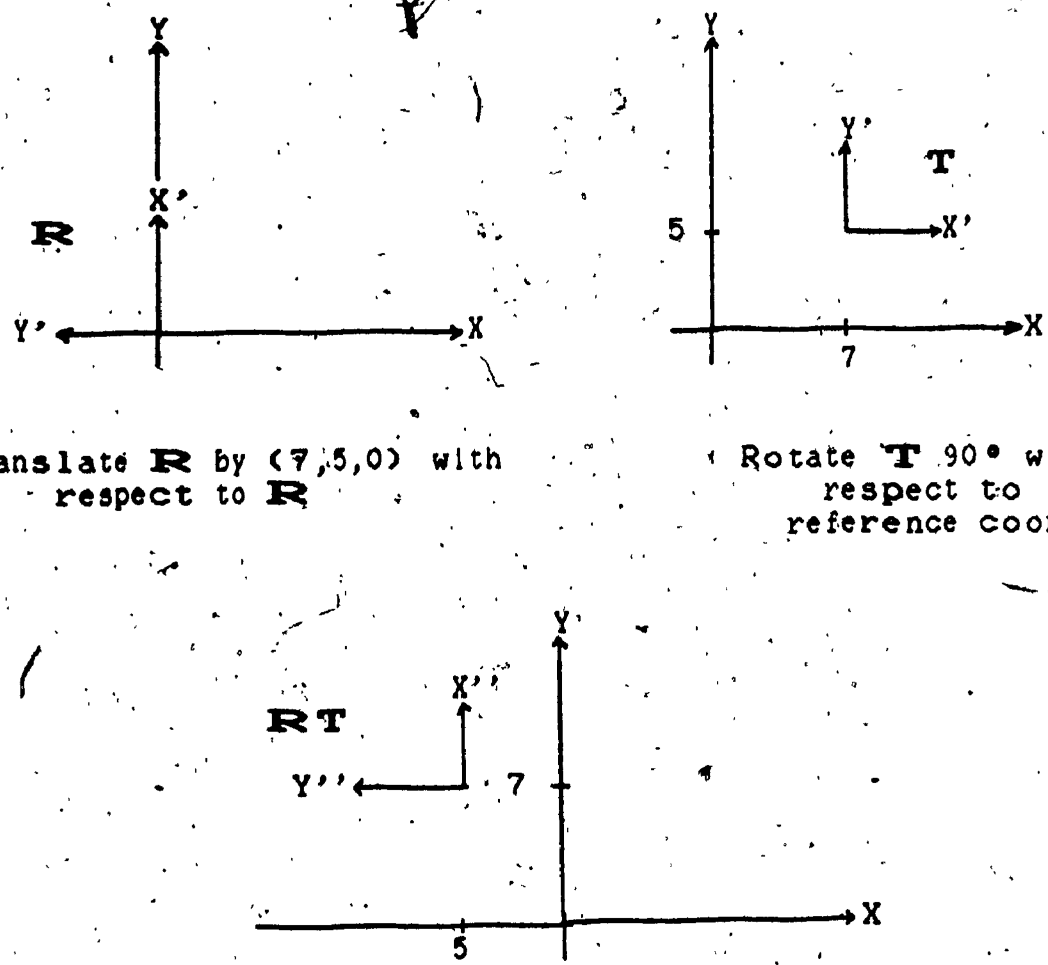


Fig 1.1 The two interpretations of the matrix product **TR**. On the left hand side, frame **T** is postmultiplied by rotation **R** giving **TR**. On the right hand side, frame **R** is premultiplied by translation **T** giving the same **TR**.



Translate **R** by (7,5,0) with respect to **R**

Rotate **T** 90° with respect to reference coord.

Fig 1.2 The two interpretations of the matrix product **RT**. On the left hand side, frame **R** is postmultiplied by translation **T** giving **RT**. On the right hand side, frame **T** is premultiplied by rotation **R** giving the same **RT**.

1.3.4 Objects

Transformations are useful for representing the position of rigid objects. If the object can be described by a set of n points relative to a coordinate frame fixed in that object, then the transformation describing that coordinate frame with respect to a reference coordinate frame, may be postmultiplied by a 4Xn matrix whose n columns represent the n points of the object. The resulting 4Xn matrix will then give the position of the object with respect to the above reference frame.

1.3.5 Inverse Transformations

A transformation **T** describes a coordinate frame with respect to the reference coordinate frame. The inverse of the matrix describing **T** represents the inverse transformation $\mathbf{T}^{-1}$ and describes the reference coordinate frame with respect to the transformed coordinate frame.

Let's consider the transformation of the last example

$$\begin{bmatrix} 0 & -1 & 0 & 7 \\ 1 & 0 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

for which the inverse transformation is

$$\begin{bmatrix} 0 & 1 & 0 & -5 \\ -1 & 0 & 0 & 7 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This last transformation represents a translation of (-5, 7, 0) and a rotation of -90° about the z axis of the translated frame. As illustrated in figure 1.3, this is exactly the description of the reference coordinate frame with respect to the transformed frame.

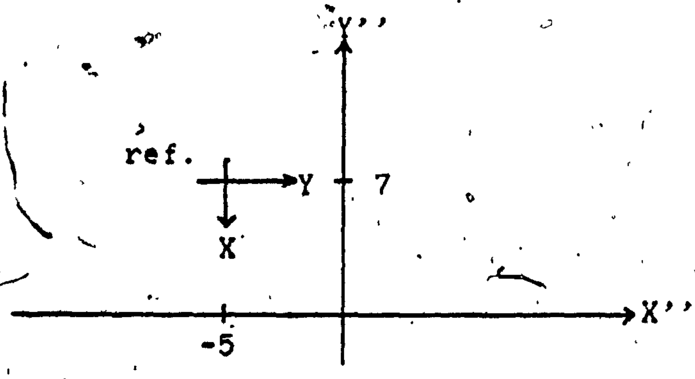


Fig 1.3 Reference coordinate frame expressed in transformed frame.

1.4 The Manipulator

Our manipulator has five degrees of freedom (excluding the gripper), one for each link. Figure 1.4 shows the manipulator in an arbitrary position.

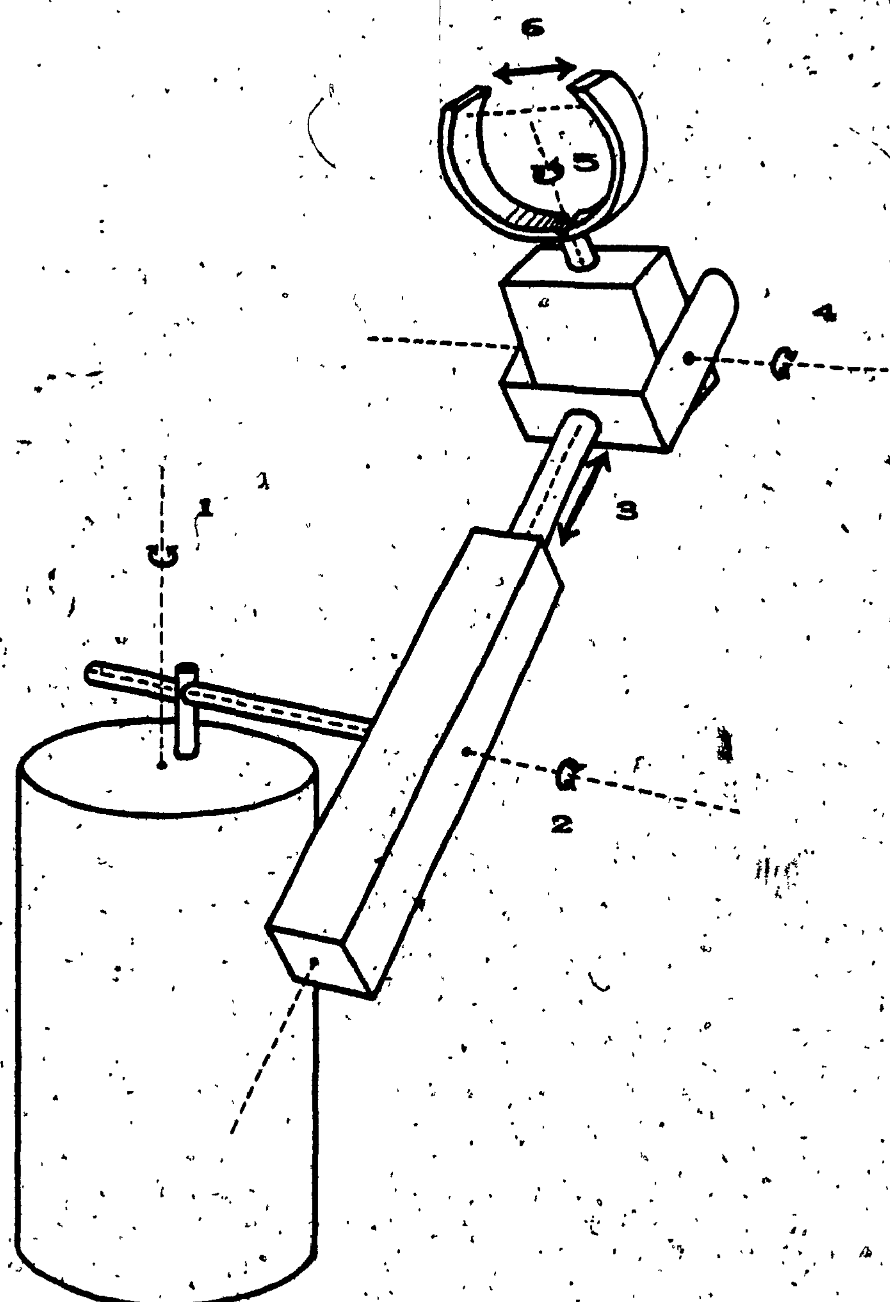


Fig 1.4 The manipulator and its axes.

We define the homogeneous transform $\mathbf{T}_5$ as the coordinate frame describing the position and orientation of its end effector as shown in figure 1.5

$$\mathbf{T}_5 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The origin of the describing frame is centrally located between the finger tips and is described by vector $\mathbf{p}$. The three unit vectors $\mathbf{n}$, $\mathbf{o}$, and $\mathbf{a}$ describe the hand's orientation. The z vector lies in the direction from which the hand would approach an object and is known as the approach vector $\mathbf{a}$. The y vector is in the direction specifying the orientation of the hand, from fingertip to fingertip and is known as the orientation vector, $\mathbf{o}$. The normal vector $\mathbf{n}$, forms a right-handed set of vectors and is specified by the vector cross-product $\mathbf{n} = \mathbf{o} \times \mathbf{a}$.

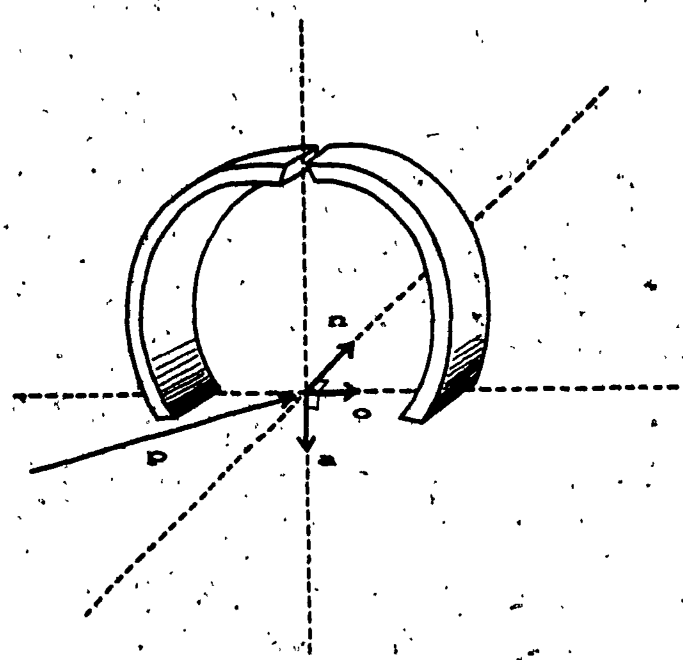


Fig 1.5 Coordinate frame of the end effector.

The manipulator can be considered to consist of a series of links connected together by joints. Each link will be described by a coordinate frame fixed in it. The relative position and orientation between these coordinate frames can be described using homogeneous transformations. Such an homogeneous transformation is called an $\mathbf{A}$ matrix. $\mathbf{A}_1$ describes the coordinate frame of link 1 with respect to some reference coordinate frame. $\mathbf{A}_n$ describes the coordinate frame of link n with respect to coordinate frame of link n-1. Since $\mathbf{A}_5$ describes the position and orientation of the end effector with respect to coordinate frame of link 4, we can write

$$\mathbf{T}_5 = \mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5$$

For our 5 degrees of freedom manipulator, there will be 5 links and 5 joints. The base of the manipulator is link 0 and is not considered one of the five links. Link 1 is connected to the base by joint 1. There is no joint at the end of the final link. The only significance of links is that they maintain a fixed relationship between the manipulator's joints at each end of the link.

Any link can be characterized by two dimensions: the common normal distance a_n (length of the link), and the angle α_n (twist of the link) between the axes in a plane perpendicular to a_n.

An axis will have two normals to it, one for each link. The relative position of two such connected links is given by d_n, the distance between the normals along the joint n axis, and θ_n the angle between the normals measured in a plane normal to the axis. d_n and θ_n are called the distance and the angle between the links, respectively. A revolute joint is one for which θ_n is the joint variable, and a prismatic joint is one for which d_n is the variable.

There exists a standard general method for assigning a coordinate frame to each link of a manipulator. The method has been applied to our manipulator and the results are shown in figure 1.7. Figure 1.6 shows the manipulator in an arbitrary chosen initial position and figure 1.7 shows the coordinate frame assignment for that position of the arm.

Joints 1, 2, 3, and 4 are revolute joints and their corresponding variables are θ_1, θ_2, θ_4, and θ_5 respectively. Joint 3 is the only prismatic joint and has d_3 as joint variable. The initial position of figure 1.6 is such that $\theta_1=\theta_2=\theta_4=\theta_5=0$ and such that $d_3=d_3^{\bullet}$. θ_n may be seen as the angle between axes x_n and x_{n-1} measured in a plane normal to z_{n-1}. For example, if a rotation is made from the initial position about axis 2, then x_1 and x_2 will no longer be parallel and the move will be described by θ_2. For prismatic joint 3, the variable is d_3 which is the distance between x_3 and x_2 measured along axis 3.

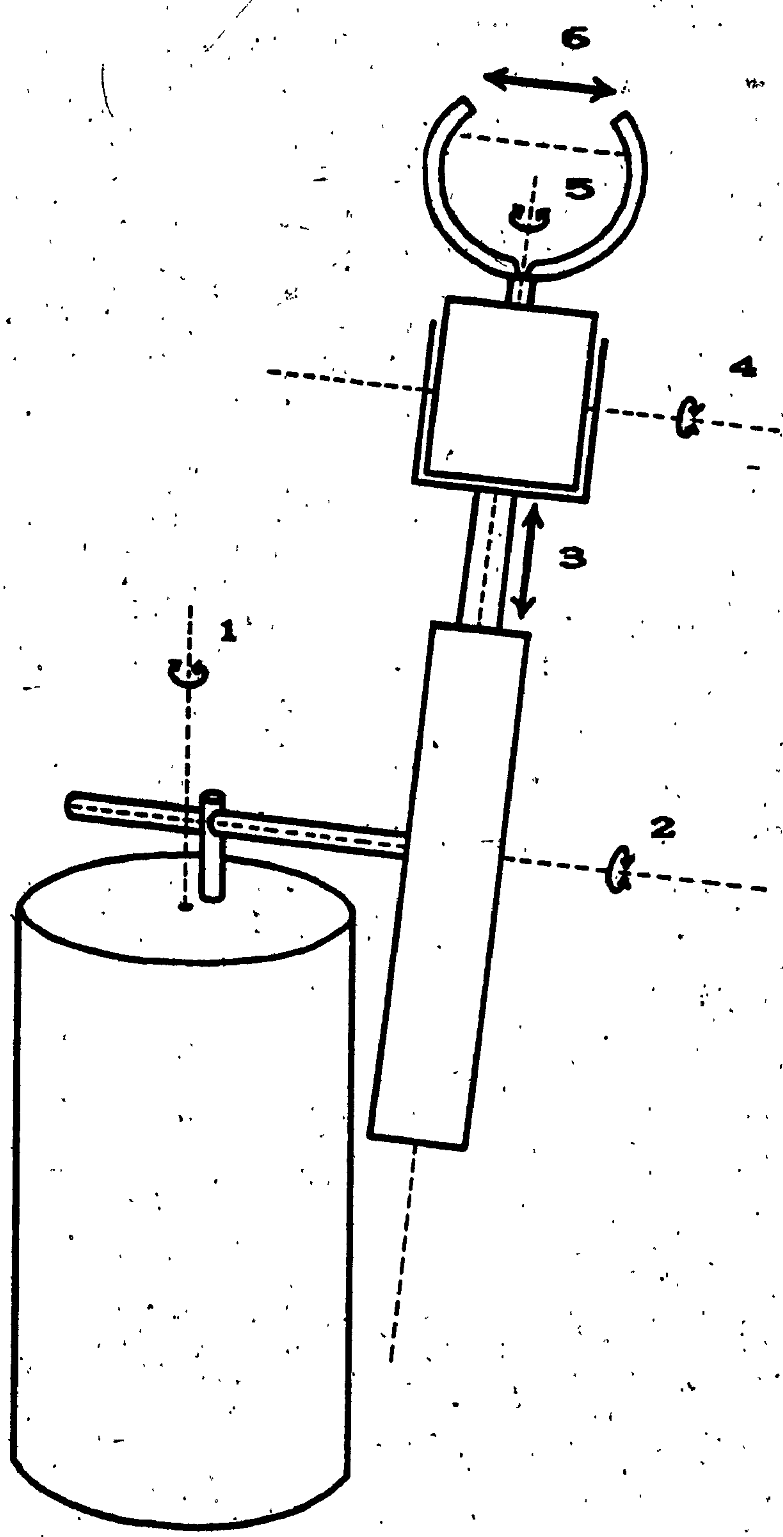


Fig 1.6 Manipulator in an arbitrarily chosen initial position.

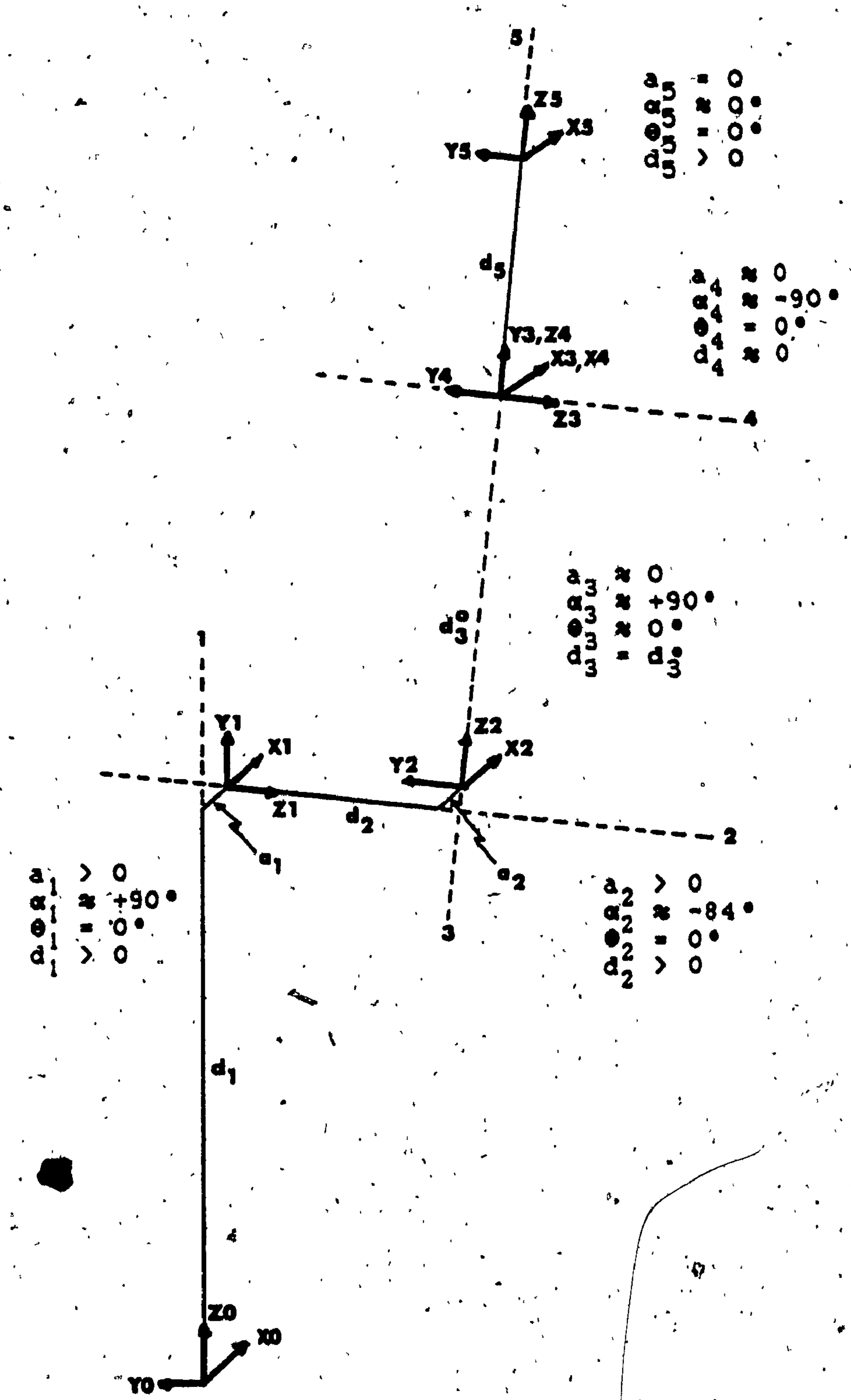


Fig 1.7 Coordinate frame assignment

Now that coordinate frames have been assigned to all links, we can establish the relationships between successive frames n-1, n by the following rotations and translations

rotate about Z_{n-1} an angle θ_n;

translate along Z_{n-1}, a distance d_n;

translate along rotated $X_{n-1} = X_n$ a length a_n;

rotate about X_n, the twist angle α_n.

This may be expressed as the product of four homogeneous transformations relating the coordinate frame of link n to the coordinate frame of link n-1

$$\mathbf{A}_n = \mathbf{Rot}(z,\theta)\ \mathbf{Trans}(0,0,d)\ \mathbf{Trans}(a,0,0)\ \mathbf{Rot}(x,\alpha)$$

$$\mathbf{A}_n = \begin{bmatrix} \cos\theta & -\sin\theta\cos\alpha & \sin\theta\sin\alpha & a\cos\theta \\ \sin\theta & \cos\theta\cos\alpha & -\cos\theta\sin\alpha & a\sin\theta \\ 0 & \sin\alpha & \cos\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In order to obtain the five **A** matrices for the manipulator, one must measure the constants a, α, and d for joints 1, 2, 4, and 5; and the constants a, α, and θ for joint 3. Once this is done, $\mathbf{T}_5$ may be expressed as a single transformation matrix with θ_1, θ_2, d_3, θ_4, and θ_5 as variables. Given as input, a set of values for the variables, $\mathbf{T}_5$ can easily be evaluated for the position and orientation of the end effector. But we normally know where we want to move the manipulator in terms of T_5 and we need to obtain the joint coordinates in order to make the move. Obtaining a solution for the joint coordinates is a difficult problem.

Joint coordinate solutions are obtained by equating transform expressions. For each transform equation we obtain 12 non-trivial equations and it is these equations which will yield the required solution. The solution is obtained in a sequential manner, isolating each variable by premultiplication by a number of the transforms in each equation.

1.5 A typical Task

A typical task would consist in picking up rectangular pins, such as described in figure 1.8, and inserting them into holes in a subassembly as shown in figure 1.9.

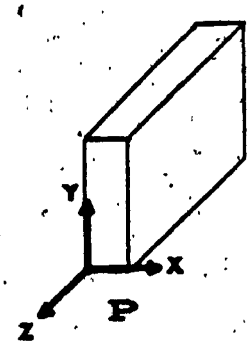


Fig 1.8 Rectangular pin with its coordinate frame.

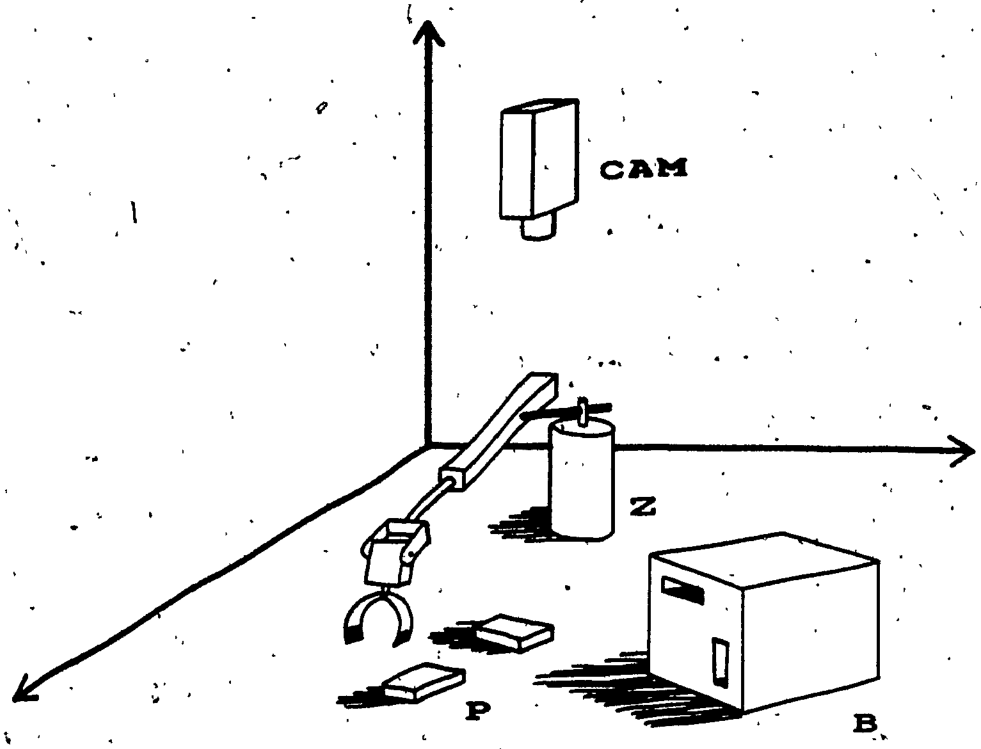


Fig 1.9 The objects involved in the task.

As mentioned earlier, any rigid object can be described in terms of coordinate system bearing a fixed relationship to the object. If we have knowledge of an object relative to a defining coordinate system as shown in figure 1.8, then the only information necessary to define the object in space, is the location and orientation of the object's coordinate frame.

The position of the manipulator will be described as the product of three transformations

$$Z \, T_5 \, E$$

where

Z represents the position of the base of the manipulator with respect to the base coordinate frame;

$\mathbf{T}_5$ represents the end of the manipulator with respect to its base. $\mathbf{T}_5$ is a computable function of the joint coordinates;

E represents a tool or end effector at the end of the manipulator, with respect to frame $\mathbf{T}_5$.

With such a description, the calibration of the manipulator to the work station is represented by **Z**. If the task is to be performed with a change of tool, only **E** must be changed.

The task may be broken into subtasks. The following transformations represent the positions associated with the subtasks and the relationship between the objects involved.

P represents the position of the pin in base coordinates;

B represents the position of the block with the two holes, with respect to the base coordinates;

$\mathbf{H}_i$, i=1,2, represents the position of the i'th hole in the block with respect to the **B** coordinate frame;

GHP, GAP, GDP represent the position of the gripper holding the pin, approaching the pin, and departing with the pin respectively, all with respect to the pin coordinate system;

PAH, PCH, PBI, PI represent the position of the pin approaching a hole, in contact with a hole, at begining of insertion into a hole, and when inserted into a hole respectively, all with respect to the hole coordinate frame;

CAM represents the camera coordinate system in base coordinates;

PC represents the position of the pin in the **CAM** coordinates.

One can classify the above 16 transforms (**H_1** accounts for two) into two groups. The first group consists of transforms that do not vary during a task execution or from one execution to the another. We can further subdivide this group into those who can be symbolically defined: **Z**, **E**, **H_i**, **PI**, **GHP**; and those who are defined using teaching-by-doing: **GAP**, **GDP**, **B**, **PBI**, **PCH**, **PAH**, **CAM**. The second group consists of transforms whose values are determined at task execution time. These task variable transforms are: **P**, **T_5**, and **PC**.

Transforms **Z**, **E**, and **GHP** are determined at installation time from physical measurements. Transforms **H_1** and **H_2** may be obtained from engineering drawings describing the block as shown in figure 1.10.

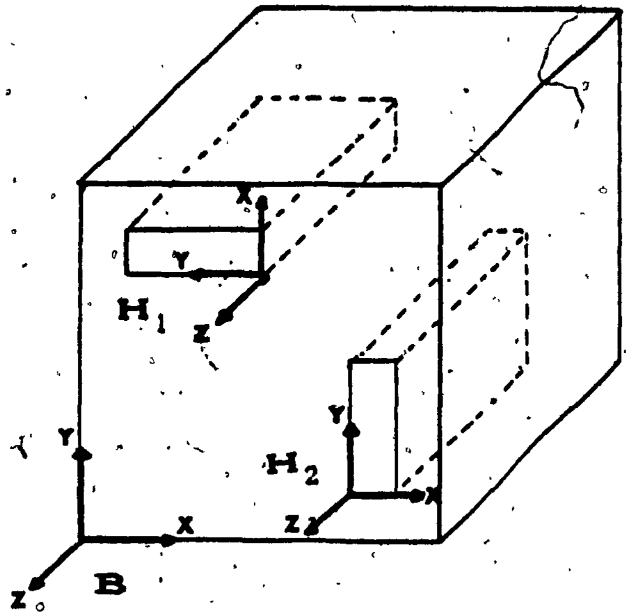


Fig 1.10 Coordinate frames describing the holes.

PI is determined such that the x, y, and z axes of the pin agree with those of the hole, and such that the origins of the hole and the pin are separated by a constant distance along the common z axis.

After **Z**, **E**, $\mathbf{H}_1$, $\mathbf{H}_2$, **PI** and **GHP** have been defined, the manipulator can be used in the teaching-by-doing mode to define the remaining constant transforms.

The end effector is placed on the pin at its pick up position and the following transform equation is true

$$\mathbf{Z}\ \mathbf{T}_5\ \mathbf{E} = \mathbf{P}\ \mathbf{GHP}$$

which defines **P** since **Z**, **E** and **GHP** are known constants and since $\mathbf{T}_5$ may be obtained from the values of the joint variables.

$$\mathbf{P} = \mathbf{Z}\ \mathbf{T}_5\ \mathbf{E}\ \mathbf{GHP}^{-1}$$

The camera control is then asked to return the position **PC** of the pin relative to the **CAM** coordinates. Since the above particular position **P** of the pin is known, we can write

$$P = CAM\ PC$$

which defines **CAM**

$$CAM = P\ PC^{-1}$$

The gripper is then moved back to the approach position and we have

$$Z\ T_5\ E = P\ GAP$$

defining **GAP**

$$GAP = P^{-1}\ Z\ T_5\ E$$

A departure point relative to **P** is now defined by lifting the pin in the gripper to the departure position **GDP**

$$Z\ T_5\ E = P\ GDP\ GHP$$

from which we obtain

$$GDP = P^{-1}\ Z\ T_5\ E\ GHP^{-1}$$

The manipulator is then used to insert the pin into hole 1 and we have

$$Z\ T_5\ E = B\ H_1\ PI\ GHP$$

solving for **B**

$$B = Z\ T_5\ E\ (H_1\ PI\ GHP)^{-1}$$

The gripper is then moved back and

$$Z\ T_5\ E = B\ H_1\ PBI\ GHP$$

from which

$$PBI = (B\ H_1)^{-1}\ Z\ T_5\ E\ GHP^{-1}$$

The pin on first contact with the hole **PCH** and an approach point **PAH** are defined by

$$PCH = (B\ H_1)^{-1}\ Z\ T_5\ E\ GHP$$

$$PAH = (B\ H_1)^{-1}\ Z\ T_5\ E\ GHP$$

which complete the description of the 13 constant transforms.

1.5.1 The Program for the Task

Before going into the description of the program we define a general form for position equations

$$T_5\ TOOL = COORD\ POS$$

where **COORD** represents a general expression for a working coordinate system and where **TOOL** represents a general expression for the tool.

The first move requires that

$$Z\ T_5\ E = P\ GAP$$

which implies that

$$T_5\ E = Z^{-1}\ P\ GAP$$

and that **COORD** and **TOOL** must be defined as

$$COORD := Z^{-1}\ P$$

$$TOOL := E$$

In this context the first move is defined as

MOVE **GAP**

which must be interpreted as: substitute **GAP** for **POS** in equation

$$T_5 \; TOOL = COORD \; POS,$$

solve for T_5, and input T_5 to the manipulator control.

The second move requires that

$$Z \; T_5 \; E = P \; GHP$$

which gives

$$T_5 \; E = Z^{-1} \; P \; GHP$$

For this second move the values of COORD and TOOL need not be changed and the move is defined as

MOV GHP

The third move comes after the pin has been grasped and requires that

$$Z \; T_5 \; E = P \; GDP \; GHP$$

which implies that

$$T_5 \; E \; GHP^{-1} = Z^{-1} \; P \; GDP$$

In this case again COORD need not be changed but TOOL does

$$TOOL := E \; GHP^{-1}$$

The third move is defined as

MOV GDP

The fourth to seventh moves require that

$$Z \; T_5 \; E = B \; H_i \; PAH \; GHP$$

$$Z \; T_5 \; E = B \; H_i \; PCH \; GHP$$

$$Z \; T_5 \; E = B \; H_i \; PBI \; GHP$$

$$Z \; T_5 \; E = B \; H_i \; PI \; GHP$$

respectively. These imply that

$$T_5\ E\ GHP^{-1} = Z^{-1}\ B\ H_i\ PAH$$

$$T_5\ E\ GHP^{-1} = Z^{-1}\ B\ H_i\ PCH$$

$$T_5\ E\ GHP^{-1} = Z^{-1}\ B\ H_i\ PBI$$

$$T_5\ E\ GHP^{-1} = Z^{-1}\ B\ H_i\ PI$$

which define a new **COORD**

$$COORD := Z^{-1}\ B\ H_i$$

The four moves become

MOV **PAH**

MOV **PCH**

MOV **PBI**

MOV **PI**

At this point the pin is inserted. It is then released.

In the last move, the gripper departs from the pin. It requires that

$$Z\ T_5\ E = B\ H_i\ PI\ \ GAP$$

which implies that

$$T_5\ E = Z^{-1}\ B\ H_i\ PI\ \ GAP$$

and that **TOOL** and **COORD** must be reassigned

$$TOOL := E$$

$$COORD := Z^{-1}\ B\ H_i\ PI$$

The last move is then defined as

MOV **GAP**

The program to insert the pins can now be described.

```
TOOL := E;
for i := 1 to 2 do
    begin
    read(camera, PC);              read in position of pin
                                        relative to CAM
    P := CAM PC;                   position of pin in base coord
    COORD := Z^-1 P;               coordinate wrt pin
    MOVE GAP;                      approach...
    MOVE GHP                                ...over pin
    GRASP;
    TOOL := E GHP^-1;              Tool now end of pin
    MOVE GDP;                      departure position
    COORD := Z^-1 B H_i;           coord wrt to hole
    MOVE PAH;                      approach hole
    MOVE PCH;                      contact with hole
    MOVE PBI;                      just before insertion
    MOVE PI;                       insert pin
    RELEASE;
    COORD := Z^-1 B H_i PI;        coord wrt pin
    TOOL := E;
    MOVE GAP;                      depart from inserted pin
    end;
```

1.5.2 Sensors for the Task

The above task makes use of four types of sensors (Lozano-Pérez, 1983):

Direct position sensors are internal sensors in the manipulator joints and are used to determine the position of the manipulator. These sensors implicitly give the values of the joint variables from which the **A** matrices and then the position of the end effector ($\mathbf{T}_5$) can be computed.

Vision sensors such as the camera are used to determine the position of parts. In our project the camera is also used (instead of internal sensors) to determine the position of the manipulator.

Sensors in the fingers are used to control the magnitude of the gripping force and to detect the presence or absence of objects between the fingers. In the above application, this type of sensor is used implicitly in the GRASP procedure.

Wrist force sensors are necessary because uncertainty in part positions, positioning errors in the manipulator, errors in grasping position and part tolerances make it impossible to reliably position parts relative to each other accurately enough for tight tolerance assembly. The forces generated as the assembly progresses are used to suggest incremental motions that will achieve the desired final state. This type of motion is known as "active compliant motion" in contrast with "passive compliant motion" achievable with mechanical devices specially designed for this type of operation. The active compliance approach is not as fast or as robust as the dedicated mechanical compliance devices but is more easily adaptable to changing applications. In the previous task, any compliance is specified in the coordinate frame of the right hand most transform of the **TOOL**, while effects of compliance are accounted for by modifying one of the transforms of the **COORD** expression.

In general, there are four principal uses of sensing in robot programming (Lozano-Pérez, 1983; Albus, 1981;

Nitzan, 1983): initiating and terminating motions; choosing among alternative actions; obtaining the identity and position of objects and features of objects; and complying to external constraints.

An example of the first use of sensors would be a robot waiting for an external binary signal, before proceeding with execution of a program or to synchronize with other machines. An other use would be to locate an imprecisely known surface by moving toward it and terminating the approach motion when a microswitch is tripped or when the value of a force sensor exceeds a threshold. This process is known as "guarded move" or "stop on force". Guarded moves can be used to identify points on the edges of an imprecisely located object such as a pallet. The contact points can then be used to determine the pallet's position relative to the manipulator and supply offsets for subsequent pickup motions.

Examples of the second use of sensors are deciding whether or not to discard an object after an inspection test or to decide if a grasp or insert action had the desired effect. Such error checking tests require multiple sensors: visual, force, position.

The last use of sensors involves active compliance. Active compliance is necessary in situations requiring continuous motion - in response to continuous sensor input or when the force-controlled motions require that the target position of the manipulator from instant to instant be determined from the direction and the magnitude of the forces acting on the manipulator hand.

1.6 Other aspects of robot programming

In the previous task, we have assumed that a robot motion is specified by its final position, in many cases, this is not sufficient; a path for the robot must also be specified. Paths are commonly specified by indicating a sequence of intermediate positions, known as "via points", that the robot should traverse between the initial and final positions. The shape of the path between via points is chosen from among some basic repertoire of path shapes implemented by the robot control system. Three types of paths are implemented in current systems: uncoordinated joint motions, straight lines in the joint coordinate space, and straight lines in Cartesian space.

Another aspect to be considered, is the general approach to robot programming. The possible approaches may be classified in three categories: teaching by showing, robot-level programming, and task-level programming.

In the teaching by showing (or guiding) approach, the robot is manually moved to each desired position and the internal joint coordinates are recorded. In this case, a program is a sequence of vectors of joint coordinates and activation signals (e.g. closing the gripper). This approach requires no general-purpose computer but is limited since a program cannot specify desired action of the robot in response to sensory input. There are no loops, conditionals or computations in the program.

Robot-level programming enables the data from external sensors (e.g. vision, force) to be used in modifying the robot's motions. A disadvantage of this approach is that it requires the programmer to be expert in computer programming and in the design of sensor-based motion strategies.

In the task-level programming approach, goals are specified for the positions of objects rather than the motion of the robot needed to achieve those goals. The task description is robot independent, the user does not specify positions or paths that depend on the robot geometry or kinematics. Such systems are also referred to as "world modeling" systems because they require complete geometric models of the environment and of the robot as input. Unlike the first approaches, this type of system has not reached the commercial stage.

CHAPTER 2

THE ROBOT MANIPULATOR

The manipulator (see fig. 1.4) has four axes of rotation: 1, 2, 4, and 5, controlled respectively by motors 1, 2, 6, and 3. Axis 3 is the only axis of translation and is controlled by motor 4. Motor 5 is used in closing or opening the gripper.

2.1 Arm Control

The control to the arm is accomplished via a 20 pin connector (pins A1..A20) of which only 15 are used:

```
Pins A1,A2,A3,A4 control coils 1,2,3,4 respectively of
                        motor 1 if pin A20 is active (= 1)
                        motor 4 if pin A19 is active (= 1)

Pins A5,A6,A7,A8 control coils 1,2,3,4 respectively of
                        motor 2 if pin A20 is active (= 1)
                        motor 5 if pin A19 is active (= 1)

Pins A9,A10,A11,A12 control coils 1,2,3,4 respectively of
                        motor 3 if pin A20 is active (= 1)
                        motor 6 if pin A19 is active (= 1)
```

Pin A18 is the logical ground

Pin A19 enables motors 4, 5, 6 when in logic state 1

Pin A20 enables motors 1, 2, 3 when in logic state 1

The control from the computer is provided by a standard IBM PC parallel port interface card with DB-25 connector. The interface has been modified to provide the +5V power. The computer and the arm are mated by a driver/buffer card as shown in figure 2.1.

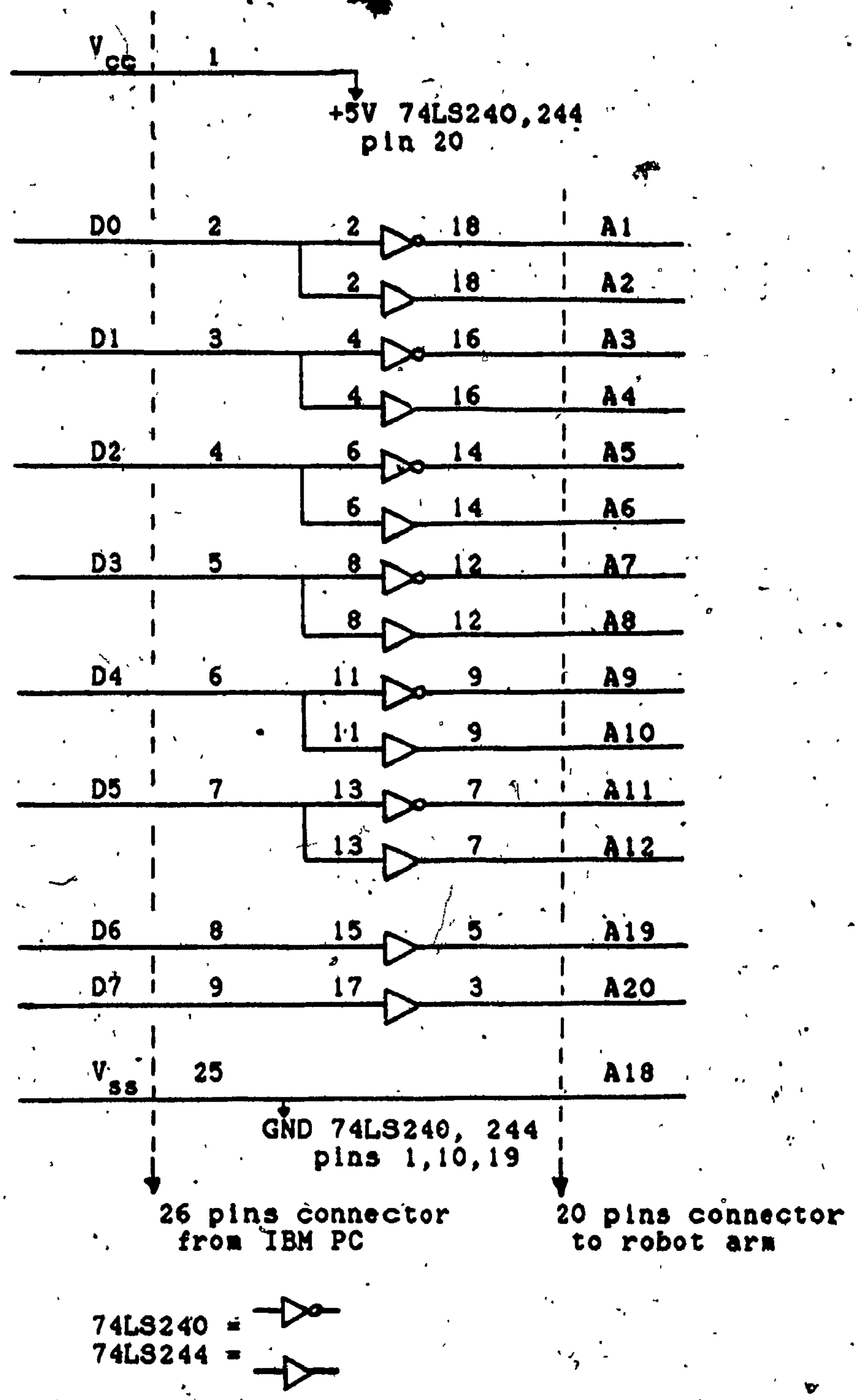


Fig 2.1 Driver/Buffer interface.

Figure 2.2 shows how the computer signals D0-D1 in combination with D7 or D6 are used to drive the HEAD (motor 1) or the EXTEND (motor 4) respectively. Similarly D2-D3 in combination with D7 or D6 drive the SHOULDER (motor 2) or the GRIPPER (motor 5) respectively; and D4-D5 in combination with D7 or D6 drive the WRIST ROTATE (motor 3) or the WRIST PIVOT (motor 6).

If we suppose that D7=1 and D6=0 (head enabled/extend disabled) and that D0=0 and D1=1, then A1 is in the logic state 1 and the transistor whose base is at A1 conducts so that coil 1 is active. Similarly in this case, coils 2 and 3 are inactive and coil 4 is active.

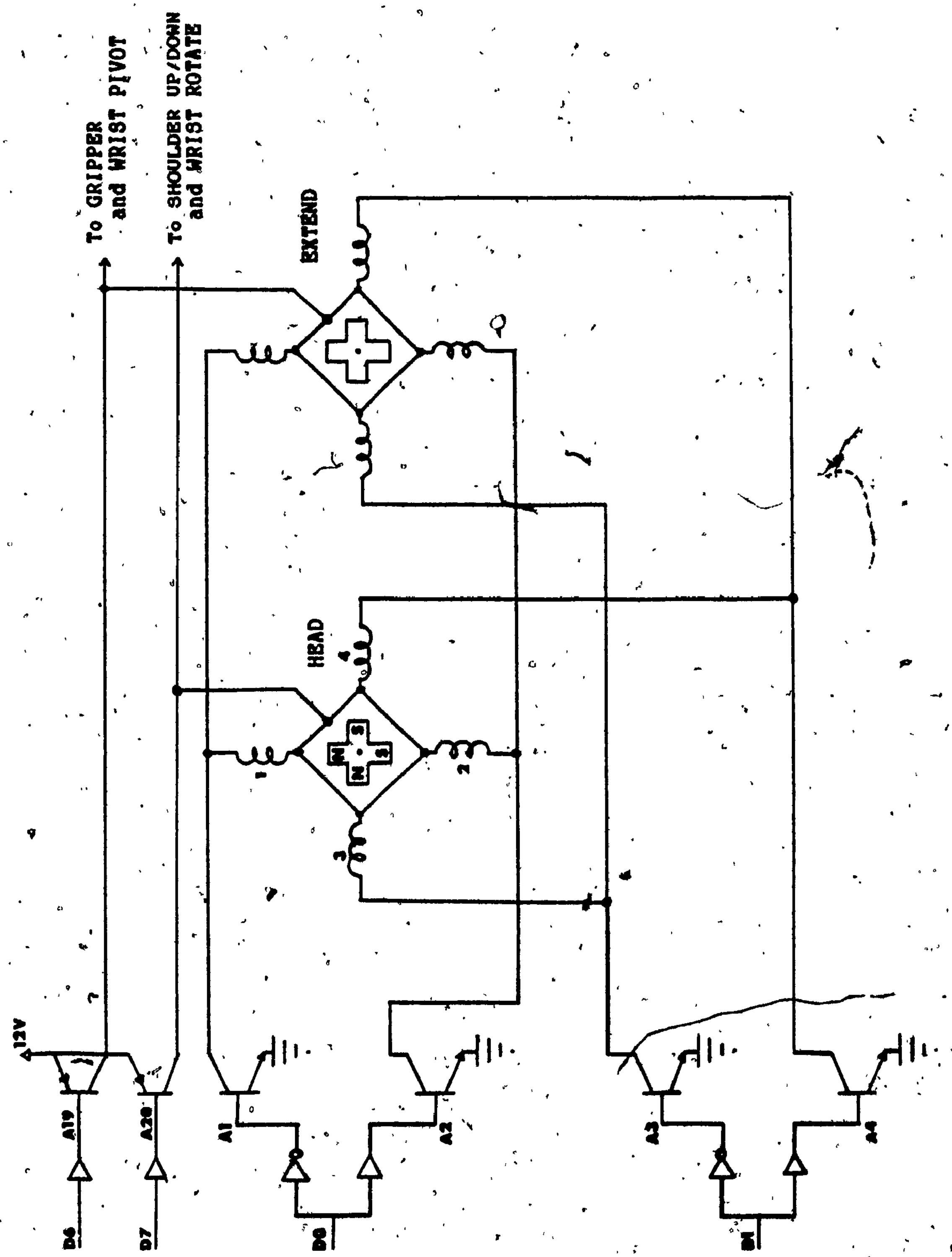


Fig 2.2 Stepper Motor Control.

2.2 Stepper motor: principle of operation

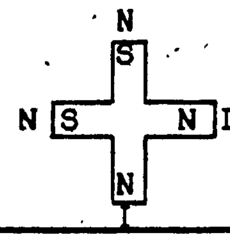
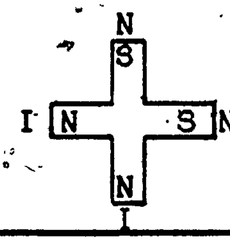
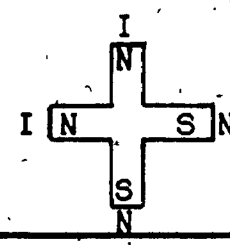
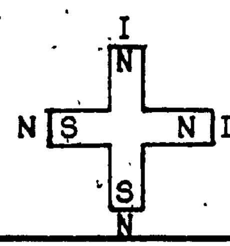
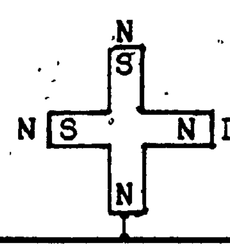
There exists many types of stepper motors: Permanent-Magnet Stepping Motor; Bifilar Stepping Motor; Variable-Reluctance Stepping Motor and Pulse Step Motor. What follows does not pretend to describe any of them in particular, it is only a simple model that explains the functionality of our stepper motors from the point of view of the programmer.

The stepper motor operates by having a set of permanent magnets attached to the rotor of the motor. A set of coils can be energized on the stator of the motor. Thus, the north and south poles of the permanent magnet on the rotor will rotate to a position where the north and south poles of the stator and rotor line up, since unlike poles attract.

Looking at figure 2.2, one sees that in our model, a motor is made of four coils on the stator and two permanent magnets on the rotor. Two opposite coils are paired so that only one is active at any one time. Let's assume that for any coil of any motor, whenever it is active, it always offers a NORTH polarity to the rotor. Furthermore suppose that coil 1 is paired with coil 2 and that coil 3 is paired with coil 4.

Table 2.1 shows how the D0-D1 computer outputs must be driven in order to make a clockwise (top to bottom in table) or a counter clockwise (bottom to top) rotation of motor 1 (A20=1) or motor 4 (A19=1). This is also applicable to lines D2-D3 to drive motors 2 or 5 and to D4-D5 to drive motors 3 or 6. One sees that for any of the four possible combination of D0-D1 there exists only one equilibrium state of the rotor.

TABLE 2.1 Control Signals for Rotor Moves.

Computer Signals D0	D1	Signals to Arm A1	A2	A3	A4	Coil Polarity N=North I=Inactive 1	2	3	4	Rotor Equilibrium State
0	0	1	0	1	0	N	I	N	I	
0	1	1	0	0	1	N	I	I	N	
1	1	0	1	0	1	I	N	I	N	
1	0	0	1	1	0	I	N	N	I	
0	0	1	0	1	0	N	I	N	I	

The period of time during which a bit pattern must be held depends upon the addressed motor. There is no maximum hold time value. The values are set from the keyboard, and then by commanding arm moves (still from the keyboard), one can determine which value seems appropriate for a given motor. The values selected range from 2 to 7 msec.

In our model, the rotor makes one quarter of a turn for each pulse, but obviously each motor is geared to obtain smaller moves. For example motor 1 requires about 20 pulses to make a 1 degree rotation of the head, and motor 4 requires about 370 pulses to make a 1 centimeter move on the extend line.

Table 2.2 summarizes the way the control to the arm must be done.

TABLE 2.2 Arm Control summary.

MOTOR #	DESCRIPTION	+DIR	-DIR
0:	ALL MOTORS DISABLED		
1:	HEAD ROTATE SEEN FROM TOP...	CW	CCW
2:	ARM TOWARD..................	FLOOR	CEILING
3:	WRIST ROTATE FACING IT.....	CW	CCW
4:	EXTEND.....................	OUT	IN
5:	GRIPPER....................	CLOSE	OPEN
6:	WRIST PIVOT TOWARD.........	FLOOR	CEILING

OUTPUT BYTE: BITS
- 7: ENABLES MOTORS 1, 2, AND 3 IF 1
- 6: ENABLES MOTORS 4, 5, AND 6 IF 1
- 5 AND 4: CONTROL MOTOR 3 IF BIT 7 = 1
  CONTROL MOTOR 6 IF BIT 6 = 1
- 3 AND 2: CONTROL MOTOR 2 IF BIT 7 = 1
  CONTROL MOTOR 5 IF BIT 6 = 1
- 1 AND 0: CONTROL MOTOR 1 IF BIT 7 = 1
  CONTROL MOTOR 4 IF BIT 6 = 1

TO OBTAIN + MOVES (FORWARD) THE FOLLOWING SEQUENCE MUST BE SENT TO THE 2 BITS CONTROLLING THE ADDRESSED MOTOR

DECIMAL	BINARY MS BIT	LS BIT
0	0	0
2	1	0
3	1	1
1	0	1

ASSUMING THAT THE 2 BITS ARE IN STATE 1 (01), THE ABOVE SEQUENCE WOULD COMMAND 4 STEPS FORWARD:

1 -> 0 -> 2 -> 3 -> 1

A PATTERN IS HELD MOTHOLD[1] msec ON MOTOR 1.
TO OBTAIN A - MOVE (BACKWARD), IN ANY STATE, THE INVERSE SEQUENCE MUST BE ASSERTED. FOR EXAMPLE

3 -> 2 -> 0 -> 1 -> 3 -> 2

WOULD COMMAND 5 STEPS BACKWARD, ASSUMING THAT THE STATE IS 3

CHAPTER 3

THE TASK AND THE PHYSICAL INSTALLATION

3.1 The task

The task may be described as follows: the system will wait for the cross to appear in the camera field; if it is considered reachable, the manipulator will move toward it, pick it, move it outside the camera field, and drop it; otherwise it will announce at the console that it is not reachable.

3.2 Physical installation and restrictions

The cross to be picked is made of wood and its dimensions are 4"X4", its height is 1" and is thickness 1/8". The cross is all black with a white spot on each of its extremities so that the camera can detect it.

The camera is installed at 89" (226cm) above a 12"X34" horizontal black plane which is a little bit larger than the field covered by the camera.

Two 4 inche fingers, each with a white spot on its extremity have been added to the original gripper. This has been done because of the restrictions on the moves of the arm; this way the finger tips can be seen by the camera. Figure 3.1 shows the original gripper with the two added fingers.

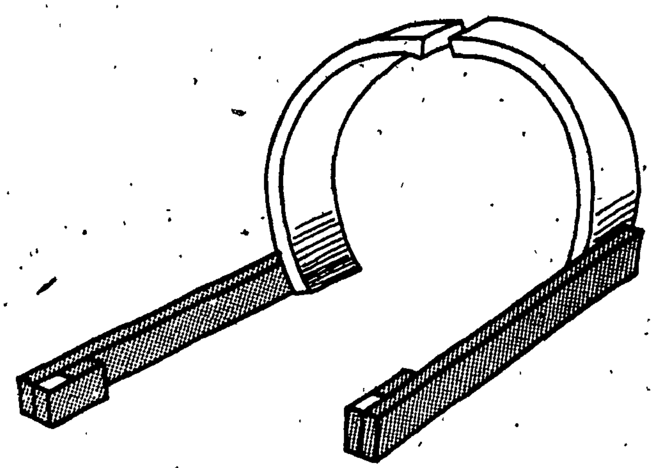


Fig 3.1 Original gripper with two added fingers.

The computations for the moves of the arm will assume that all displacements are made horizontally above the black plane. Axes of rotation for motors 1 and 3 are vertical while axis of translation for motor 4 is horizontal, giving horizontal moves only, in the three cases. There will be no moves by motor 6 since axis of rotation of motor 3 must always be vertical. As will be explained in chapter 5, motor 2 will be used to raise or lower the arm at well defined heights, from the horizontal position. Figure 3.2 shows a top view of the arm as it will be used.

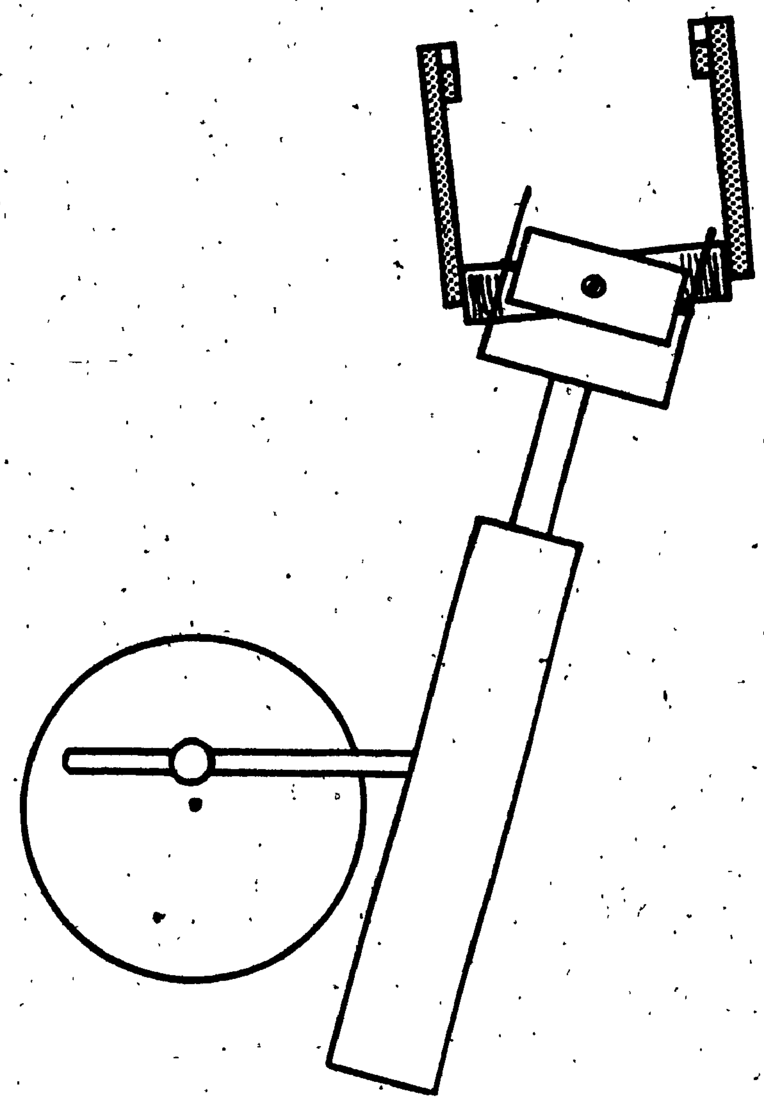


Fig 3.2 Top view of the arm.

Figures 3.3 and 3.4 show the installation with the cross on the black plane and the manipulator just about to pick it. A black cardboard has been placed on the arm in order to mask its reflecting parts. This will permit the picture to be made up of the desired white dots only. As will be seen in chapter 5, the cardboard could be removed after the system has been calibrated and initialized. At that time it will be possible for the program to localize the cross or the finger tips, even if the picture contains undesired informations.

Fig 3.3 Top view of installation.

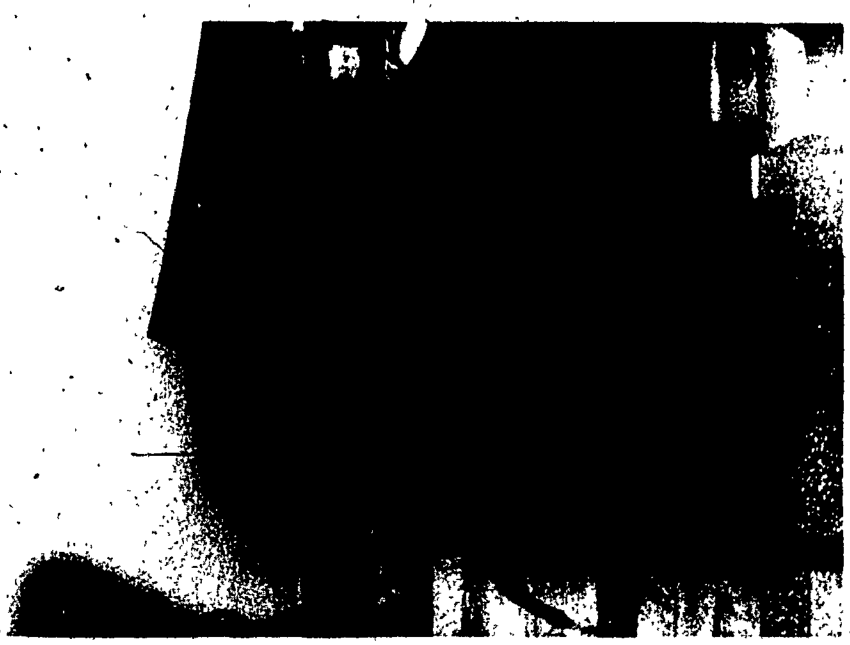


Fig 3.4 Perspective view of installation.

3.3 The computer

The system is controlled by an IBM PC compatible micro-computer (Eggebrecht, 1983) with 256K bytes of RAM; a 8088 microprocessor; no 8087 math co-processor; and two 360K bytes floppy disk drives. The development software includes the DOS 2.1 (IBM, 1983), the SBB Pascal compiler (ver 3.0) (SBB, 1984), and the IBM Macro Assembler (ver 1.0) (IBM, 1982a).

CHAPTER 4

THE CAMERA SYSTEM

4.1 Introduction

This chapter introduces the camera system used in the project (Ciarcia, 1983a, 1983b; Micromint Inc., 1983). We first start with a justification for using optic RAMs as the image sensors in the camera.

Most of the computer image-input devices currently available use a conventional black-and-white television camera as the image sensor. The camera's video output must be converted to digital logic levels for the computer: a difficult task because the output produced using a Vidicon-type pickup tube is a high frequency analog signal divided into 30 complete frames of picture information transmitted and scanned each second. Frame grabbing is the process by which one of the frames is sampled, digitized, and stored during a 1/30-second frame-scan interval.

High-speed frame grabbers generally cost more than $10,000, while the slower units (those who assume that the camera and the object in its view will remain still, long enough, for the picture to be processed by slower, cheaper circuitry) cost somewhat less, depending upon speed and resolution.

The problem with the Vidicon-type camera is that it is an analog device which must be adapted to work in digital applications. Using photodiode (or charged coupled device (CCD) arrays as image sensor one gets a computer video camera that is inherently digital and dispenses with the analog-to-digital conversion. But a 256X256 CCD array costs from $800 to $2000, depending upon the number of bad pixels you get. Photodiodes are available in 128X1 or 256X1 arrays, but arrays more than one element wide are hard to find. To create a 256X128 or 128X128 picture one would need to devise an optical, mechanical, or electronic way to move the array across the image plane, or move the image across the array, stopping periodically for the values of the picture elements to be registered and stored.

It was found that dynamic RAMs are light-sensitive. As an alternative to Vidicon camera, CCDs, or photodiodes, why not use them as image sensors? The ideal would be to have a 64K-bit dynamic RAM chip configured as a 256X256 array, but the problem, however, is that these chips were designed only as memory devices and not optical arrays. None of the 64K-bit dynamic RAMs on the market are configured as an orthogonal array laid out 256 elements long by 256 wide. In fact, most have 4 or 8 sections of 16K or 8K bits, and many include redundant sections that can be wired in to replace bad sections on the chip. The bit addresses don't proceed linearly through the chip either; one bit may be in the upper-left corner and the next bit in the lower-right corner.

A dynamic-RAM manufacturer has recognized the light-sensing potential of its 64K-bit device. Micron

Technology of Idaho, produces a dynamic RAM chip that has its memory cells laid out in only two sections, both of which are 256X128 cells. The IS32 Optic RAM comes in a package with a see-through quartz lid. Its bit-for-bit uniformity is not as good as CCDs but per pixel costs 1000 times less. This is the chip used in our camera.

4.2 The Components

The camera system used in the project is the Micro D-Cam (IBM PC Version) manufactured by Micromint inc. of Cedarhurst, NY. This low cost ($295 US) kit includes a Printed Circuit Board that connects into one of the micro-computer I/O expansion slots; a camera made of the IS32 Optic RAM mounted inside a light-tight box with a 16mm C-mount lens focussing light onto its cell arrays; a four foot ribbon cable that leads straight from the Optic RAM to the interface card in the micro-computer.

4.3 The IS32 Optic RAM

The IS32 contains 65536 light-sensitive memory cells laid out in two planar, rectangular arrays of 32768 elements, each a matrix of 128 rows and 256 columns. The two arrays are seperated by an optically nonsensitive "dead" zone of 150 microns (approximately 18 elements wide). The horizontal center to center distance between two adjacent pixels is 21.5 microns, the vertical distance is [illegible]5 microns. Figure 4.1 shows the IS32 chip's physical data.

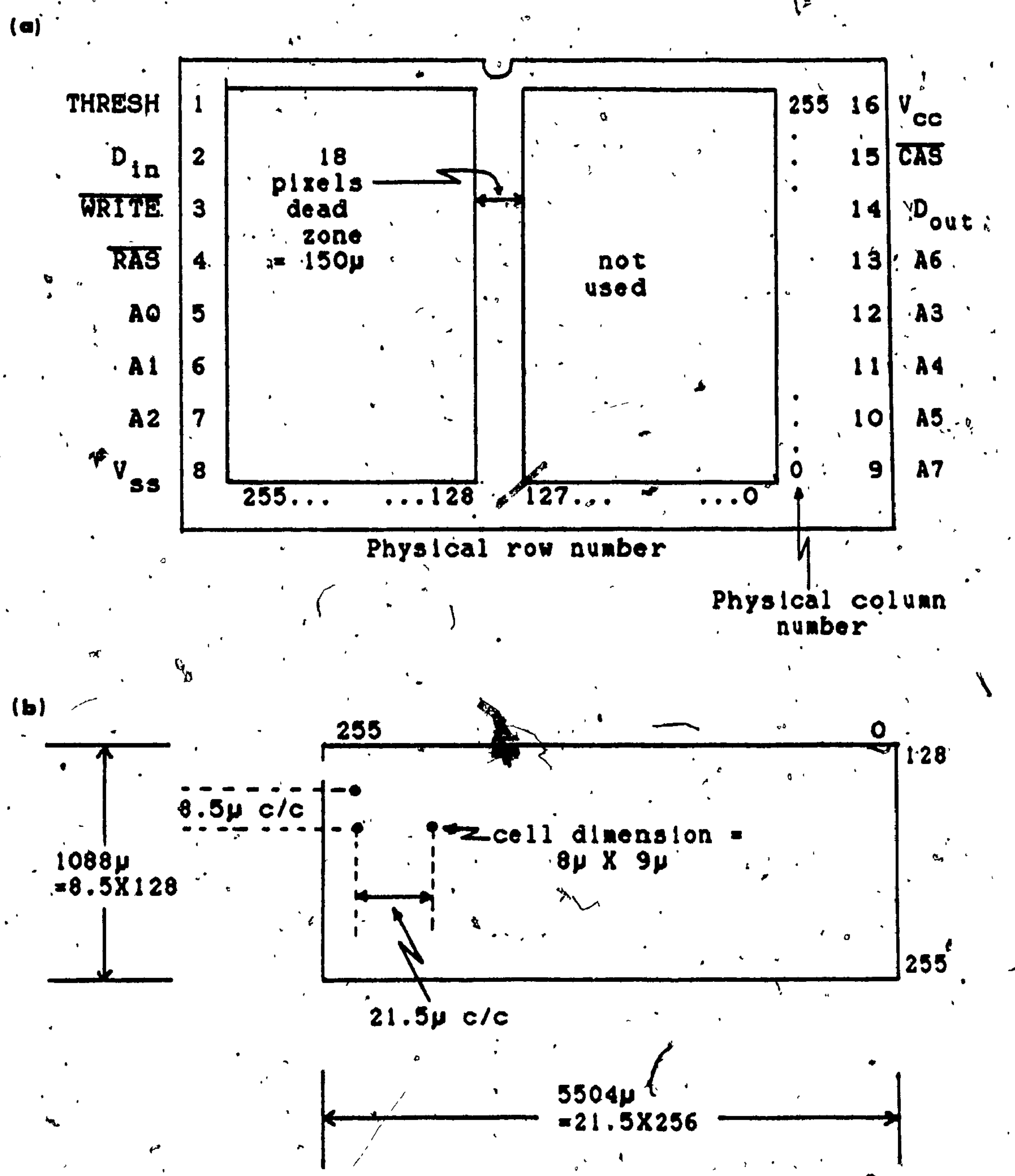


Fig 4.1 (a): IS32 pin-out with its two pixel arrays. (b): one of the two arrays.

4.4 IS32: Principle of operation

The data stored in a cell of a dynamic RAM is destroyed (the cell discharge) if not refreshed periodically. This discharge process is amplified by light: when an individual element is struck by photons of light, the capacitor in the cell, which is initially precharged to a fixed voltage, begins to discharge toward zero volts. The capacitor discharges at a rate proportional to the light intensity throughout the duration of the exposure.

Refreshing consists of sensing the charge of a cell and bringing it back to the nominal voltage for the logic state it represents. This can happen when the computer reads a bit value from the cell, but more frequently it happens when circuitry external to the memory chip periodically activates the cell's address just for that purpose. The IS32 Optic RAM like many memory chips, can refresh its cells a whole row at a time.

If one assumes that all the cells in the array are filled with the positive voltages that represent logic 1s, if refreshing is prevented during a certain period of time (exposure time), then the cells exposed to a light intensity sufficient to discharge them past the threshold point will be considered as white pixels (logic 0) and the other ones as black pixels (logic 1). After the appropriate exposure interval has elapsed, the states of the memory cells are freezed by restarting the refresh procedure. The picture or RAM data can then be transferred to the computer memory for processing. The algorithm for the image-sensing cycle of the Micro D-Cam will be described later on.

The Optic RAM resembles Film. Like a black-and-white film emulsion in a conventional photographic camera, the IS32 contains many light-sensitive elements lying in a single plane. The image is focussed (optically) on the plane, the aperture (measured in f-stops) and the length of exposure can be ajusted. In the Optic RAM, the film is "advanced", by refreshing the voltage on all the memory elements. The different gray levels obtained with a photographic film may be obtained with the optic RAM by making multiple scans of the same optical image, averaging the results obtained from either changing the sensitivity of the cells, using a different threshold voltage for each scan, or varying the scan rate (exposure time).

The nominal threshold potential, 2.1 volts (V), can be adjusted through pin 1 (Analog Threshold) on the IS32 from 1.5 to 3V, but Micron Technology suggests that gray-scale capability be achieved by varying the scan rate rather than by adjusting the threshold voltage. In the present project none of the above gray-scale capabilities are exploited. The input image is therefore stored, processed and displayed as a matrix of binary pixels.

4.5 Lenses and the IS32

The Micro D-Cam utilizes a standard 16mm mount lens with variable focus and aperture adjustment, the aperture controls the amount of light reaching the optic RAM. The focus control focuses the image onto the surface of the IS32. A mechanical shutter is not needed since this function is performed electronically by the Micro D-Cam.

Placing the lens at a distance of 226cm (89 inches which is the distance chosen) and utilizing only one of the two IS32 arrays, the rectangle covered has an horizontal length of 37.7 cm and a vertical length of 7.7cm. This represents only 9.5 degrees horizontally and only 2 degrees vertically.

These small viewing angles are due to the tiny dimension of the film, that is, the length of the diagonal of the cell array, 5.61mm.[3]

One knows that the viewing angle of a lens increases with the ratio of the diagonal of the film over the focal length (Bovis, 1973). Since one cannot increase the diagonal of the "film", the only solution is to decrease the focal length. This is what has been done.

The lens has been changed to a standard 6.5mm C-mount lens with aperture adjustment and no focus adjustment. Under the same conditions, as used with the 16mm lens (i.e.

distance 226 cm and using only one array of the IS32), the rectangle covered has now 126cm horizontally and 25.6cm vertically, giving viewing angles of 31 degrees and 6.5 degrees respectively. The fact that the 6.5mm lens had no adjustable focus (focus at infinity) was not a problem, the lens was unscrewed until the plane 226 cm apart came into focus.

4.6 The Printed Circuit Board

The camera system connects to the micro-computer via a printed circuit board that fits in one of the 62 pin I/O expansion slots (IBM, 1982b).

The interface to the micro-computer is accomplished by a 6850 Asynchronous Communication Interface Adapter (ACIA) chip which performs the serial-to-parallel and parallel-to-serial data conversion, mating the Micro D-Cam to the host computer. The ACIA is decoded at addresses 0318H and 0319H.

Five lines run between the rest of the Micro D-Cam and the ACIA, carrying the transmit, receive, ground, external clock signals, and +5V power.

Figure 4.2 shows the interface circuit from the Micro D-Cam to an IBM Personal Computer (Micromint Inc., 1983).

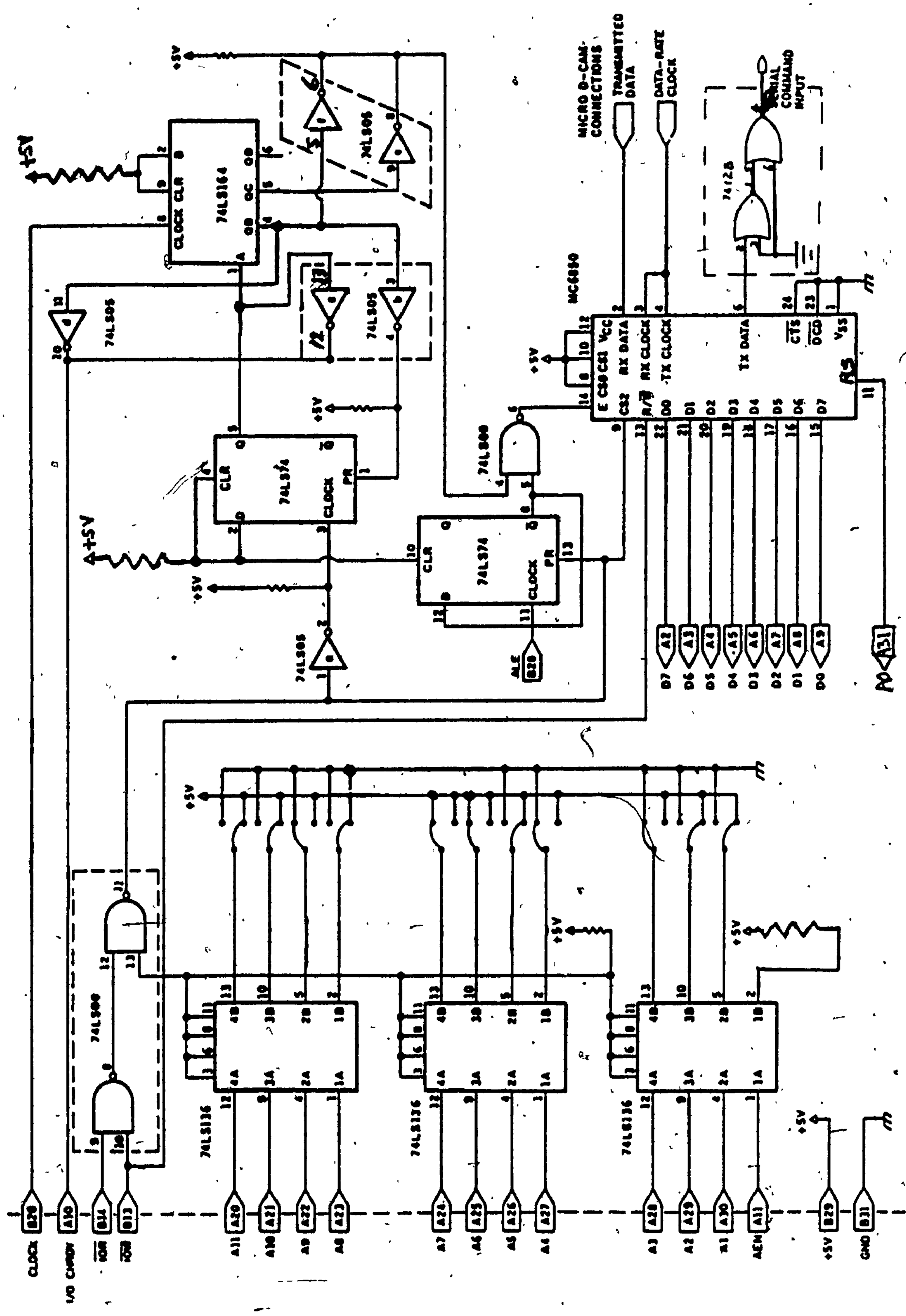


Fig 4.2 IBM PC/Camera Interface.

4.7 Programmer's Model

Before going into the details about the software to operate the Micro D-Cam, two facts must be mentioned about the physical distribution of cells in the IS32 Optic RAM.

Considering only one of the two arrays (128X256 pixels) of the IS32, one might suggest that the covered horizontal field is twice as large as the vertical one. This is not the case since horizontal pixel density is about 2.5 times less than the vertical one, giving an horizontal to vertical ratio of approximately 5. Remember from a previous section that the picture (6.5mm lens at 226 cm) was a 126cm X 25.6cm rectangle giving a 4.92 ratio. Because of this difference in horizontal and pixel densities, a non-modified picture would appear squeezed horizontally. This problem will be addressed later on.

The second fact to be considered is the way the cells are distributed. The image sensing elements in the IS32 are arranged in a "honeycomb" design so that they dont line up in the vertical direction. Here again a non-modified picture would appear with some of the pixels slightly misplaced.

The following describes the Micro D-Cam interface. It is made of 4 I/O registers:

Write only 0318H: ACIA control;

Read only 0318H: ACIA status;

Write only 0319H: Camera control;

Read only 0319H: Pixels from camera.

Note that bits 0..7 from/to ACIA are flipped to 7..0 on card. This is to accommodate the way the IBM PC stores the graphic data in its video RAM. This is shown in figure 4.2.

ACIA control register (meaning when set to 1):

- bits 0,1,4,5: not used;
- bit 2: data received at 0319H before previous byte read;
- bit 3: data received at 0319H was improperly framed;
- bit 6: a command may be sent to the camera;
- bit 7: data has been received in register 0319H.

ACIA status register:

- bit 0: reset to 0 (interrupts disabled);
- bits 1,2: reset to 0,0 disabling transmitting interrupts from ACIA to host computer;
- bits 3,4,5: control the word format. It is set to binary 101 giving 1 start bit, 8 data bits, 1 stop bit, no parity;
- bits 6,7: if 1,1 then ACIA master reset
  if 0,0 then baud rate is input frequency divided by 1.

Camera control register:

bit 0,1:	always 1;	
bit 2:	-ALTBIT/+NOALTBIT;	(1 selected)
bit 3:	-WIDEPIX/+NOWIDEPIX;	(1 selected)
bit 4:	-7BITS/+8BITS;	(1 selected)
bit 5:	-ONEARRAY/+2ARRAY;	(0 selected)
bit 6:	-REFRESH/+SOAK;	
bit 7:	-SEND/+NOSEND;	

When bit 2 is clear (0), the Micro D-Cam will transmit only the pixels from the even-numbered rows and columns in the image array. Because of the placement of the pixels in the image sensor, this mode will usually produce an image of clearer resolution than the NOALTBIT mode at the expense of utilizing fewer pixels in the array.

When bit 3 is clear (0), the Micro D-Cam will "double transmit" each pixel in the array. Each image sensing elements is rectangular in shape. By "double transmitting", the proper width to height ratio is maintained when displaying an image.

When bit 4 is clear (0), the Micro D-Cam transmits data so that it is compatible with APPLE's high resolution format.

When bit 5 is clear (0), the Micro D-Cam will only transmit one of the two 128X256 pixels arrays (rows 128..255).

When bit 6 is clear (0), the Micro D-Cam will refresh the IS32 Optic RAM meaning no exposure to light.

When bit 7 is clear (0), the Micro D-Cam will send the picture (data from IS32) to the ACIA data register (0319H), which can be read by the program 8 pixels at a time.

4.8 Picture Format

Because the NOALTBIT, NOWIDEPIX, 8BITS and 2ARRAY features are selected, one must apply an ENHANCE algorithm to the raw 128X256 pixels obtained from the camera for the reasons mentioned in section 4.7. The algorithm (written in 8088 assembly language and called from the Pascal program) transforms the middle 160 columns of the 128X256 picture into a 128X640 picture. There are two reasons for discarding columns 0..47 and 208..255 of the source frame. First, because of screen limitation (200 lines X640 columns) but also because—640 horizontal pixels cover a field of about 79 cm (at distance 226 cm using the 6.5mm lens), which is sufficient for the application since the robot arm itself covers less than that.

The order (facing the picture to be taken) of pixel reading is in the natural order from left to right and then from top to bottom. This is illustrated in figure 4.3.

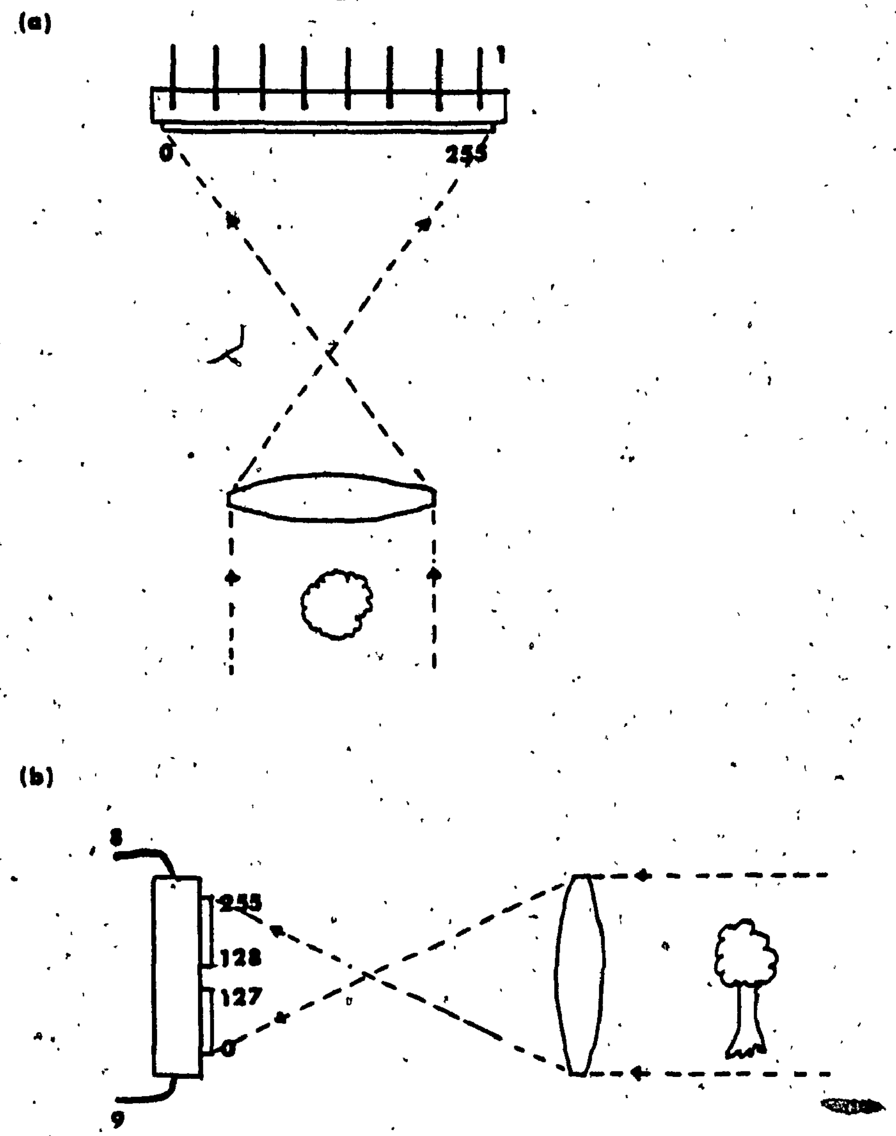


Fig 4.3 The IS32, the lens, and the object. (a): top view, (b): side view.

The first bit to be input is the bit at row 128 column 0 (recall that the other array, rows 0 to 127, is not used) the last one is bit at row 255 column 255. The first bit correspond to the topmost, leftmost corner of a scene as seen by someone behind the camera. This bit arrives in bit 0 (of ACIA register) of the first input byte of the picture. But the IBM PC way to display a byte on the screen is from bit 7 to bit 0 going from left to right on the screen. That is why the data bits from the ACIA to the computer bus have been flipped so that bits 7..0 sent by the ACIA are received by the computer as bits 0..7.

4.9 Frame grabbing rate

The time to obtain a frame for processing or display is the sum of the exposure time, the transmit time and the time to enhance the picture. The exposure time is of the order of a second under good lighting and the time to enhance is considered negligible. The transmission time depends upon the number of bits to be transmitted and upon the selected baud rate. One has 128X256 bits = 4096 bytes but each transmitted byte has one start and one stop bit giving a total of 40960 bits. The transmission rate is 153600 bits/second giving a total transmission time of 0.27 second.

4.10 Assembler Subprograms

Some subprograms have been implemented in 8088 assembly language and are accessible from the Pascal main program. FRGRAB is given the address of a 128X32 bytes variable, the desired exposure time and the physical address of the camera port, it reads a frame from the camera and returns it in the array variable. ENHANCE2 is given as input, the address of a 128X32 bytes variable and as output, the address of a 128X80 bytes variable where it returns the modified

128X640 pixels frame ready for display and/or processing. MOVESCR is given the address of a 128X80 bytes variable and it displays it on the computer screen.

Here is the algorithm describing FRGRAB.

step1: initialize ACIA for a master reset (0318H := C0H) and then for parity, stop bit, data bits, start bit and baud rate (0318H := 28H);

step2: specify the picture format, NOSEND, REFRESH, ONEARRAY, 8BITS, NOWIDEPIX (0319H:= 9FH);

step3: command exposure to light, that is NOSEND and SOAK (0319H := DFH);

step4: delay the specified exposure time (EXPTIME);

step5: command SEND and REFRESH (0319H := 1FH);

step6: read and store 128X32=4096 bytes from port 0319H, each time testing bit 7 of the ACIA status register (receiver full) at port 0318H. There is a time out counter implemented here: if 15 consecutive tests of the "receiver full" bit fail, then this indicates the end of transmission;

step7: specify NOSEND and REFRESH (0319H := 9FH), ready for next exposure;

step8: return to caller. If a key has been typed during the process, its value is returned. The number of bytes read is also returned for purpose of verification.

Note that the above procedure works only if we assume that at the beginning, all cells are in their high voltage state (logic 1). This is not the case for the first call after power up, but, it is afterward because the Micro D-Cam circuitry ensures that a logic 1 is written back into the IS32 cell after this cell has been read.

Figure 4.4 is a printout of a picture taken by the camera. It shows the 4 dots corresponding to the cross and the two dots of the finger tips around one of its branch. As we can see, even after the enhance algorithm has been applied to the raw picture, the image still appears squeezed horizontally.

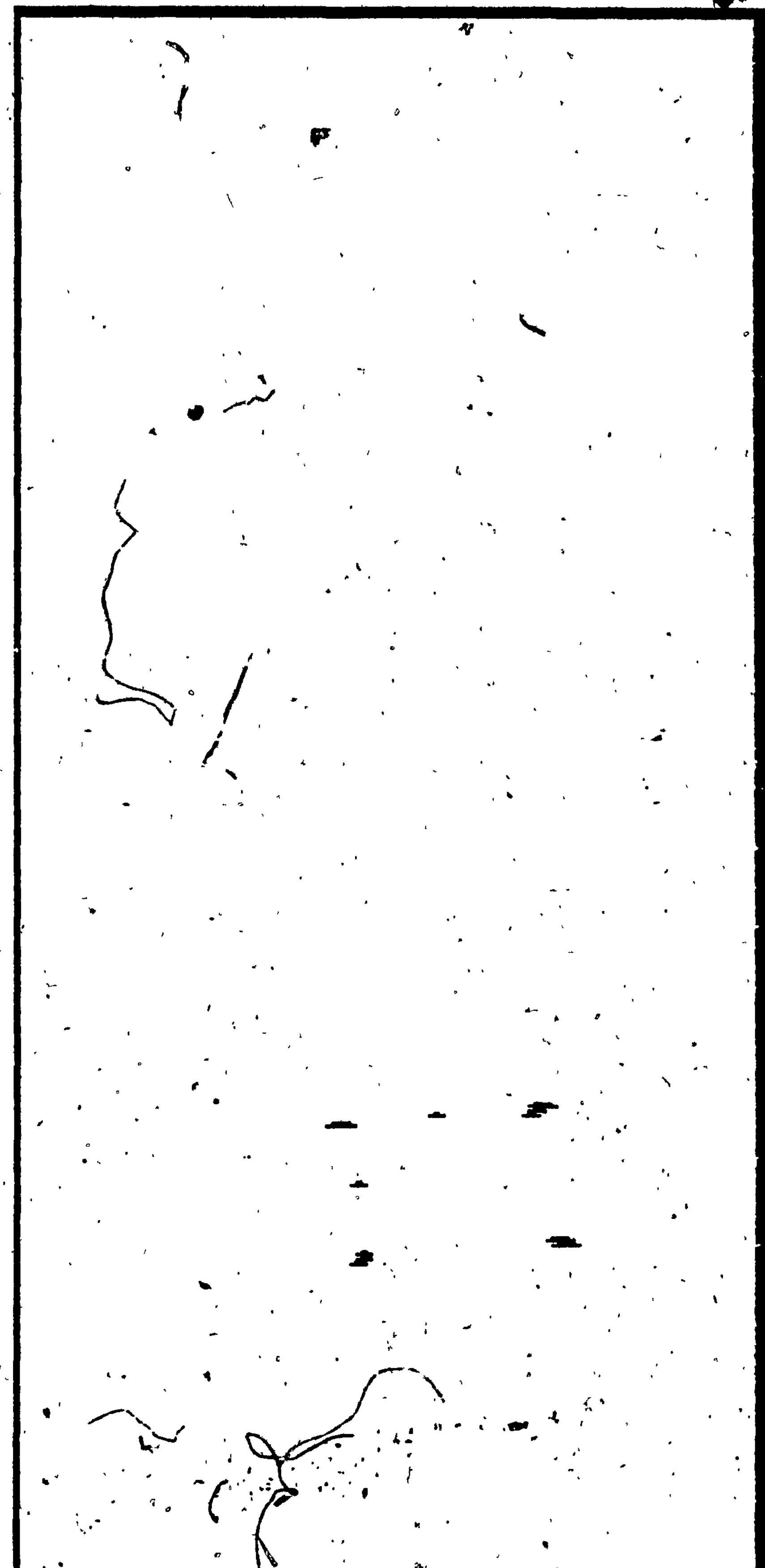


Fig 4.4 Picture from camera showing the cross and the fingertips.

CHAPTER 5

THE SOFTWARE

5.1 Introduction

The control software is divided into two parts: a Pascal main program which implements the user interface and most of the robot arm/camera system control logic; a set of Pascal accessible 8088 assembly subroutines which take care of the low level interface to the camera system I/O, the robot arm, the screen output, and the keyboard input.

5.2 Assembler Subroutines

As described in the camera section, subroutine FRGRAB reads a 128X256 pixels frame from the camera and stores it in a 128X32 bytes variable. Subroutine ENHANCE2 modifies this raw picture into a 128X640 picture.

In graphic mode, the IBM PC video controller displays 640 columns by 200 rows images. Subroutine MOVESCR is responsible for displaying the picture obtained from ENHANCE2 by writing it in the top part of the video RAM (address B8000H). The picture therefore occupies the top 16 lines (1 line = 8 rows), leaving 9 lines for the user communication area. This area is cleared by a call to subroutine CLRCOMM.

Calls can be made to the DOS or to the ROM BIOS by subroutines CALLDOS and CALLBIOS respectively. As an example, CALLBIOS may be used to position the cursor at the top left corner of the communication area.

Subroutine BLINK is given the column (0..639) and the row (0..127) coordinates of a pixel, it will make it blink until a key is pressed at the keyboard.

Subroutine GMODE sets the display mode (graphic, alphanumeric etc...) and returns the previous setting.

Subroutine ARMOUT outputs a byte to the parallel port (0378H) controlling the robot arm. It is also given the time the pattern must be held and returns a key value if any was typed during the process.

Subroutine DETDOTS is given the address of a 128X80 bytes picture and returns an array of dot coordinates. A dot coordinate is a pair (col,row) with col: 0..639 and row: 0..127. The last dot in the array is followed by the (-1,-1) dot.

In this project, each picture is considered as a set of dots. The cross to be picked as four white spots on it and each finger of the manipulator has one.

A dot may be considered as a cluster of white pixels surrounded by four segments (a rectangle) of black pixels.

A dot is detected as follows: the frame is scanned from left to right and from top to bottom until a white pixel is detected. Starting at that pixel, rectangular paths are then tested until a path of black pixels is found. This is shown in figure 5.1.

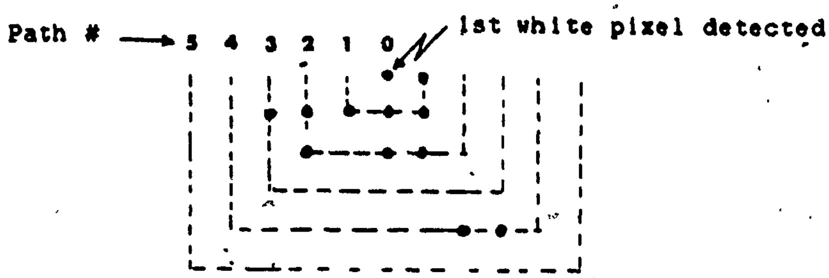


Fig 5.1 A cluster of pixels.

Path number five of figure 5.1 satisfies these conditions. Note that if part of a path extends outside the picture (dot near boundary) that part of the path is considered by the algorithm as being made of black pixels only. The last non-black path starting from path 0 is defined as the current outlying path (path 4 in figure 5.1). The leftmost vertical segment of the outlying path is defined by the column coordinate LMOST, the rightmost segment by RMOST. The topmost horizontal segment is defined by the row coodinate TMOST and the bottommost segment by BMOST.

The rectangle defined by the above four coordinates is then refined: vertical segments between the horizontal segments TMOST and BMOST are then tested starting from the inner path and going left until the last non-black segment is found. The column number corresponding to this non-black segment is then assigned to LMOST. The same process is applied for vertical segments between TMOST and BMOST starting at the innermost path and going right. A new value is then possibly assigned to RMOST. Horizontal segments are then tested between the two vertical segments LMOST and RMOST, starting from the top and going down. The last non-black segment defines the new value for BMOST.

The whole process is repeated until the new values of RMOST, LMOST, BMOST and TMOST equal the old ones. That is, when no changes have occured in the last processing.

Applying this algorithm to the dot of figure 5.1 one would obtain the final outlying path shown in figure 5.2.

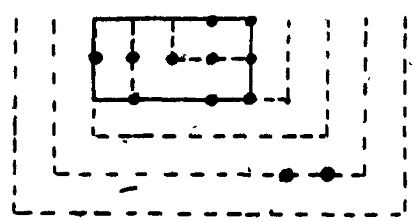


Fig 5.2 Dot determined from pixel cluster.

The column coordinate of the dot is returned as the middle column between LMOST and RMOST, and the row coordinate has the middle row between TMOST and BMOST. These coordinates are then saved and the pixels in the rectangle are made black so that the search for the next dot will not find the same dot again.

5.3 The Pascal Program

Upon entering the Pascal program the computer is in a mode where pictures are continuously taken and displayed on the screen. This process may be terminated by typing a key at the key board. The user may save the current picture on disk by typing 'S' or may load/print a picture from disk ('L') or may change the camera expsosure time ('T') or may exit the display mode and enter the main procedure (ARMMOVE) of the program ('A').

In the above procedure, one can activate the arm by typing keys 1,2,...,6 for + direction moves or the same shifted keys for - direction moves. Key numbers correspond to motor numbers as described before. The motor is stopped by typing the 'S' key which has also the effect of taking a picture and displaying it on the screen.

Arm control is made throughout calls to procedure MOVEARM. This procedure accepts a motor number and a number of steps (+or-) as arguments. It uses and updates global variables to remember the state (MSTATE[i]: 0..3) of each motor (to determine which bit pattern to apply next); the number of pulses each motor has received (MPULSE[i]) since last initialization (procedure INITMOT); and the pulse hold time (MOTHOLD[i]) for each motor. If a key is typed during the activation process, it returns its ASCII value to the caller. The actual control to the arm interface is done by a call to assembler subroutine ARMOUT.

By using the 'H' command to set the pulse hold time for each motor and by experimenting arm moves (keys 1,2,...,6) one can establish optimal hold time values.

The rest of the commands of procedure ARMMOVE will be described in what follows, when pertinent to the discussion.

Before procedure ARMMOVE can accomplish the task of identifying the cross and picking it, (command 'X') certain parameters must be initialized and stored on disk. This must be done only when the relative positions of the base of the arm and the camera have changed that is when installing the system. Some of the parameters are computed by the program, moving the arm and using the data obtained from the camera, others are entered manually from the keyboard (like the pulse hold time above).

5.3.1 Calibrating the Camera

Having installed the camera at a certain height (226 cm here) above the black plane, one must make the mapping between the pixels representation and the real world representation (plane coordinates). We must obtain the distance (in the real world) covered by two adjacent pixels. Since the horizontal and vertical pixel densities are not the same we want two correction factors CORX and CORY, which give the number of cm covered by two adjacent pixels horizontally and vertically respectively.

If we place two white dots (figure 5.3) on the black plane, at known distance D apart, and we then take a picture and obtain the ROW and COL coordinates for each dot we may then write:

$$D^2 = [CORX*(COL_2-COL_1)]^2 + [CORY*(ROW_2-ROW_1)]^2$$

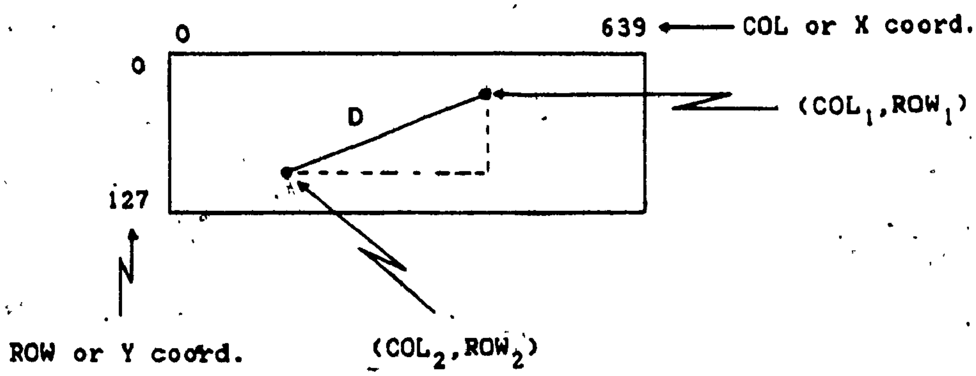


Fig 5.3 Picture with two dots at known distance D apart.

But there is an infinity of solutions for the unknowns CORX and CORY. The way to proceed is to have three dots with known distances D_{12} between dots 1 and 2 and D_{13} between dots 1 and 3, as shown in figure 5.4.

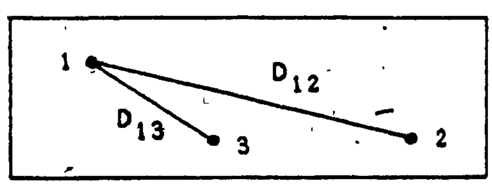


Fig 5.4 Picture with 3 dots.

We may then write

$$D_{12}^2 = [CORX*(COL_2-COL_1)]^2 + [CORY*(ROW_2-ROW_1)]^2$$
$$D_{13}^2 = [CORX*(COL_3-COL_1)]^2 + [CORY*(ROW_3-ROW_1)]^2$$

These two equations may be rewritten as two straight line equations in the form

$$CORY = m_1 \; CORX + b_1$$

$$CORY = m_2 \; CORX + b_2$$

where

$$m_1 = -[(COL_2-COL_1)/(ROW_2-ROW_1)]^2$$

$$m_2 = -[(COL_3-COL_1)/(ROW_3-ROW_1)]^2$$

$$b_1 = [D_{12}/(ROW_2-ROW_1)]^2$$

$$b_2 = [D_{13}/(ROW_3-ROW_1)]^2$$

With ROW_2 <> ROW_1 and ROW_3 <> ROW_1 and (COL_1 <> COL_2 or COL_1 <> COL_3).

Solving this simultaneously for CORX and CORY, one obtains

$$CORX = SQRT[(b_2-b_1)/(m_1-m_2)]$$

$$CORY = SQRT[(m_1*CORX^2)+b_1]$$

In practice, this is done using the 'C' command of procedure ARMMOVE. The CALIB procedure is called, and it asks the user to place a template on the black plane, the picture is then continuously displayed on the screen (calls to FRGRAB, ENHANCE2 and MOVESCR) and when the user is satisfied with it, only then, are the three dots visible on the screen. He then types the return key which freezes the frame. The procedure then makes a call to subroutine DETDOTS to obtain the ROW/COL coordinates of the three dots. Since it knows the distance between dots 1 and 2 (71.0 cm) and the distance between dots 1 and 3 (37.0 cm) it can compute and display the values of the global variables CORX and CORY.

From now on every dot coordinate COL/ROW obtained by subroutine DETDOTS may be expressed in real world coordinates by multiplying them by CORX and -CORY respectively. This is shown in figure 5.5.

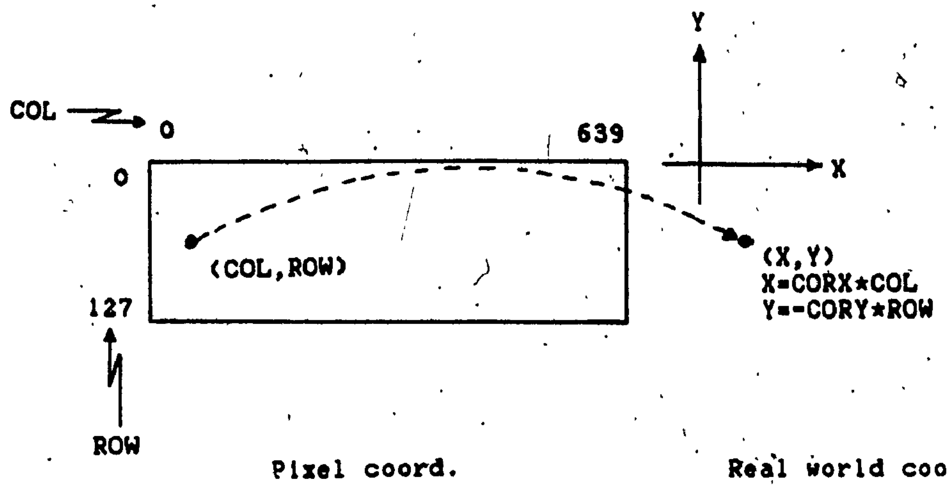


Fig 5.5 Mapping of coordinate systems.

Experimentally we obtained the following values, placing the camera at 226cm above the black plane

CORX = 0.12 cm/pixel

CORY = 0.20 cm/pixel

giving an horizontal coverage of CORX*639 = 76.7cm and a vertical coverage of CORY*127 = 25.4cm.

5.3.2 Center of Rotation of the HEAD

The center of rotation of the HEAD of the manipulator must be localised in the XY coordinates system defined above. Note that with our physical set-up this point is outside the viewing field of the camera.

If we can have the system take three different pictures each of which corresponding to a different position of the white tips of the fingers by moving on motor 1 only, one would obtain three dots lying on a circle whose center is the desired value as shown in figure 5.6.

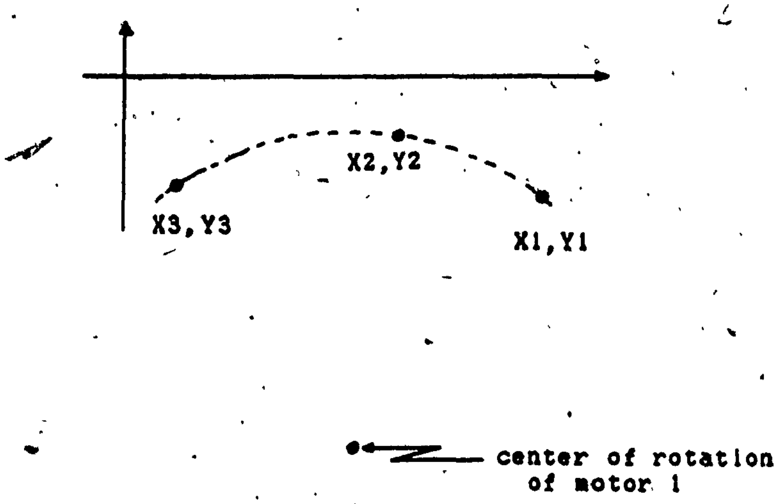


Fig 5.6 Three different positions.

One knows that three dots in a plane are sufficient to define a circle. The three-point form equation is given by the following determinant equation (Selby, 1975):

$$|M| = \begin{vmatrix} x^2+y^2 & x & y & 1 \\ x_1^2+y_1^2 & x_1 & y_1 & 1 \\ x_2^2+y_2^2 & x_2 & y_2 & 1 \\ x_3^2+y_3^2 & x_3 & y_3 & 1 \end{vmatrix}$$

The general form of the circle equation is:

$$x^2 + y^2 + 2dx + 2ey + f = 0$$

where the center is at (-d,-e) and where the radius r =

SQRT(d^2+e^2-f). To obtain the center and the radius, we must transform the three-point form equation into the general form equation. The determinant equation can be rewritten as (Laplace development):

$$|M| = (x^2+y^2)\ cof_{11}(M) + x\ cof_{12}(M) + y\ cof_{13}(M) + cof_{14}(M)$$

The cofactor $cof_{ij}(M)$ is defined as the determinant of a matrix obtained by striking the i'th row and the j'th column of m and choosing positive (negative) sign if i+j is even (odd).

By some algeabric manipulations, the above equation can be rewritten in the general form, giving

$$d = cof_{12}(M)\ /\ [2cof_{11}(M)]$$

$$e = cof_{13}(M)\ /\ [2cof_{11}(M)]$$

$$f = cof_{14}(M)\ /\ cof_{11}(M)$$

These values are then used in computing the center and the radius of the circle.

In practice this is accomplished by using the 'E' command of procedure ARMMOVE. But before using this command, one must register the three dots using the '.' command. To do so, one must first issue the 'I' command to call procedure INITMOT which initializes to zero the global variable which counts the number of pulses each motor as received (MPULSE[i]) and which places all motors in state 0 (out of 0..3), updating the corresponding global variable (MSTATE[i]). The 'I' command also initializes the counter for use with the '.' command. The '.' command localizes a dot in the current displayed picture by calling DETDOTS. It then translates the obtained COL/ROW coordinates into the current X/Y coordinates:

X := CORX * COL; Y := -CORY * ROW;

another transformation is then applied (procedure TRANSLAT) which transforms the camera X/Y coordinates into the working base coordinate

X := X - XO; Y := Y - YO;

This has no effects now since XO=YO=0 but we will later see that the origin of the working coordinate system (XO,YO) <> (0,0) will be the center of rotation of the head (motor 1).

These coordinates are then stored into an array of dots. The address in the array where to store the dot is given by the dot counter initialized in the 'I' command. The dot counter is then updated. Finally this command makes the center pixel of the dot blink, to show that the dot was properly localized. To exit the command the user must type the return key.

The user therefore first move the arm so that the tip of the fingers fall into the camera field. This is done by using the '1' key and then the 'S' key to stop all motors and take, and display the picture. The 'I' command is then issued to initialize the dot counter and pulse counters to zero. The first dot is registered by the '.' command, then the arm is moved on motor 1 only using the '1' command and then stopped by using the 'S' command. The second and third dots are also registered in this manner. The circle parameters can now be computed by typing the 'E' command which calls procedure CIRCLE. This procedure accepts the coordinates of 3 dots as input and returns as output the center of the circle and its radius (irrelevant here). The 'E' code will also compute the angle covered between dots 1 and 3 (figure 5.7).

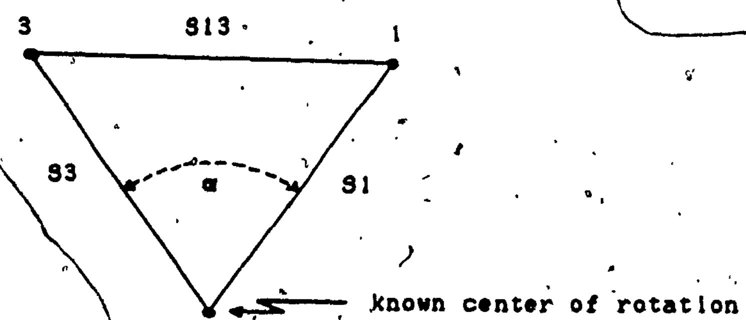


Fig 5.7 Determining the angle covered.

From the law of cosines

$$S13^2 = S1^2 + S3^2 - 2\ S1\ S3\ \cos\alpha$$

which is solved for α since S1, S3, and S13 are known. This angle information is used in computing the Number of Impulse per Degree on Motor 1 (NIDM1) since the number of pulses applied to move from dot 1 to dot 3 is known (variable MPULSE[1]). Remember that the 'I' command inititialize the global variables "pulse counter" to zero for each motor and that the '1' to '6' commands move motors by calling MOVEARM which updates the variables.

The value found for NIDM1 is 20.4 pulses/degree. The values for XO,YO depends on the relative positioning of the arm and the camera, in one installation it was found to be X0 = 38.03cm and Y0 = -47.63cm.

From now on, the new working coordinate system will have origin (X0,Y0) at the center of rotation for motor 1. The x axis will be parallel to the pixel rows and the y axis to the pixel columns. This is illustrated in figure 5.8.

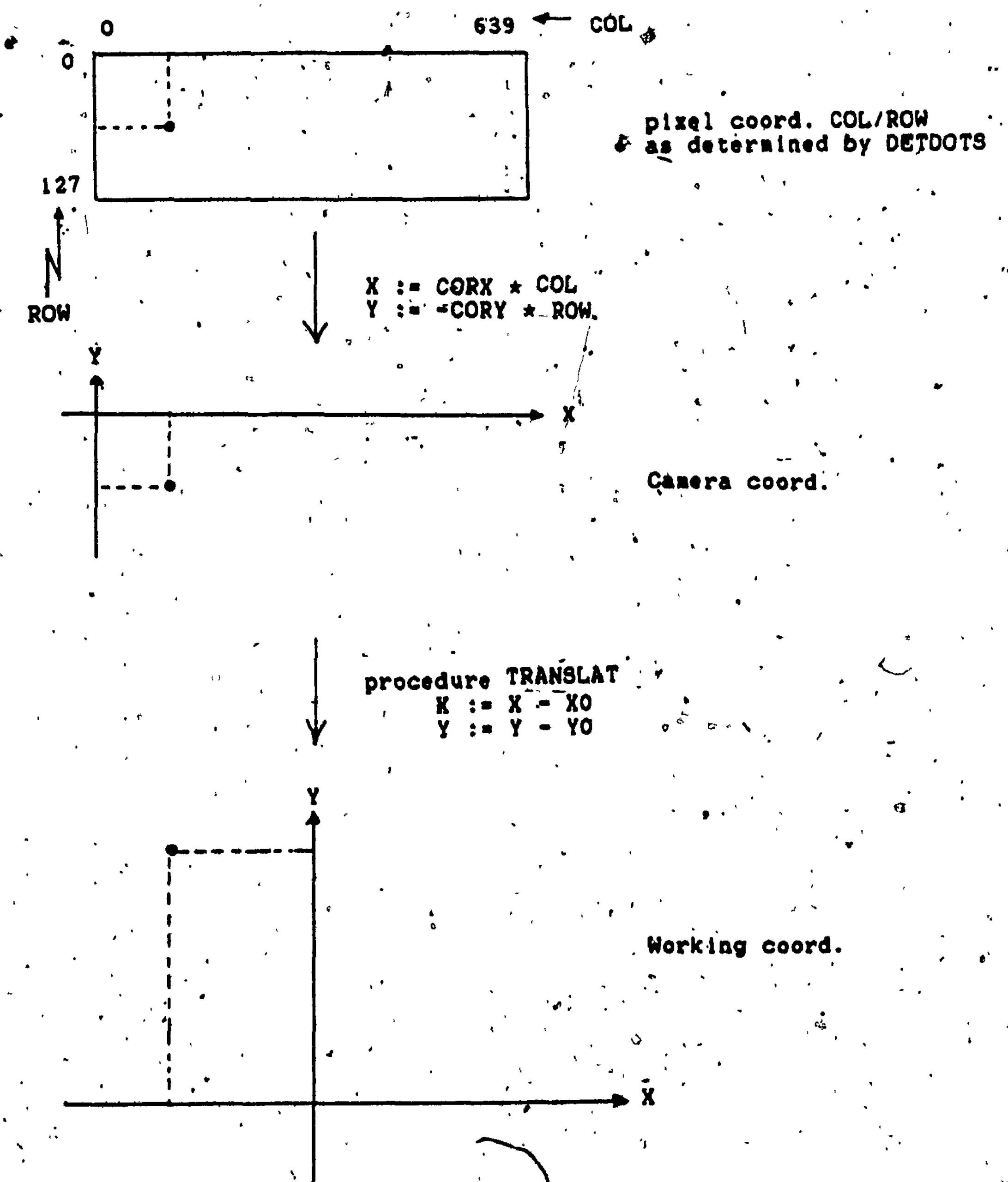


Fig 5.8 Mappings of coordinate systems.

5.3.3 Obtaining RAD3

When motor 3 is activated, the finger tips describe a circle whose radius is RAD3, this is the value we want to determine here. We also want the Number of Impulse per Degree on Motor 3 (NIDM3). To make it, we as usual use the 'I' command to initialize the counters, and then we register 3 different dots using the '.' command. Between each registration the finger tips are moved using the '3' command. Once this is done, the 'E' command is issued. It is the same command as the one we used previously to compute X0,Y0, and NIDM1. It will use the three registered dots to compute the radius RAD3 of the circle, and the global variable NIDM3.

The values obtained are 10.03cm for RAD3 and 3.31 pulses/degree for NIDM3.

5.3.4 Determining the SHOULDER length

Another physical data that must be obtained is the shoulder length of the arm. This length is intrinsic to the manipulator and does not depend on the relative positioning of the arm and the camera. It is defined as the square of the normal distance between the axis of rotation of the head (the just defined origin) and the straight line defined by the EXTEND (motor 4) movement. This is shown in figure 5.9.

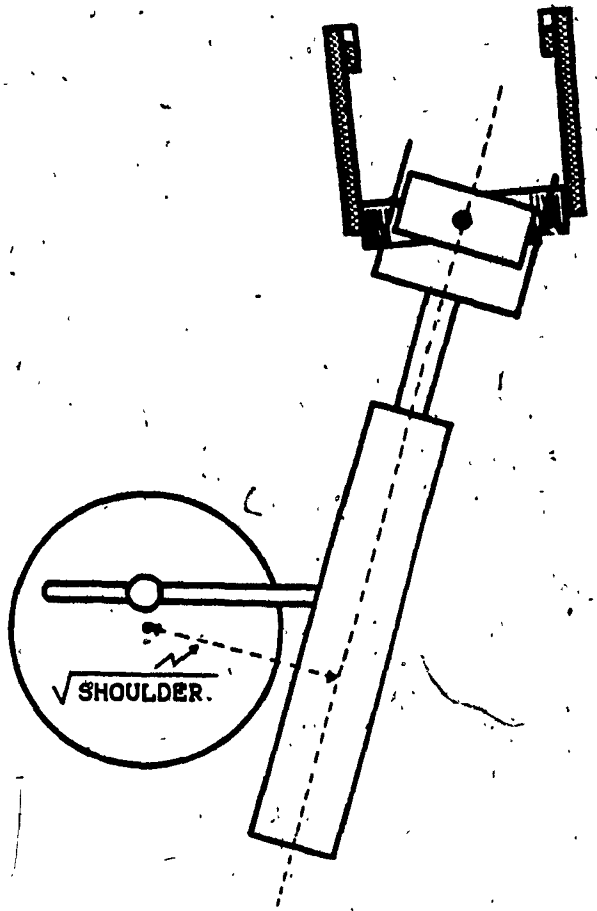


Fig 5.9 Definition of parameter SHOULDER.

If using the arm control commands and camera, we register two dots along the EXTEND line, then we may obtain the straight line (l_e) equation and compute the desired distance along l_s (figure 5.10).

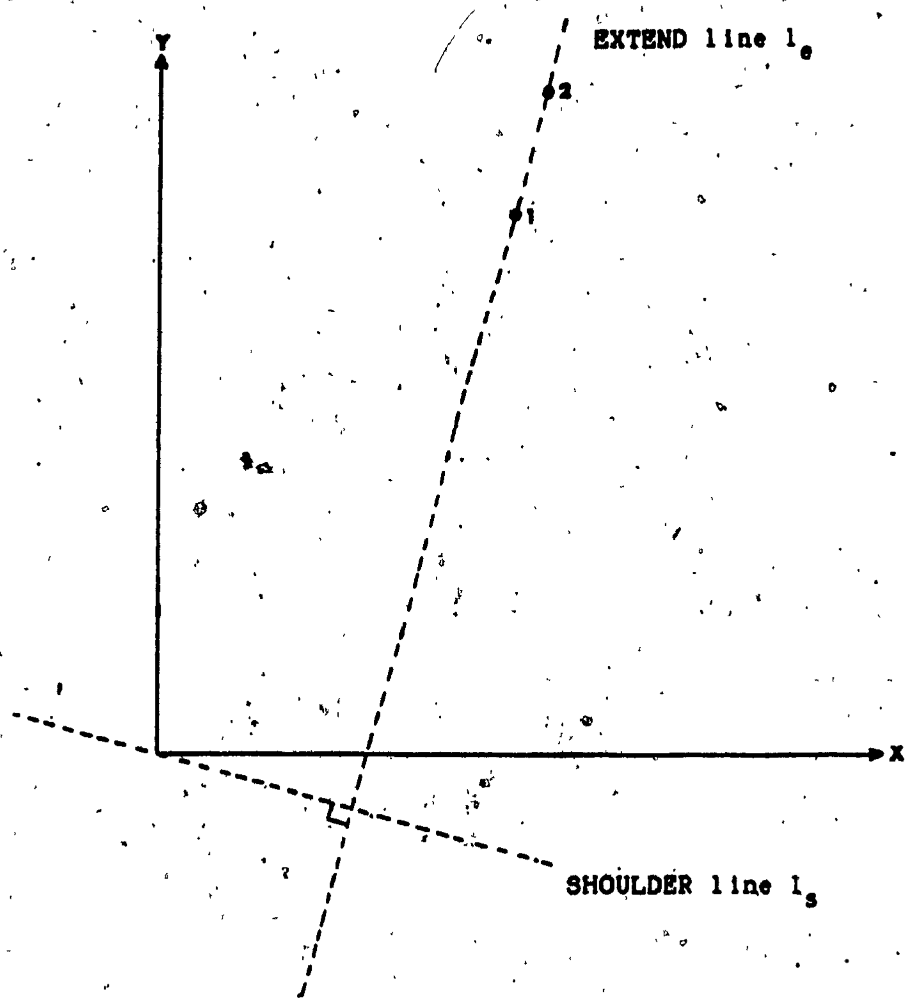


Fig 5.10 Determining the value of SHOULDER.

The equation for l_e is $y = m_e x + b_e$, where

$$m_e = (y_2 - y_1)/(x_2 - x_1)$$

$$b_e = y_1 - m_e x_1$$

To determine the equation for l_s we know that l_e is perpendicular to l_s and that the origin belongs to l_s. The slope of l_s is therefore $m_s = -1/m_e$ and $b_s = 0$, giving the equation for l_s

$$y = (-1/m_e)\ x$$

We can then calculate the point of intersection (x_I, y_I) between l_e and l_s and compute its distance to the origin, obtaining the desired value. This point is the simultaneous solution to equations for l_e and l_s

$$x_I = (-m_e b_e) / (1 + m_e^2)$$

$$y_I = -x_I / m_e$$

and SHOULDER is given by

$$SHOULDER = x_I^2 + y_I^2$$

In practice, one closes the gripper and aligns the fingers with the EXTEND line using commands '5' and '3' respectively. The 'I' command is issued and the position of the finger tips is registered ('.' command). The finger tips are then moved along the EXTEND line ('4' command) and the second dot is registered. The actual command to compute the SHOULDER global variable is '\'. This command will compute SHOULDER from the two registered dots and will also compute the Number of Impulse per Centimeter on Motor 4 (NICM4) since it can calculate the distance between the two dots and since it knows from global variable MPULSE[4], the number of pulses that were required to move from dot 1 to dot 2.

The value found for NIDM4 was 371.1 pulses/cm and the value for SHOULDER was 192.0 cm^2.

5.3.5 Initial Position of the Arm

With our restriction of the arm almost always moving in a plane, its position may be uniquely determined by three numbers: the X and Y coordinates CRW of the center of rotation of the wrist (motor 3) and WRISTANG which is defined as the angle between the segment joining the origin (XO,YO) to point CRW and the segment joining CRW to the closed finger tips (figure 5.11). When the fingers are opened, the middle point between the two white dots on the finger tips is to be considered in computing the wrist angle. Note that this way of determining the position of the arm does not consider its height as controlled by motor 2 and the finger (gripper) opening as controlled by motor 5.

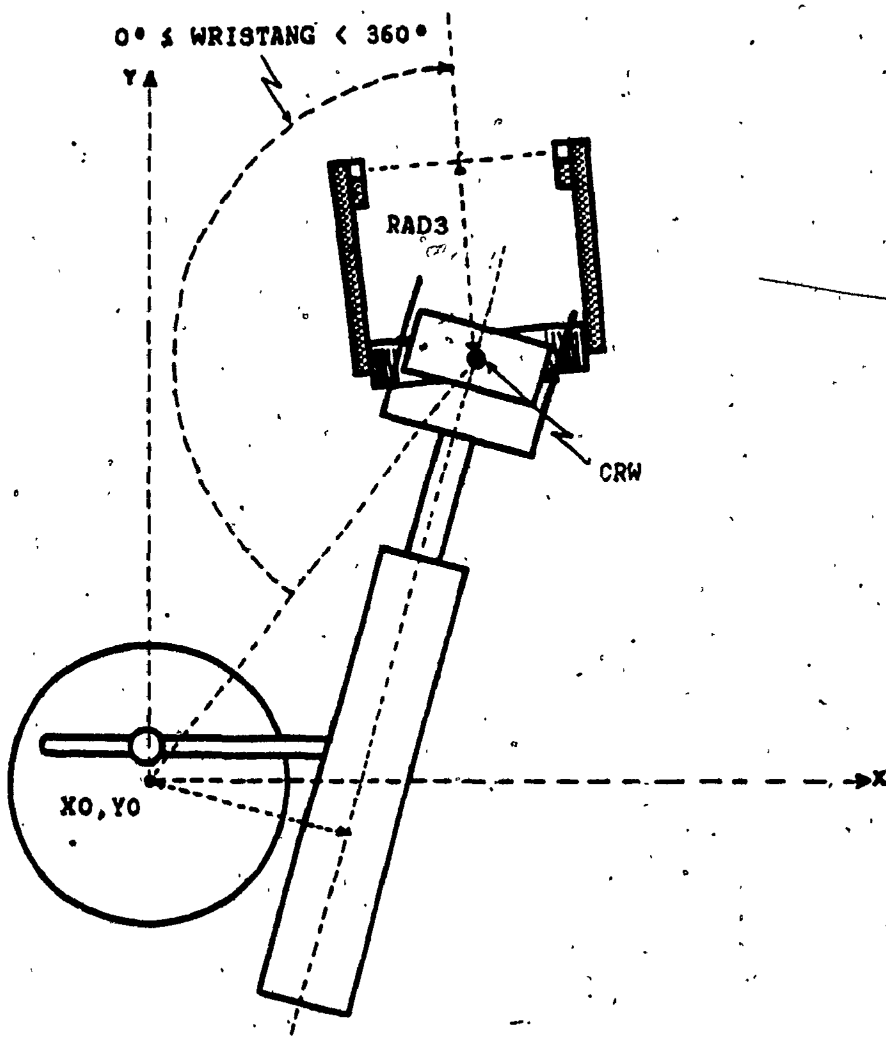


Fig 5.11 Determining the current position.

The initial position of the arm is the position from which all displacements are computed. It is specified by the initial CRW given by (XSTART,YSTART); by the initial wrist angle WASTART and by the coordinates of the finger tips (XOTIP,YOTIP) which may be deduced from the other three. These five values are set once after installation and are stored on disk along with the other parameters (CORX, CORY, XO, YO, etc...).

To obtain these values one must first position the end of the manipulator in the field of the camera and then type the 'O' command which will call procedure FINDPOS to obtain XSTART, YSTART, XOTIP, and YOTIP, and call function WRISTANG to obtain WASTART from the above four values. The command will also do some initialisation by calling procedure INIMOT. The functionality of FINDPOS and WRISTANG will be described after what follows.

5.3.6 Setting the limits for moves

After having determined the initial position, the number of pulses each motor has received (MPULSE[i]) is 0. The arm can then be moved along each degree of freedom and the maximum (+ direction) and the minimum (- direction) number of pulses can be established for each motor.

In practice, limits are established only for motors 1, 3, and 4. The reason for this is that motor 6 is not used and that motors 2 and 5 are used but always with the same constant number of pulses. The moves on motors 2, 5, and 6 do not depend on the position of the object to be picked.

As we will later see these limits are defined so that it will be possible to determine the reachability of the object.

To set these min/max values (in global variables MINPUL[i] and MAXPUL[i]) for motors i=1,3,4, one moves on motor i to the max or min of its course. The 'M' command is then typed. It asks if it is to set a min or a max and to which motor it applies. It then assigns the corresponding MPULSE[i] value to the MINPUL[i] or MAXPUL[i] variable.

5.3.7 Procedure FINDPOS

As mentioned earlier, procedure FINDPOS returns the current position of the center of rotation of the wrist (CRW) and the coordinates of the finger tips (gripper closed). To do so, it first commands a move of the wrist (motor 3) of approximately 160° in the minus direction by calling procedure MOVEARM. It then takes a picture by calling procedure GETSHOW which combines calls to assembler subroutines FRGRAB, ENHANCE2, and MOVESCR. Subroutine DETDOTS is then called to obtain a set of dots which are translated to the base coordinate frame. Each pair of dots whose separating distance is less than the threshold value DBLDOT (0.635cm) is replaced by a single dot located at the center of the line joining the two parent dots. This is done because sometimes (depending on light conditions) a single dot is returned as two adjacent dots.

Normally the above process should return only one dot: the white spots on the finger tips. But provisions have been made so that if the picture contains undesired information, the procedure will be able to determine which is relevant. This is done using procedure GESSPOS which is given the number of pulses motors 1,3, and 4 have received and which returns the position where the finger tips should be. Therefore, if more than one dot is returned by DETDOTS, the one selected will be the nearest from the one returned by GESSPOS.

Note that this dot discrimination feature can not work before the initial position ('O' command) has been determined since the pulse counters are not yet defined at that time. Since the 'O' command makes a call to FINDPOS, one must make sure that at that time the picture contains no invalid information.

Until now, FINDPOS has registered a single dot at approximately 160° in the minus direction from the position of the tip when entering the procedure. The same process is repeated to register the second and the third dot, each time moving the finger tips approximately 80° in the positive direction. This is illustrated in figure 5.12.

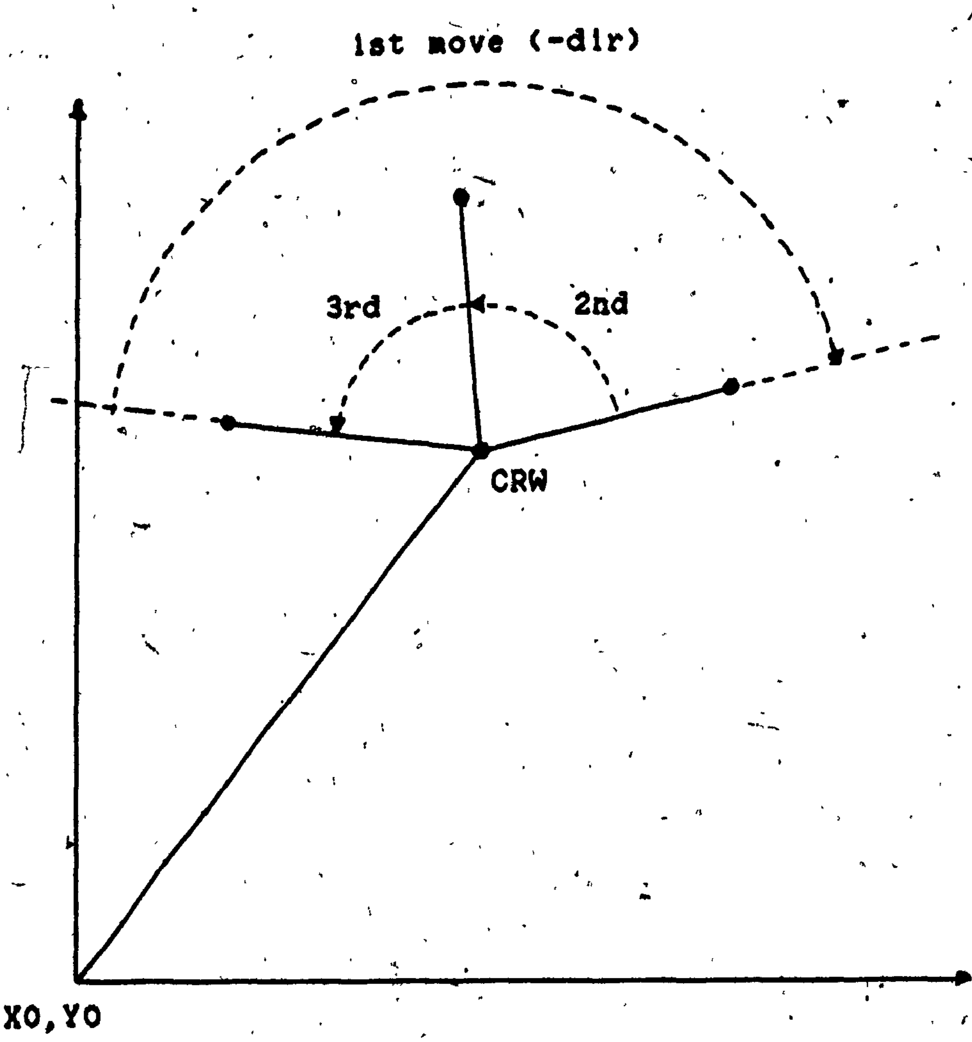


Fig 5.12 Finger tips moves.

The three registered dots lie on a circle whose center is the desired CRW. By calling procedure CIRCLE we obtain the coordinates of the center. The coordinates of the finger tips are simply the coordinates of the third dot. FINDPOS can then return to the caller, leaving the arm in the same position it had upon enterring the procedure.

5.3.8 Function WRISTANG

WRISTANG is given the x and y coordinates of two dots: D1X,D1Y and D2X,D2Y. It returns the angle (degree) between segments D0-D1 and D1-D2 where D0 is the origin X0,Y0 which must have been set. The angle is taken clockwise from D0-D1 to D1-D2. 0° <= WRISTANG < 360°. Figure 5.13 shows the dots and the desired angle.

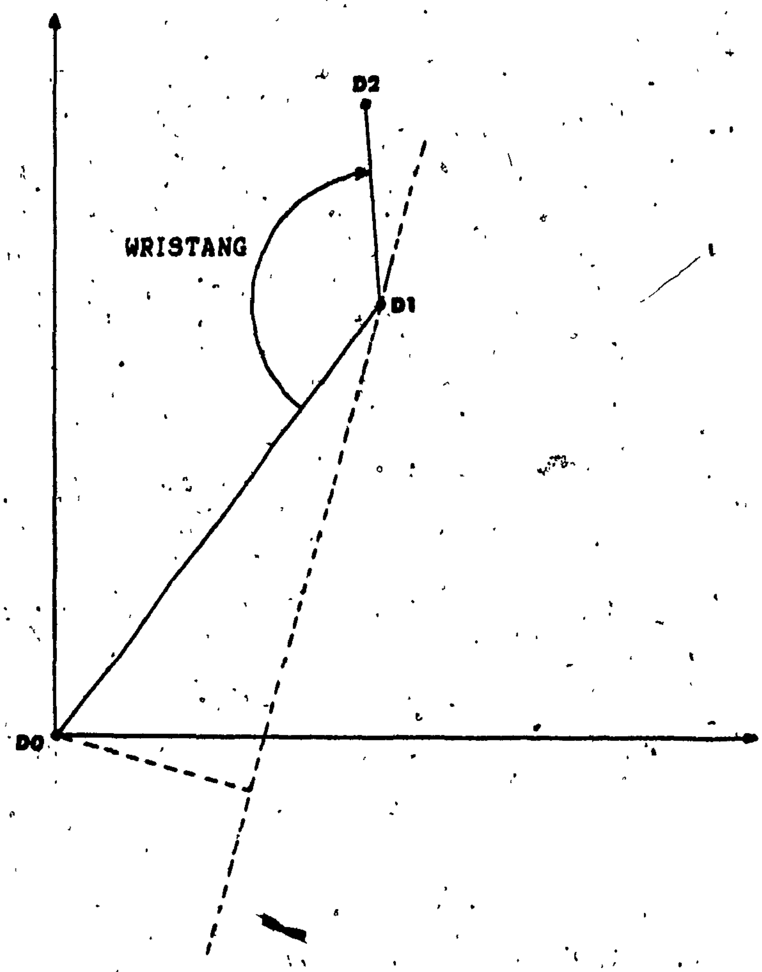


Fig 5.13 The wrist Angle.

WRISTANG is normally called with D1 being the center of rotation of the wrist CRW, and D2 being the tips of the fingers.

5.3.9 Procedure GESSPOS

This procedure deduces the position of the finger tips from the number of pulses motors 1, 3, and 4 have received. These values are stored in global variable MPULSE[i] and must be valid. GESSPOS also uses most of the previously defined parameters in its geometric/trigonometric computations: X0, Y0, NIDM1, RAD3, NIDM3, SHOULDER, NICM4, XSTART, YSTART, and WASTART. It reconstructs the path from the initial position where all pulse counters were 0, to the current position. This is not a simple path in polar coordinates since for example the move on the EXTEND line (motor 4) is not radial and therefore affects both the wrist angle and the position of the CRW.

As we have seen, this procedure is used in FINDPOS to determine the most plausible position of the finger tips given a picture containing more than one dot.

5.3.10 Detecting the Cross

Procedure WAITCROSS is responsable for detecting the cross to be picked, returning the coordinates of its four extremities, the coordinates of the four possible access points (defined later), and the four corresponding wrist angles.

The procedure continuously takes pictures by calls to GETSHOW, and for every picture, dots are extracted (DETDOTS) and transformed to the base coordinates. The set of dots is then processed: each pair of dots whose separating distance is less than the threshold value DBLDOT is replaced by a single dot at mid distance between the two original dots. A dot belongs to the cross if it has at least one neighbor dot at a distance of FAR ± TOLFAR and at least two neighbor dots at a distance NEAR ± TOLNEAR (figure 5.14). This does not garantee that the dot belongs to the cross, it is only a necessary condition, but the procedure tries to find four such dots mutually satisfying the condition. If the picture contains more than one cross, the first detected will be returned.

The global variable FAR defines the dimensions of the cross. NEAR can be computed from FAR: NEAR = SQRT(2)*FAR, but it is stored as a separate variable. TOLFAR and TOLNEAR are the tolerances accepted in testing for matches. The above four variables are manually set using command 'L' which prompts for their values. These parameters are stored on disk along with the others previously defined. The values selected are: FAR = 9.524cm; NEAR = 6.667cm; and TOLFAR = TOLNEAR = 0.952cm.

After having determined the four dots belonging to the cross, the procedure computes the four access points. Figure 5.14 shows the cross and its four access points.

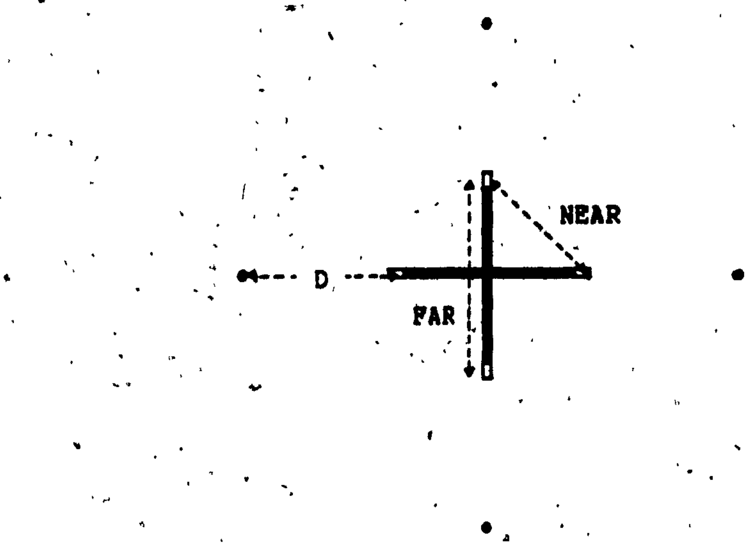


Fig 5.14 Access points and parameters for cross search.

In figure 5.14, D = RAD3 - GRIP3, The meaning of this will be explained later on.

The wrist angle corresponding to each access point is then computed by calls to WRISTANG with the access point as the first parameter and with the opposite extremity of the cross as the second parameter. This is illustrated in figure 5.15.

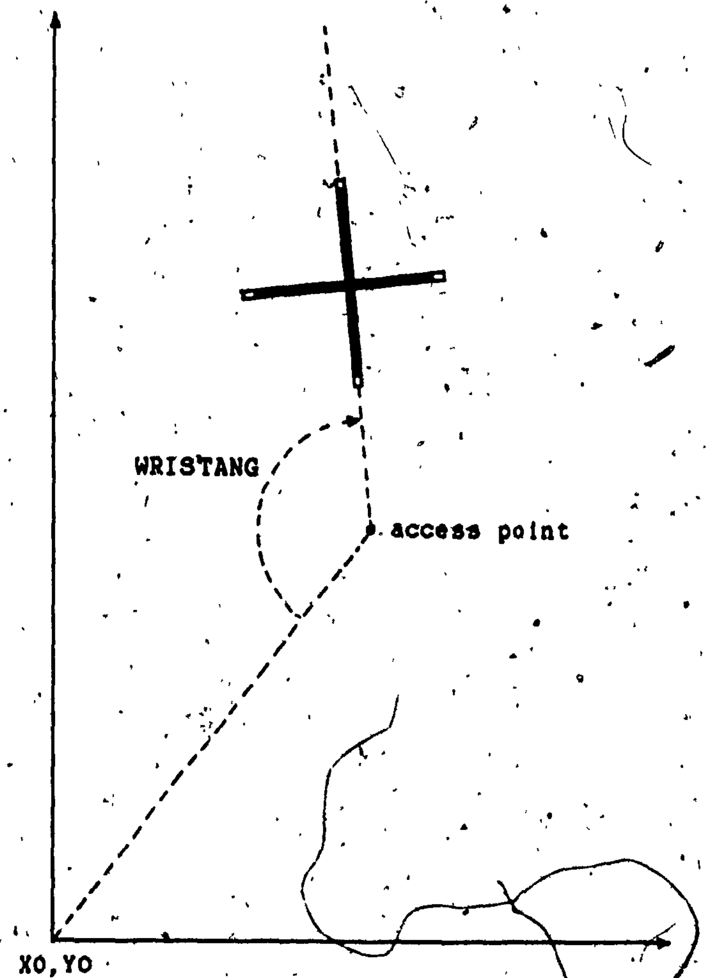


Fig 5.15 The cross and one of its 4 access points.

Note that an access point may fall outside the field of the camera, but this is not a problem as we will later see.

5.3.11 The cross and the required arm positioning

When the cross is about to be picked, the arm must occupy a precise position relative to it. The two fingers must be opened, the aperture being defined by the parameter CLOSE5, and the line joining the finger tips must be perpendicular to a branch of the cross in order that the two fingers be almost parallel to that branch. This way, the CRW of the arm lies on the line defined by that branch. This is illustrated in figure 5.16.

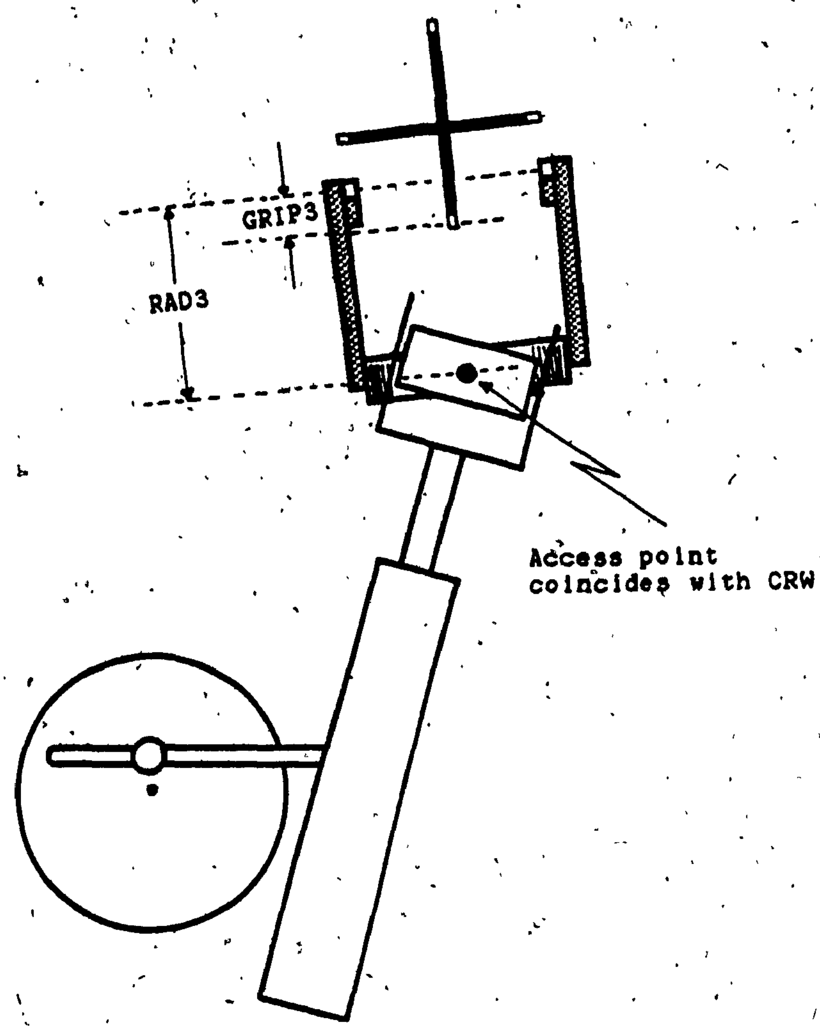


Fig. 5.16 Required finger positioning.

RAD3 of figure 5.16 has been defined before and can be considered here as the length of the fingers. GRIP3 is the distance in cm the fingers must cover on the cross, that is the contact length. Each of the four access points returned by WAITCROSS lies on the line defined by one of the branch of the cross at a distance RAD3-GRIP3 from the extremity of that branch.

The opening of the fingers is defined in terms of the number of pulses that must be applied to motor 5 to move from the closed position (where the two white dots on the finger tips coincide) to the opened position. This number of pulses is held in variable CLOSE5 and like GRIP3 is one of the parameters of the system that must be stored on disk file at system initialization. The values for GRIP3 and CLOSE5 are defined using the 'G' command. The values chosen were 1 inch (2.54cm) for GRIP3 and 250 pulses for CLOSE5. Applying 250 pulses to motor 5 in the minus direction when the gripper is closed, will result in a finger opening of about 8cm.

Recall that the position of the arm is uniquely determined by the position of its CRW and the value of the wrist angle. Therefore, accessing the cross is simply a matter of matching the CRW of the arm with one of the access points returned by WAITCROSS and adjusting the wrist angle so that the above finger alignment constraints are satisfied.

5.3.12 Moves from one position to another

The position of the arm is given by the x,y coordinates of its CRW, and the wrist angle at that position. What we want, is a procedure that will return the number of pulses required on motors 1, 3, and 4 to move from a current position to a destination position. Figure 5.17 shows the initial and final positions and figure 5.18 illustrates the geometrical aspects of the move.

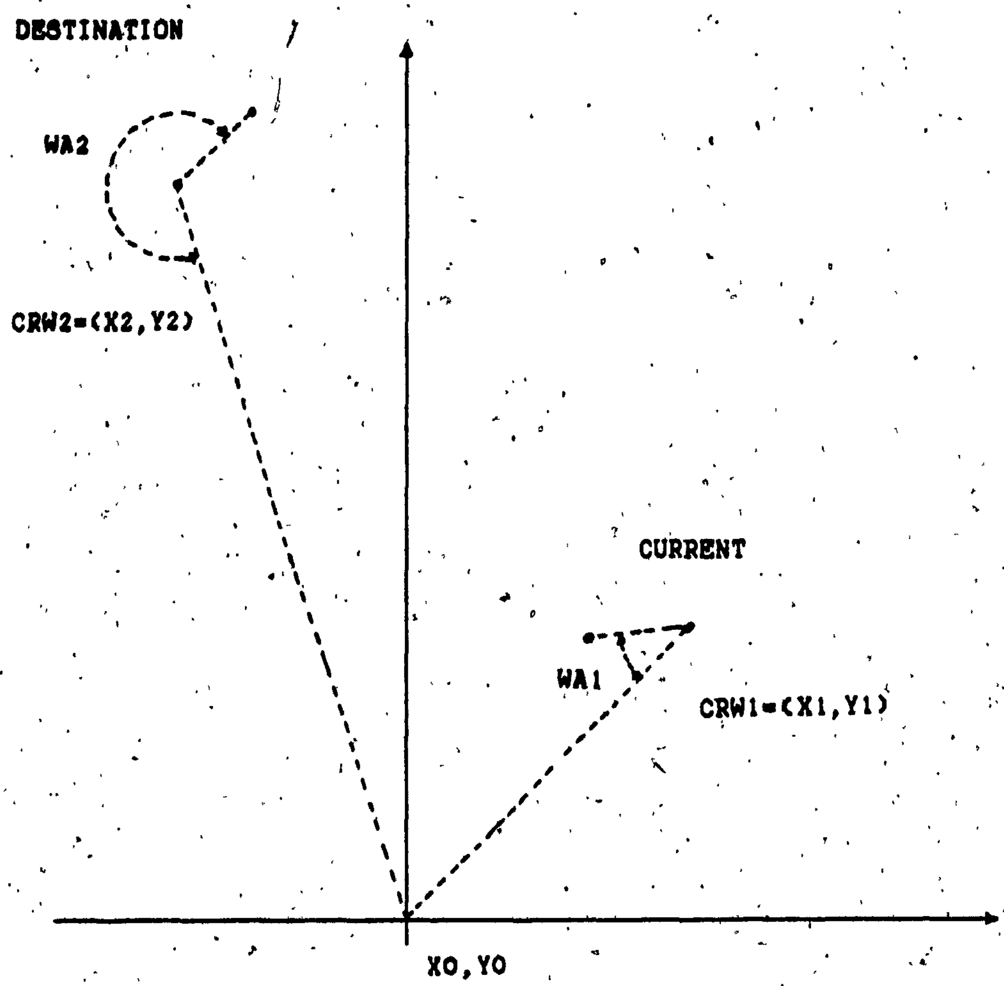


Fig 5.17 Initial and final positions for a move.

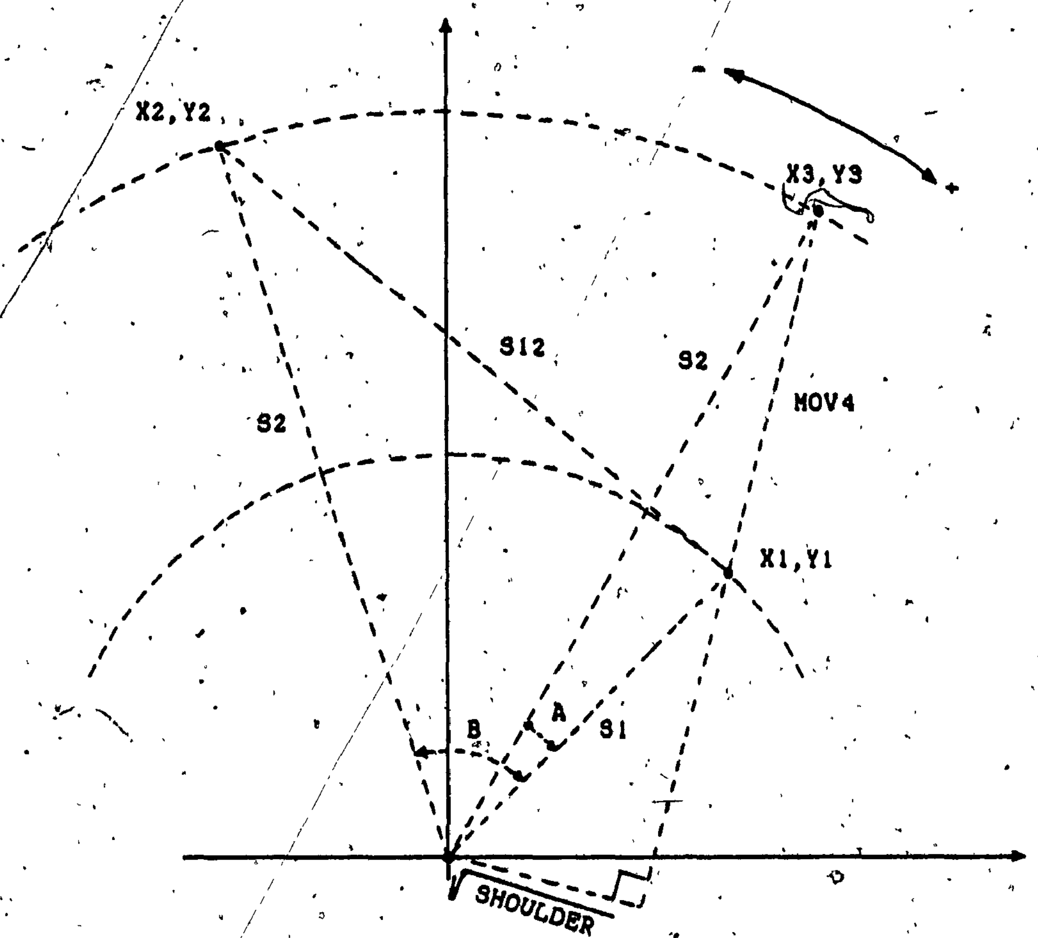


Fig 5.18 Computing angles A and B.

In order to move the CRW from X1,Y1 to X2,Y2 one must first move it a distance MOV4 along the extend line to get to X3,Y3.

$$S1 := SQRT(X1^2+Y1^2)$$

$$S2 := SQRT(X2^2+Y2^2)$$

Since S1, S2, and SHOULDER are known, and since the shoulder line is perpendicular to the extend line, one can write

$$MOV4 := SQRT(S2^2-SHOULDER) - SQRT(S1^2-SHOULDER)$$

The number of pulses required on motor 4 is therefore

$$PUL4 := NICM4*MOV4$$

Since the move on the extend is not radial, a certain angle A has been traveled, toward or away, from the destination. From the law of cosines

$$MOV4^2 = S1^2 + S2^2 - 2*S1*S2*cos(A)$$

from which we can obtain angle A. In the same manner we evaluate angle B, the angle between S1 and S2

$$S12^2 = S1^2 + S2^2 - 2*S1*S2*cos(B)$$

Angle A is then added to or subtracted from angle B to get the angle that must be travelled to get the CRW at X2,Y2. We then obtain the number of pulses required on motor 1

$$PUL1 := NIDM1 * (B \pm A)$$

The last thing left, is to determine PUL3, the number of pulses required on motor 3 to adjust the wrist angle at the desired destination position. Here again, since the move on the extend line is not radial, it has modified the initial wrist angle WA1 to WA3 as illustrated in figure 5,19.

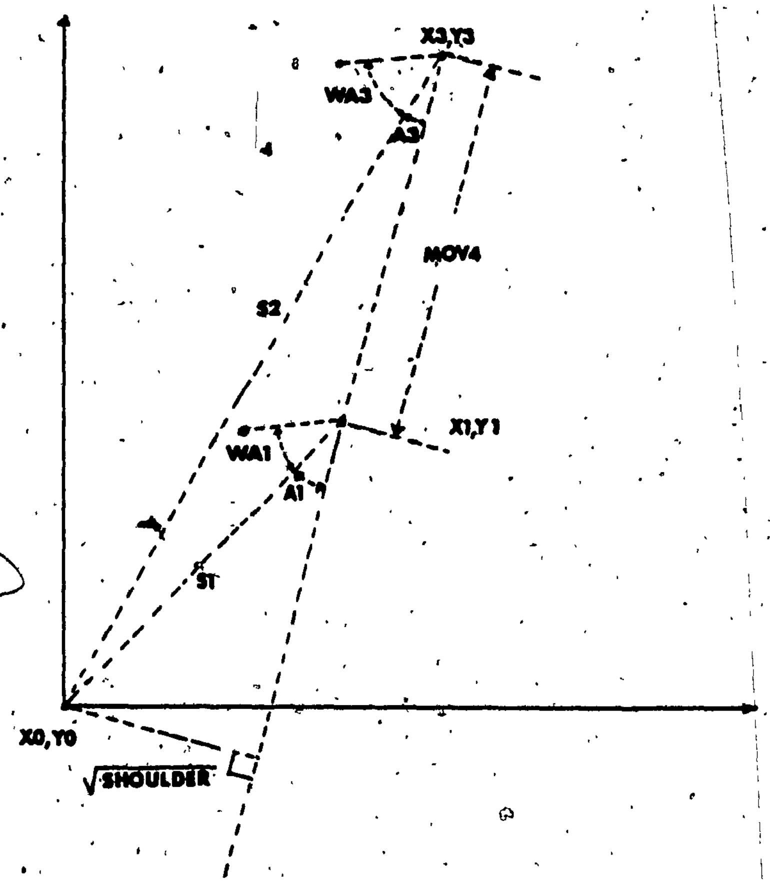


Fig 5.19 Computing the wrist angle at destination.

From figure 5.19 one can write

$$A1 + WA1 = A3 + WA3$$

WA1 is known and A1 and A3 can be computed

A1 := ARCTAN[SQRT(SHOULDER/(S1^2-SHOULDER))]

A3 := ARCTAN[SQRT(SHOULDER/(S2^2-SHOULDER))]

Once WA3 is known it is added to or subtracted from the desired destination wristangle WA2. The number of pulses required on motor 3 is then given by

PUL3 := NIDM3 * (WA2 ± WA3)

The above algorithm is implemented in procedure GIVEPUL which is given X1, Y1, WA1, X2, Y2, and WA2 as input and which returns PUL1, PUL3, and PUL4 as output. The actual moves are done by calls to procedure MOVEARM

```
MOVEARM(1, PUL1);
MOVEARM(3, PUL3);
MOVEARM(4, PUL4);
```

5.3.13 The three possible heights

As mentioned before, motor 6 (the wrist pivot) is not used, motors 1, 3, and 4 are used to command the arm positioning, and motor 5 controls the gripper opening. What about motor 2 which commands the up/down moves? It is used to position the arm at three well defined heights: the normal height where the white spots on finger tips are at the same level as the ones on the cross, and where the fingers are parallel to the black plane; the picking height, which is below the normal height so that the cross could be grasped; and the travel height which is above the normal height so that when the arm travels toward the cross it will not hit it.

The arm is adjusted to the normal height from the keyboard at system installation. The extend line must be parallel to the black plane, the axis for wrist rotation must be perpendicular to the plane implying that the fingers are parallel to it as illustrated in figure 5.20.

The travel height is specified by a constant number of pulses to be applied to motor 2 from the normal height. This negative value is stored in parameter UPM2. The picking height is specified as the constant number of pulses that must be applied from the travel height. This positive number is held in parameter PICKM2. Parameter NORMM2 holds the number of pulses to move from the picking height to the normal height. Figure 5.20 illustrates this.

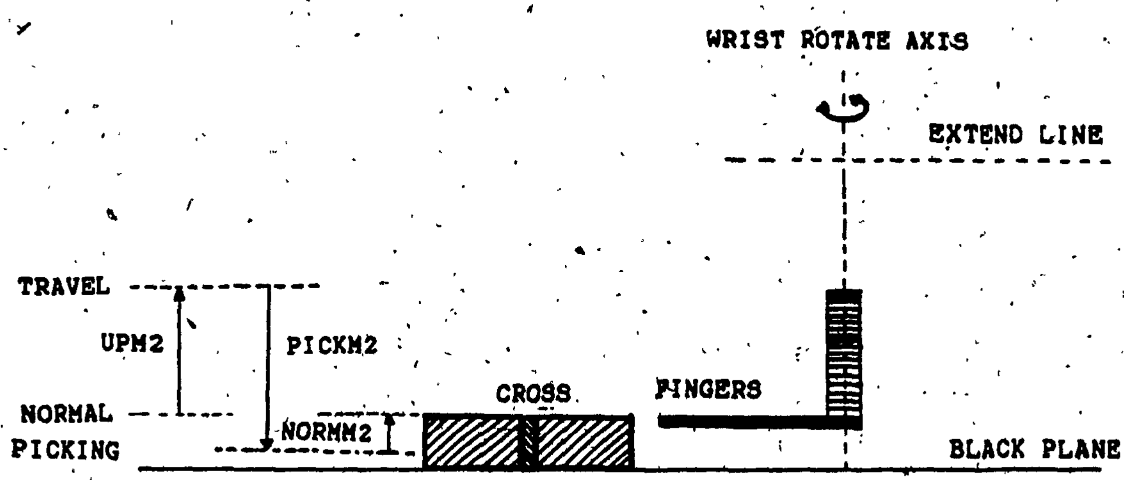


Fig 5.20 The three possible heights.

The above values are determined by experimenting and are assigned to the parameters at any time using the 'U' command. The values selected were: UPM2 = -150, PICKM2 = +190, and NORMM2 = -40.

5.3.14 The Task

The task of identifying and picking the cross can now be described in terms of what has been defined. The process is initiated using the 'X' command. At this point, the gripper must be closed and the arm must be at the normal height.

Upon entering the command, the arm is moved to the initial position by applying the same number of pulses motors 1, 3, and 4 have received, but in the opposite direction. The arm is then repositioned to determine the new initial position (which should not differ greatly from the previous one): XSTART, YSTART, XOTIP, YOTIP, WASTART. This is done by calls to procedures FINDPOS and WRISTANG. The counters are then initialized (INITMOT).

The arm is then raised to the travel height applying UPM2 pulses to motor 2. It is then moved outside the camera field by a predetermined constant number of pulses on motor 1. This positive number is held in parameter STANDBY and as with the parameter DROPCROSS (defined below), it can be determined by experimenting and can be assigned to the parameters using the 'Y' command. At this stage, the gripper is opened by applying -CLOSE5 pulses to motor 5.

The program now waits for the cross: a call is made to WAITCROSS which will return the four possible acces points and the four corresponding wrist angles.

Each access point is then tested until a reachable one is found: procedure GIVEPUL is used to compute the number of pulses NOPUL1, NOPUL3, NOPUL4, required on motors 1, 3, 4 respectively, to move from the initial position XSTART, YSTART, WASTART to the access position given by the access point and its corresponding wrist angle. An access point is considered reachable if

$$(NOPUL_i >= MINPUL[i]) \text{ AND } (NOPUL_i <= MAXPUL[i])$$

where i=1,3,4 and where MINPUL and MAXPUL are the limits from the initial position as set by the 'M' command.
The order of testing is dictated by the distance of the access point to the origin; The nearest is tested first. One could also use the distance to the initial position as the criterion. Another approach would be to evaluate NOPUL1, NOPUL3, and NOPUL4 for each access point and test first the one that minimizes the total number of pulses or the total time required.

If none of the access point is reachable, the program announces it and asks the user if he wants to reposition the cross or exit the command. In the case the command is exited, the gripper is closed and the arm is lowered at the normal height.

If a reachable access point is found, the arm is moved toward it. Since NOPUL1 was computed from the initial position and since the arm is now at an offset STANDBY from that position, the number of pulses received by motor 1 is NOPUL1-STANDBY. Motors 3 and 4 receive NOPUL3 and NOPUL4 respectively. The arm is then lowered at the picking height, the gripper closed, and the arm repositioned at the travel height. The arm will then travel along the opposite path to the initial position by applying the negative number of pulses: -NOPUL1, -NOPUL3, and -NOPUL4. From the initial position, the arm is again moved but on motor 1 only by the constant number of pulses given by parameter DROPCROSS. At this stage the cross is outside the camera field and it is dropped by opening and then closing the gripper. The arm is then moved to the standby position by applying STANDBY-DROPCROSS pulses to motor 1. Finally it is lowered to the normal height (PICKM2 + NORMM2 pulses to motor 2), and the command is exited.

5.3.15 Using the program

When the main procedure ARMMOVE is entered from the main program (command 'A'), it asks if the parameters are to be loaded from a file. If this is not the case (new installation), then they must all be determined as described before; otherwise, the only thing to do is to position the arm at the initial position. This is done by the '*' command which determines the current position (FINDPOS) and which moves (GIVEPUL and MOVEARM) the arm to the initial position specified by the file parameters XSTART, YSTART, and WASTART. The program is now ready for the 'X' command. When the procedure is exited ('Z' command) it asks if the parameters are to be saved on file.

Table 5.1 lists all the parameters, the commands to set them, and their meanings.

TABLE 5.1 List of Task parameters.

CORX, CORY	'C'	correction factors
XO, YO, NIDM1	'E'	center of rotation and number of pulses/degree for motor 1
RAD3, NIDM3	'E'	radius of rotation and number of pulses/degree for motor 3
SHOULDER, NICM4	'A'	square of distance between XO,YO and the extend line; number of pulses/cm motor 4
MOTHOLD[i]	'H'	step motors hold times
XSTART, YSTART, WASTART XOTIP, YOTIP	'O'	coordinates of initial position
MINPUL[i], MAXPUL[i]	'M'	limits of moves from the initial position
UPM2, PICKM2, NORMM2	'U'	the 3 possible heights expressed in number of pulses on motor 2
CLOSE5, GRIP3	'G'	gripper openning (pulses on motor 5) and contact lenght on cross (cm)
FAR, TOLFAR, NEAR, TOLNEAR	'L'	lenghts and tolerances for cross search
STANDBY, DROPCROSS	'Y'	number of pulses (motor 1) to move from initial pos., to the standby and dropcross positions respectively.

5.4 The results

The program has been extensively tested, and every time the cross is considered reachable, the manipulator is able to pick it and move it outside the camera field. The actual approach position always falls within an acceptable range around the expected position, so that the cross can be lifted. The process could be repeated almost indefinitely since the system determines the initial position at the beginning of every cross grabbing cycle. However, moves that are not seen by the camera (up/down moves) could lead to unexpected displacements from the computed position. This has not been the case but in the long run it could happen.

Figures 5.21 to 5.26 show the different steps of the task execution. Figure 5.21 shows the arm in the standby state where the system waits for the cross to appear in the camera field. The arm is at the normal height and the gripper is open. Figure 5.22 is a picture of the arm approaching the cross. It has already rotated its head (motor 1) and extended its hand (motor 4). It is just about to rotate its wrist (motor 3). Figure 5.23 shows that the arm has been lowered to the picking height, and that the gripper is just about to be closed. In figure 5.24 we see the manipulator leaving with the cross. The gripper has been closed, the arm raised to the travel height, and the wrist is now rotating. Figure 5.25 shows the arm moving (head rotate) to the position where it will drop the cross. In figure 5.26 we see the arm dropping the cross outside the camera field. The next moves are: closing the gripper, lowering the arm at the normal height, and moving to the standby position.

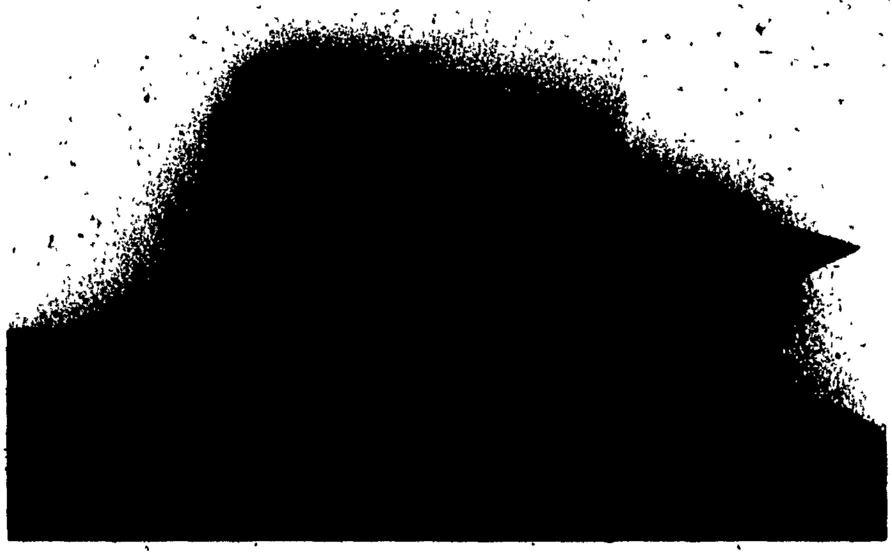


Fig 5.21 Arm in Standby State.

Fig 5.22 Arm approaching the Cross.

Fig 5.23 Arm just about to pick the Cross.

Fig 5.24 Arm leaving with the Cross.

Fig 5.25 Arm near destination.

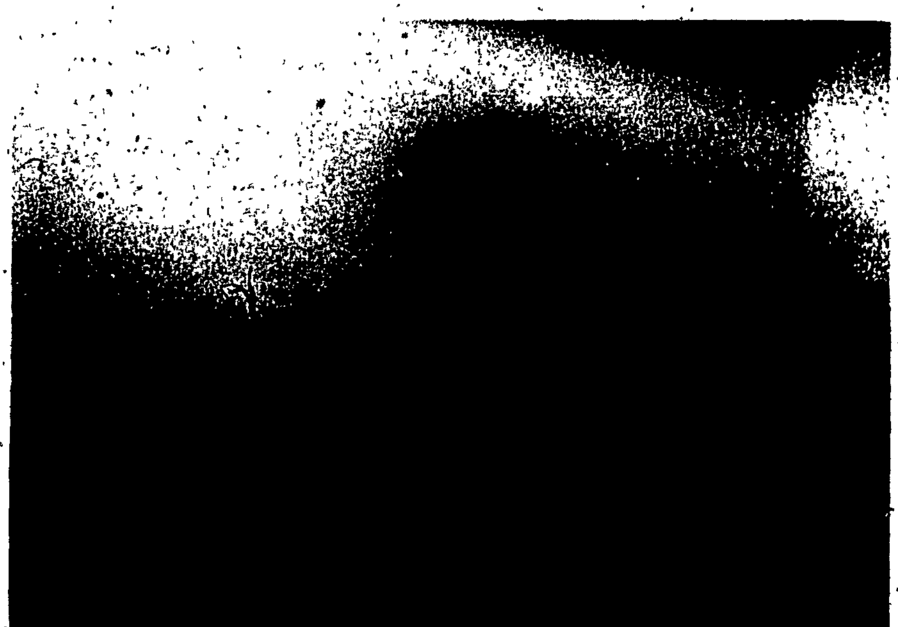


Fig 5.26 Arm dropping the cross.

5.5 Possible improvements

The system could be improved by having the camera to monitor the final approach, that is when the gripper is just about to be closed on the cross. The white spots on the finger tips could be localized relative to the cross and the arm incrementally moved until its position complies with the desired one. After the gripper has been closed, the system could also test if this action had the desired effect.

A second camera could also be added to track vertical moves. In the present implementation, a pulse counter is the only tool for describing the height, a second camera would be used to make corrections when necessary.

CHAPTER 6

CONCLUSION

Our approach is such that the degrees of freedom of the arm are defined entirely in terms of the vision input parameters. The arm "learns" the relationships between its movements and positions which are defined in terms of pixels in the vision input system. The parameters of its motion equations are computed in these terms, and the system is therefore able to define its movements directly in terms of pulses to the manipulator's motors.

We have demonstrated that, given a robot system equipped with a camera, it is possible to use that camera to localize the robot manipulator's end effector as well as the objects involved in the robot's task. In the standard approach, the objects and the manipulator are treated separately: the camera localizes the objects; feedbacks from internal sensors are used to obtain positions of each joint of the manipulator, and then, using homogeneous transformations, the position of its end effector; the final approach (grasping) being monitored by force sensors on the end effector.

The goal of the project was not to give incentives for robot sensing, this has been done already (Albus, 1981; Nitzan, 1983) but rather to show that existing robot camera systems could be used more efficiently. The following task illustrates a situation where the standard approach may not work.

Let's consider the task of picking a delicate object (e.g. a sphere containing nitro-glicerine), currently in a non-stable equilibrium state (see figure 6.1). The standard approach requires contact with the object (force sensors) in order to fine tune the end effector's final grasping position. In this case, this would lead to a "disaster" since any non-zero force would move the object far from its current position. In our approach ("believe what the camera sees"), the camera is used to monitor the end effector's position and its final approach, requiring no blind contact with the object.

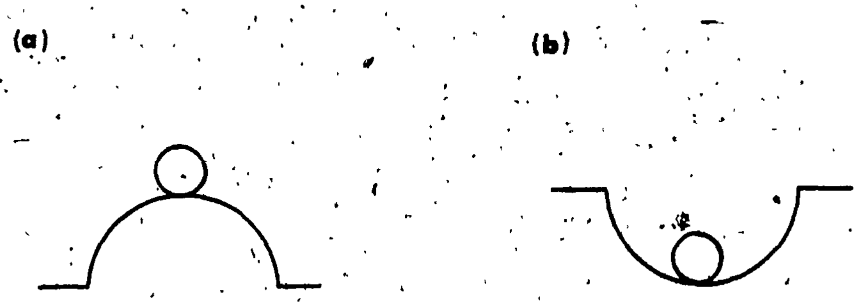


Fig 6.1 (a): Non-stable equilibrium state. (b): Stable equilibrium state.

REFERENCES

Albus, J.S. (1981). Brains, behavior, and robotics. Peterborough: BYTE Books. 1981.

Bovis, M., Caillaud, L. (1973). Initiation à la photographie. Paris: Le Bélier-Prisma.

Ciarcia, S.A. (1983a). Build the Micro D-Cam solid-state video camera, Part 1. BYTE magazine, 8, (No. 9), 20-31.

Ciarcia, S.A. (1983b). Build the Micro D-Cam solid-state video camera, Part 2. BYTE magazine, 8, (No. 10), 67-86.

Eggebrecht, L.C. (1983). Interfacing to the IBM Personal Computer. Indianapolis: Howard W. Sams.

International Business Machines, (1982a). IBM Macro Assembler. Boca Raton: IBM Personal Computer Software Reference Library.

International Business Machines, (1982b). IBM PC Technical Reference. Boca Raton: IBM Personal Computer Hardware Reference Library.

International Business Machines, (1983). DOS 2.1 reference manual. Boca Raton: IBM Personal Computer Software Reference Library.

Lozano-Pérez, T. (1983). Robot Programming. Proceedings of the IEEE, 71, 821-841.

Morgan, C.L., Waite M. (1982). 8086/8088 16-bit microprocessor primer. Peterborough: McGraw-Hill.

Micromint inc. (1983). Micro D-Cam Users Manual. Cedarhurst: The Micromint inc.

Nitzan, D., Barrouil, C., Cheeseman, P., Smith, R. (1983). Use of Sensors in Robot Systems. ICAR Proceedings.

Paul, R.P. (1981). Robot Manipulators: Mathematics, Programming, and Control. Cambridge: MIT Press.

Selby, S.M. (1975). Standard Mathematical Tables. Cleveland: CRC Press.

Software Building Blocks inc. (1984). SBB Pascal Manual. Ithaca: Software Building Blocks, inc.

APPENDIX

PASCAL AND ASSEMBLER SOURCE LISTINGS

```
PROGRAM CAMPAS;
($$M64  64K OF DATA$)
(* AUTHOR: BERNARD BROCHU *)

CONST
    MAXDOT = 100;    (* MAX NUMBER OF DOTS IN A FRAME FROM CAMERA *)
    DBLDOT = 0.635;  (* DIST IN cm UNDER WHICH 2 DOTS ARE CONSIDERED AS
                               A SINGLE DOT  *)
TYPE
    REALARRAY = ARRAY[1..MAXDOT] OF REAL;
    PARAMETER = RECORD
             KEYIN: INTEGER;   (* RETURNED ASCII KEY IN LS
                                               BYTE (AL)  *)

             CTRLOUT: INTEGER; (* IN LS BYTE (AL)  *)
                   HOLDTIME: INTEGER; (* PULSE HOLD TIME (MSEC) *)
        END;
    MOTTYPE= 0..6;  (* MOTOR NUMBER: 0=ALL MOTORS OFF  *)
    STATYPE= 0..3;  (* THE 4 MOTOR STATES *)

    BYTE = 0..255;

    MISC =   RECORD
      KEY: INTEGER;  (* RETURNED ASCII KEY IN LS BYTE (AL)
                                  OR EXPOSURE TIME UPON ENTRY TO FRGRAB *)
      BYTECNT: INTEGER; (* NUMBER OF BYTES READ BY FRGRAB *)
      PORTADD: INTEGER; (* CAMERA PORT ADDRESS *)
      SUMB: INTEGER;    (* SUM OF ALL BYTES READ *)
    END;
    DOTREC = RECORD ROW,COL: INTEGER END; (* DOT COORDINATES *)
    DOTS = ARRAY[1..MAXDOT] OF DOTREC;    (* RETURNED BY DETDOTS *)

    SCREEN_REC =   RECORD
      FIRST_BYTE, SCREEN_START, SCR_ROWCT, SCR_COLCT : INTEGER;
     END;

    FILESTRING = STRING 14;

    M128X32 = ARRAY[1..128] OF ARRAY[1..32] OF BYTE;
      (* 128 ROWS X 256 COL: RECEIVED BY FRGRAB FROM CAMERA  *)

    M130X80 = ARRAY[1..130] OF ARRAY[1..80] OF BYTE;
      (* 128 ROWS X 640 COL: PRODUCED BY ENHANCE2 AND TO BE DISPLAYED BY
           MOVESCR.
         NOTE: 2 ROWS HAVE BEEN ADDED (130 INSTEAD OF 128) BECAUSE
              ENHANCE2 OVERFLOWS IN THE 2 EXTRA ROWS          *)
($L+ $)
```

```
        REGISTERS = RECORD
                AL,AH,BL,BH,CL,CH,DL,DH: BYTE;
                BP,SI,DI: INTEGER;
                INTNO: INTEGER; (* BIOS INT NUMBER 00..1F *)
            END;

(* ######################################################################## *)
VAR
    PI,RADDEG: REAL;
    PARAM: PARAMETER;
    MSTATE: ARRAY[MOTTYPE] OF STATYPE;
    MPOLSE: ARRAY[MOTTYPE] OF INTEGER;


    (* FILE PARMF VARIABLES *)
    CORX, CORY,X0,Y0,NIBM1,RAD3,NIDM3,SHOULDER,NICM4: REAL;
      (* SHOULDER IS SQUARE OF DIST BETWEEN LINE OF ELONGATION
         OF MOTOR 4 AND ORIGIN (I.E. CENTER OF ROTATION MOTOR 1 *)
    MOTHOLD: ARRAY[MOTTYPE] OF INTEGER; (* MOTHOLD[0] <--MAX OF 1..6 *)
    XSTART,YSTART,WASTART,XOTIP,YOTIP: REAL;
    UPM2,PICKM2,NORMM2,CLOSE5: INTEGER;
    GRIP3: REAL;
    MINPUL,MAXPUL: ARRAY[1..6] OF INTEGER;
    FAR,TOLFAR,NEAR,TOLNEAR: REAL;
    STANDBY,DROPCROSS: INTEGER;


    TEMP: REAL;

    COORD: DOTREC;
    DOT: DOTS;
    PARMF: TEXT;


    PICFILE : FILE OF BYTE;
    FILENAME: FILESTRING;
    PICB: BYTE;

    FR1 : M120X32;
    FR2 : M130X80;

    VIDMODE : INTEGER;
    PARAM1: MISC; PARAM2: SCREEN_REC; C : CHAR;
    I,J: INTEGER;
    ROW,COL: INTEGER;
    EXPTIME: INTEGER;
(*$L+ *)
```

```
Line File Stmt Level
 95   95    1    1    (* *********************** ASSEMBLER SUBROUTINES *************************** *)
 96   96    1    1
 97   97    1    1    PROCEDURE FRGRAB(VAR X: MISC; VAR Y: M128X32); EXTERNAL;
 98   98    1    1    (* GET PICTURE FOM CAMERA AND STORE INTO VARIABLE *)
 99   99    1    1
100  100    1    1    PROCEDURE MOVESCR(VAR X: SCREEN_REC; VAR Y: M130X80); EXTERNAL;
101  101    1    1    (* MOVE PICTURE FROM VARIABLE TO SCREEN (VIDEO-RAM)  *)
102  102    1    1
103  103    1    1    PROCEDURE GMODE(VAR X: INTEGER); EXTERNAL;
104  104    1    1    (* SET VIDEO MODE AND RETURN PREVIOUS MODE *)
105  105    1    1
106  106    1    1    PROCEDURE ENHANCE2(VAR X: M128X32; VAR Y: M130X80); EXTERNAL;
107  107    1    1    (* MODIFY PICTURE X INTO PICTURE Y *)
108  108    1    1
109  109    1    1    PROCEDURE CALLBIOS(VAR X: REGISTERS); EXTERNAL;
110  110    1    1    (* CALL TO ROM BASIC I/O SYSTEM *)
111  111    1    1
112  112    1    1    PROCEDURE CALLDOS(VAR X: REGISTERS); EXTERNAL;
113  113    1    1    (* CALL TO OPERATING SYSTEM *)
114  114    1    1
115  115    1    1    PROCEDURE CLRCOMM; EXTERNAL;
116  116    1    1    (* TO CLEAR ROW 16 TO 24 OF DISPLAY SCREEN: COMMUNICATION AREA *)
117  117    1    1
118  118    1    1    PROCEDURE BLINK(VAR X:DOTREC); EXTERNAL;
119  119    1    1    (* WILL MAKE THE GIVEN DOT BLINK, WILL STOP WHEN A KEY IS PRESSED *)
120  120    1    1
121  121    1    1    PROCEDURE DETDOTS(VAR X:DOTS; VAR Y: M130X80); EXTERNAL;
122  122    1    1    (* RETURN ARRAY OF DOT COORDINATES CONTAINED IN PICTURE *)
123  123    1    1
124  124    1    1    PROCEDURE ARMOUT(VAR X: PARAMETER); EXTERNAL;
125  125    1    1    (* WILL WRITE BYTE TO IO PORT 0378H AND RETURN KEY TYPED IF ANY *)
126  126    1    1    (*$L+ *)
```

```
Stmt Level
  1    1    (* ************************************************************************ *)
  1    1    FUNCTION COS(ANGLE: REAL): REAL;
  1    1    (* COS FUNCTION BECAUSE OF BUG IN STANDARD ONE *)
  1    1    VAR A,COSINE: REAL;
  1    2    BEGIN
  1    2    A := ABS(ANGLE);  (*  COS(A) = COS(-A)  *)
  2    2    WHILE A >= 2*PI DO A := A - 2*PI;
  4    2    COSINE := SQRT(  1  -  SQR( SIN(A) )   );
  5    2    IF (A > PI/2) AND (A < 3*PI/2) THEN COS := -COSINE
  7    2    ELSE COS := COSINE;
  8    2    END;
  9    1
  9    1    (* ************************************************************************ *)
  9    1    PROCEDURE CADRE;
  9    1    VAR I,J: INTEGER;
  9    2    BEGIN
  9    2    FOR I:= 1 TO 80 DO
 10    3        BEGIN
 11    4        FR2[1,I] := 255;
 12    4        FR2[128,I] := 255;
 13    4        END;
 14    2    FOR I := 1 TO 128 DO
 15    3        BEGIN
 16    4        IF FR2[I,1] < 128 THEN FR2[I,1] := FR2[I,1] + 128;
 18    4        IF NOT(ODD(FR2[I,80])) THEN FR2[I,80] := FR2[I,80] + 1;
 20    4        END;
 21    2    END;
 22    1
 22    1    (* ************************************************************************ *)
 22    1    PROCEDURE GETSHOW;
 22    1    (* TAKE PICTURE, ENHANCE AND DISPLAY *)
 22    1    BEGIN
 22    2    PARAM1.KEY := EXPTIME;
 23    2    PARAM1.PORTADD := 792; (* 792 = 0318H = CAMERA PORT *)
 24    2    FRGRAB(PARAM1,FR1);
 25    2    ENHANCE2(FR1,FR2);
 26    2    CADRE;
 27    2    MOVESCR(PARAM2,FR2);
 28    2    END;
 29    1    (*$L+ *)
```

```
Stat Level
 29   1   (* ************************************************************************** *)
 29   1   PROCEDURE HOME;
 29   1   (* POSITION CURSOR AT TOP/LEFT OF COMMUNICATION AREA *)
 29   1   VAR I,J: INTEGER;
 29   2       REGSET: REGISTERS;
 29   2   BEGIN
 29   2   CLRCOMM;
 30   2   REGSET.AH := 2;  (* FUNCTION IS POSITION CURSOR *)
 31   2   REGSET.BH := 0;  (* PAGE #, ALWAYS 0 IN GRAPHIC MODE *)
 32   2   REGSET.DH := 17; (* ROW  0..24 *)
 33   2   REGSET.DL := 0;  (* COL  0..79 *)
 34   2   REGSET.INTNO := 16; (* INT 10H    IS VIDEO ACCESS *)
 35   2   CALLBIOS(REGSET);
 36   2   END;
 37   1
 37   1   (* ************************************************************************** *)
 37   1   PROCEDURE PRINTSCR;
 37   1   (* PRINT CURRENT DISPLAYED PICTURE *)
 37   1   VAR
 37   2       REGSET: REGISTERS;
 37   2   BEGIN
 37   2   REGSET.INTNO := 5; (* INT 5 IS PRINT SCREEN *)
 38   2   CALLBIOS(REGSET);
 39   2   END;
 40   1
 40   1
 40   1   (* ************************************************************************** *)
 40   1   PROCEDURE TRANSLAT(VAR X,Y: REAL);
 40   1   (* X0,Y0 IS 0,0 I.E. LEFTMOST,TOPMOST PIXEL (COL 0,ROW 0)  DURING CALIBRATION,
 40   1       OR IS CENTER OF ROTATION (MOTOR1) AFTER CALIBRATION  *)
 40   1   VAR TEMP: REAL;
 40   2   BEGIN
 40   2   TEMP := X-X0;   X := TEMP;
 42   2   TEMP := Y-Y0;   Y := TEMP;
 44   2   END;
 45   1
 45   1
 45   1   (* ************************************************************************** *)
 45   1   PROCEDURE BLINK2(X,Y: REAL);
 45   1   (* X0,Y0 IS 0,0 I.E. LEFTMOST,TOPMOST PIXEL (COL 0,ROW 0)  DURING CALIBRATION,
 45   1       OR IS CENTER OF ROTATION (MOTOR1) AFTER CALIBRATION  *)
 45   1   (* CORX,CORY MAY BE 1.0,1.0 IF NOT YET SET *)
 45   1   BEGIN
 45   2   TEMP:= X+X0;
 46   2   COORD.COL := ROUND(TEMP/CORX);
 47   2   TEMP := -(Y+Y0);
 48   2   COORD.ROW := ROUND(TEMP/CORY);
 49   2   BLINK(COORD);
 50   2   END;
 51   1   (*$L+ *)
```

```
Stmt Level
 51    1    (* ************************************************************************** *)
 51    1    PROCEDURE GIVEPUL(X1,Y1,WANG1,X2,Y2,WANG2: REAL;
 51    1                      VAR NOPUL1,NOPUL3,NOPUL4: INTEGER);
 51    1    (* GIVEN POSITION OF CENTER OF ROTATION OF MOTOR 3 (X1,Y1) AND THE WRIST
 51    1       ANGLE AT THIS POSITION (WANG1), RETURN THE PULSES REQUIRED ON MOTORS
 51    1       1,3,4 TO MOVE TO X2,Y2 WITH A WRIST ANGLE WANG2.
 51    1       BOTS 1 AND 2 MUST BE RELATIVE TO X0,Y0 (THE CENTER OF ROTATION OF
 51    1       MOTOR 1).  THE RETURNED VALUES MAY BE POSITIVE OR NEGATIVE INDICATING
 51    1       THE DIRECTION OF MOVES  *)
 51    1    VAR
 51    2       S1,S2,S12,ANGLE,ANGLEA,ANGLEB,COSINE,SINE: REAL;
 51    2       MOV4,WADEST: REAL;
 51    2    BEGIN
 51    2    (* COMPUTE NOPUL4 TO MOVE FROM RADIUS S1 TO RADIUS S2 *)
 51    2    S1 := SQRT( SQR(X1) + SQR(Y1) );
 52    2    S2 := SQRT( SQR(X2) + SQR(Y2) );
 53    2    MOV4 := SQRT(S2*S2-SHOULDER)  -  SQRT(S1*S1-SHOULDER);
 54    2    NOPUL4 := ROUND(NICM4*MOV4);
 55    2    (* ANGLEA IS THE ANGLE COVERED BY MOVING MOTOR 4. WE MUST CONSIDER THIS
 55    2       BECAUSE MOVING MOTOR 4 DOES NOT ONLY CHANGE THE RADIUS TO X0,Y0
 55    2       BUT ALSO THE ANGLE (MOVEMENT IS NOT EXACTLY RADIAL)            *)
 55    2    COSINE := (MOV4*MOV4 - S1*S1 - S2*S2) / (-2*S1*S2);
 56    2    SINE := SQRT(1 - COSINE*COSINE);
 57    2    ANGLEA := ABS( RADDEG * ARCTAN(SINE/COSINE) );
 58    2    IF MOV4 >= 0 THEN ANGLEA := -ANGLEA;
 60    2    (* ANGLEB IS THE ANGLE COVERED BY MOVING MOTOR 1 *)
 60    2    S12 := SQRT( SQR(X1-X2) + SQR(Y1-Y2) );
 61    2    COSINE := (S12*S12 - S1*S1 - S2*S2) / (-2*S1*S2);
 62    2    SINE := SQRT(1 - COSINE*COSINE);
 63    2    ANGLEB :=  RADDEG * ARCTAN(SINE/COSINE);
 64    2    IF ANGLEB < 0 THEN ANGLEB := ANGLEB + 180;
 66    2    IF X1 > 0 THEN
 67    2       IF Y2  >  (Y1/X1) * X2   THEN
 68    2          ANGLEB := - ANGLEB;
 69    2    IF X1 < 0 THEN
 70    2       IF Y2  <  (Y1/X1) * X2   THEN
 71    2          ANGLEB := - ANGLEB;
 72    2    IF X1 = 0 THEN
 73    2       IF X2 < 0  THEN
 74    2          ANGLEB := - ANGLEB;
 75    2    ANGLE := ANGLEB - ANGLEA; (* ADJUST WITH ANGLE COVERED BY MOTOR 4 *)
 76    2    NOPUL1 := ROUND(NIDM1*ANGLE);
 77    2    (* WADEST: MOVING FROM X1,Y1 TO X2,Y2 WILL MODIFY THE WRIST ANGLE. *)
 77    2    WADEST := RADDEG * ARCTAN( SQRT( SHOULDER/(SQR(S1)-SHOULDER)))
 78    2              -  RADDEG * ARCTAN( SQRT( SHOULDER/(SQR(S2)-SHOULDER)))
 78    2              + WANG1;
 78    2    NOPUL3 := ROUND( (WADEST-WANG2) * NIDM3 );
 79    2    END; (*$L+ *)
```

```
Line File Stmt Level
 265  265   80   1   (* ************************************************************************** *)
 266  266   80   1   PROCEDURE GESSPOS(PUL1,PUL3,PUL4: INTEGER; VAR X,Y: REAL);
 267  267   80   1   (* THIS DETERMINE THE CURRENT POSITION OF THE TIP GIVEN THAT
 268  268   80   1       PUL1,PUL3,PUL4 HAVE BEEN APPLIED FROM XSTART,YSTART
 269  269   80   1       WASTART *)
 270  270   80   1   VAR
 271  271   80   2       ALPHA3: REAL;  (* NEW WRISTANGLE AFTER APPLYING PUL3 *)
 272  272   80   2       P,Q: REAL;     (* COORDINATES OF TIP (AFTER APPLYING PUL3)
 273  273   80   2                         IN COORDINATE SYSTEM OF ARM *)
 274  274   80   2       DO: REAL;      (* DISTANCE FROM XSTART,YSTART TO ORIGIN *)
 275  275   80   2       TETAO: REAL;   (* ANGLE BETWEEN LINE JOINING XSTART,YSTART TO
 276  276   80   2                         ORIGIN AND X AXIS. *)
 277  277   80   2       X3,Y3: REAL;   (* COORDINATE OF TIP AFTER APPLYING PUL3 ONLY *)
 278  278   80   2
 279  279   80   2       BETA: REAL;    (* ANGLE BETWEEN SEGMENT DO AND LINE OF ELONGATION
 280  280   80   2                         ALONG MOTOR 4 *)
 281  281   80   2       M4,B4: REAL;   (* COEFFICIENTS OF LINE // TO ELONGATION ALONG
 282  282   80   2                         MOTOR 4,  AND WHICH HAS X3,Y3 AS A POINT. *)
 283  283   80   2       D4: REAL;      (* DISPLACEMENT ALONG MOTOR 4 *)
 284  284   80   2       HYPO1X4,HYPO1Y4,  HYPO2X4,HYPO2Y4: REAL;
 285  285   80   2       X4,Y4: REAL;   (* CORDINATE OF TIP AFTER APPLYING PUL3 AND PUL4 *)
 286  286   80   2       A,D,C: REAL;   (* COEFFICIENTS OF QUADRATIC EQUATION TO SOLVE X4 *)
 287  287   80   2       ANGLE4: REAL;  (* ANGLE OF SEGMENT TIP TO ORIGIN WITH RESPECT
 288  288   80   2                         X AXIS AFTER PUL3,PUL4 *)
 289  289   80   2
 290  290   80   2       FINANGLE: REAL; (* FINAL ANGLE OF TIP RELATIVE TO POSITIVE PART
 291  291   80   2                          OF X AXIS *)
 292  292   80   2
 293  293   80   2
 294  294   80   2       TEST: CHAR;
 295  295   80   2   BEGIN
 296  296   80   2   ALPHA3 := WASTART - PUL3/NIDM3;
 297  297   81   2   P := RAD3 * COS(  (180-ALPHA3)/RADDEG  );
 298  298   82   2   Q := RAD3 * SIN(  (180-ALPHA3)/RADDEG  );
 299  299   83   2   DO := SQRT(  SQR(XSTART) + SQR(YSTART)  );
 300  300   84   2   IF XSTART=0 THEN
 301  301   85   2       TETAO := 90/RADDEG
 302  302   86   2   ELSE
 303  303   86   2       TETAO := ARCTAN(YSTART/XSTART);
 304  304   87   2   X3 := DO*COS(TETAO) + P*COS(TETAO) - Q*SIN(TETAO);
 305  305   88   2   Y3 := DO*SIN(TETAO) + P*SIN(TETAO) + Q*COS(TETAO);
 306  306   89   2
 307  307   89   2
 308  308   89   2   BETA := ARCTAN( SQRT(SHOULDER) / SQRT( SQR(DO)-SHOULDER )  );
 309  309   90   2   M4 := SIN(TETAO+BETA)/COS(TETAO+BETA);
 310  310   91   2   B4 := Y3 - M4*X3;
 311  311   92   2   D4 := PUL4 / NICM4;
 312  312   93   2   A := SQR(M4)+1;
 313  313   94   2   D := 2*( M4*B4 - M4*Y3 - X3);
 314  314   95   2   C := SQR(X3) + SQR(B4) - 2*Y3*B4 + SQR(Y3) - SQR(D4);
 315  315   96   2   (*$L+ *)
```

```
Line File Stmt Level
 316  316   96   2    HYPO1X4 := (-D + SQRT(D*D-4*A*C)) / (2*A);
 317  317   97   2    HYPO2X4 := (-D - SQRT(D*D-4*A*C)) / (2*A);
 318  318   98   2    HYPO1Y4 := M4*HYPO1X4 + B4;
 319  319   99   2    HYPO2Y4 := M4*HYPO2X4 + B4;
 320  320  100   2
 321  321  100   2
 322  322  100   2    IF (SQR(HYPO1X4) + SQR(HYPO1Y4))  >  (SQR(HYPO2X4) + SQR(HYPO2Y4)) THEN
 323  323  101   2        IF PUL4 > 0 THEN
 324  324  102   2            BEGIN
 325  325  103   3            X4 := HYPO1X4;
 326  326  104   3            Y4 := HYPO1Y4;
 327  327  105   3            END
 328  328  106   2        ELSE
 329  329  106   2            BEGIN
 330  330  107   3            X4 := HYPO2X4;
 331  331  108   3            Y4 := HYPO2Y4;
 332  332  109   3            END
 333  333  110   2    ELSE
 334  334  110   2        IF PUL4 < 0 THEN
 335  335  111   2            BEGIN
 336  336  112   3            X4 := HYPO1X4;
 337  337  113   3            Y4 := HYPO1Y4;
 338  338  114   3            END
 339  339  115   2        ELSE
 340  340  115   2            BEGIN
 341  341  116   3            X4 := HYPO2X4;
 342  342  117   3            Y4 := HYPO2Y4;
 343  343  118   3            END;
 344  344  119   2
 345  345  119   2
 346  346  119   2    IF X4=0 THEN
 347  347  120   2        ANGLE4:= 90/RADDEG
 348  348  121   2    ELSE
 349  349  121   2        ANGLE4:= ARCTAN(Y4/X4);
 350  350  122   2
 351  351  122   2    FINANGLE := (-PUL1/NIDN1)/RADDEG  + ANGLE4;
 352  352  123   2    X := SQRT(X4*X4 + Y4*Y4) * COS(FINANGLE);
 353  353  124   2    Y := SQRT(X4*X4 + Y4*Y4) * SIN(FINANGLE);
 354  354  125   2
 355  355  125   2    END;
 356  356  126   1    (*$L+ *)
```

```
Stmt Level
126   1   (* ************************************************************************* *)
126   1   FUNCTION WRISTANG(D1X,D1Y,D2X,D2Y: REAL): REAL;
126   1   (* THIS RETURNS ANGLE (DEGREE) BETWEEN SEGMENTS D0-D1 AND D1-D2, D0
126   1      BEING THE ORIGIN (I.E. X0,Y0 CENTER OF ROTATION OF ARM WHICH MUST HAVE
126   1      BEEN SET).   THE ANGLE IS TAKEN CLOCKWISE FROM D0-D1 TO D1-D2.
126   1      HYPO: ANGLE OF D0-D1 WITH RESPECT TO X AXIS IS BETWEEN 0 AND 180
126   1            EXCLUSIVELY.
126   1            0 <= WRISTANG < 360                                              *)
126   1   VAR
126   2       M1,M2: REAL; (* SLOPES OF SEGMENTS D0-D1, D1-D2 *)
126   2       ARC,INVARC,ANG: REAL; (* TEMPORARIES *)
126   2
126   2   BEGIN
126   2   IF D1X=0 (* CASE M1=INF *) THEN
127   2       IF D2X=0 THEN
128   2           IF D2Y > D1Y THEN WRISTANG := 180
130   2           ELSE WRISTANG := 0
131   2       ELSE
131   2           IF D2Y=D1Y (* CASE M2=0 *) THEN
132   2               IF D2X>0 THEN WRISTANG := 270
134   2               ELSE WRISTANG := 90
135   2           ELSE
135   2               BEGIN
136   3               M2 := (D2Y-D1Y)/(D2X-D1X);
137   3               ARC := RADDEG * ARCTAN(M2); INVARC := RADDEG * ARCTAN(-1/M2);
139   3               IF M2>0 THEN
140   3                   IF D2X>0 THEN ANG := 270-ARC
142   3                   ELSE ANG := 90-ARC
143   3               ELSE
143   3                   IF D2X>0 THEN ANG := 360-INVARC
145   3                   ELSE ANG := 180-INVARC;
146   3               WRISTANG := ANG;
147   3               END
148   2   ELSE
148   2       IF D2X=D1X (* CASE M2=INF *) THEN
149   2           BEGIN
150   3           M1 := D1Y/D1X;
151   3           ARC := RADDEG * ARCTAN(M1); INVARC := RADDEG * ARCTAN(-1/M1);
153   3           IF M1>0 THEN
154   3               IF D2Y > M1*D2X THEN ANG := 90+ARC
156   3               ELSE ANG := 270+ARC
157   3           ELSE
157   3               IF D2Y > M1*D2X THEN ANG := 180+INVARC
159   3               ELSE ANG := INVARC;
160   3           WRISTANG := ANG;
161   3           END
162   2   (*$L+ *)
```

```
Line File Stmt Level
 404  404  162   2        ELSE
 405  405  162   2          BEGIN
 406  406  163   3          M1 := D1Y/D1X;  M2 := (D2Y-D1Y)/(D2X-D1X);
 407  407  165   3          IF M1=M2 THEN
 408  408  166   3             IF SQR(D2X)+SQR(D2Y) > SQR(D1X)+SQR(D1Y) THEN WRISTANG := 180
 409  409  168   3             ELSE WRISTANG := 0
 410  410  169   3          ELSE
 411  411  169   3             BEGIN
 412  412  170   4             IF M1*M2 = -1 THEN ANG := 90
 413  413  172   4             ELSE ANG := RADDEG * ARCTAN( (M1-M2) / (1+M1*M2) );
 414  414  173   4             IF M1>0 THEN
 415  415  174   4                IF D2Y >= M1*D2X THEN
 416  416  175   4                   IF ANG>0 THEN (* ANG := ANG *)
 417  417  176   4                   ELSE ANG := 180+ANG
 418  418  178   4                ELSE
 419  419  178   4                   IF ANG>0 THEN  ANG := 180+ANG
 420  420  180   4                   ELSE ANG := 360+ANG
 421  421  181   4             ELSE
 422  422  181   4                IF D2Y >= M1*D2X THEN
 423  423  182   4                   IF ANG>0 THEN ANG := 180+ANG
 424  424  184   4                   ELSE ANG := 360+ANG
 425  425  185   4                ELSE
 426  426  185   4                   IF ANG>0 THEN (* ANG := ANG *)
 427  427  186   4                   ELSE ANG := 180+ANG;
 428  428  188   4             WRISTANG := ANG;
 429  429  189   4             END;
 430  430  190   3          END;
 431  431  191   2    END;  (* FUNCTION WRISTANG *)
 432  432  192   1    (*$L+ *)
```

```
Stmt Level
192   1   (* ************************************************************************** *)
192   1   PROCEDURE GETMID(D3X,D3Y,D4X,D4Y: REAL; VAR D2X,D2Y: REAL);
192   1   (* RETURN MIDDLE POINT OF SEGMENT DEFINED BY 3 AND 4 *)
192   1   BEGIN
192   2   D2X := D3X + (D4X-D3X)/2;   D2Y := D3Y + (D4Y-D3Y)/2;
194   2   END;
195   1
195   1
195   1
195   1
195   1
195   1   (* ************************************************************************** *)
195   1   PROCEDURE CIRCLE(X1,Y1,X2,Y2,X3,Y3: REAL; VAR CX0,CY0,RADIUS: REAL);
195   1   (* RETURN CENTER AND RADIUS OF CIRCLE DEFINED BY POINTS 1,2 AND 3 *)
195   1   VAR A,D,E,F: REAL;
195   2   BEGIN
195   2   A := X1*(Y2-Y3) + X2*(Y3-Y1) + X3*(Y1-Y2);
196   2
196   2   D := Y1 * ( (X2*X2+Y2*Y2)-(X3*X3+Y3*Y3) )  +
197   2        Y2 * ( (X3*X3+Y3*Y3)-(X1*X1+Y1*Y1) )  +
197   2        Y3 * ( (X1*X1+Y1*Y1)-(X2*X2+Y2*Y2) );
197   2
197   2   E := X1 * ( (X3*X3+Y3*Y3)-(X2*X2+Y2*Y2) )  +
198   2        X2 * ( (X1*X1+Y1*Y1)-(X3*X3+Y3*Y3) )  +
198   2        X3 * ( (X2*X2+Y2*Y2)-(X1*X1+Y1*Y1) );
198   2
198   2   F := X1 *
199   2             ( Y3*(X2*X2+Y2*Y2)-Y2*(X3*X3+Y3*Y3) )  +
199   2        X2 *
199   2             ( Y1*(X3*X3+Y3*Y3)-Y3*(X1*X1+Y1*Y1) )  +
199   2        X3 *
199   2             ( Y2*(X1*X1+Y1*Y1)-Y1*(X2*X2+Y2*Y2) );
199   2
199   2   D := D/(2*A);      E := E/(2*A);      F := F/A;
202   2   CX0 := -D; CY0 := -E;
204   2   RADIUS := SQRT(D*D + E*E - F);
205   2   END;
206   1
206   1
206   1
206   1
206   1   (* ************************************************************************** *)
206   1   FUNCTION DISTANCE(X1,Y1,X2,Y2: REAL): REAL;
206   1   BEGIN
206   2   DISTANCE := ABS(SQRT( SQR(X1-X2) + SQR(Y1-Y2) ));
207   2   END;
208   1   (*$L+ *)
```

```
Stmt Level
208   1   (* ************************************************************************ *)
208   1   PROCEDURE GETEXIT(X1,Y1,X2,Y2,OFFSET: REAL; VAR X,Y: REAL);
208   1   (* GET POINT AT DISTANCE OFFSET FROM POINT 1 AND WHICH BELONG TO LINE
208   1      THROUGH SEGMENT DEFINED BY POINTS 1 AND 2. THIS IS USED 4 TIMES IN
208   1      PROCEDURE WAITCROSS TO DETERMINE THE 4 ACCESS POINTS.              *)
208   1
208   1   VAR
208   2       HYPO1X,HYPO1Y,HYPO2X,HYPO2Y,M,B,A,D,C: REAL;
208   2   BEGIN
208   2   IF Y1=Y2 THEN
209   2       IF DISTANCE(X2,Y2,X1-OFFSET,Y1) > DISTANCE(X2,Y2,X1+OFFSET,Y1)
210   2           THEN BEGIN X:=X1-OFFSET; Y:=Y1; END
214   2           ELSE BEGIN X:=X1+OFFSET; Y:=Y1; END
218   2   ELSE
218   2       IF X1=X2 THEN
219   2           IF DISTANCE(X2,Y2,X1,Y1-OFFSET) > DISTANCE(X2,Y2,X1,Y1+OFFSET)
220   2               THEN BEGIN X:=X1; Y:=Y1-OFFSET; END
224   2               ELSE BEGIN X:=X1; Y:=Y1+OFFSET; END
228   2       ELSE
228   2           BEGIN
229   3           M := (Y2-Y1)/(X2-X1); (* SLOPE 1-2 *)
230   3           B := Y1 - M*X1;
231   3           A := SQR(M) + 1;
232   3           D := 2 * (M*B -X1 -M*Y1);
233   3           C := SQR(Y1) - 2*Y1*B + SQR(B) + SQR(X1) - SQR(OFFSET);
234   3           HYPO1X := ( -D + SQRT( SQR(D) - 4*A*C ) ) / (2*A);
235   3           HYPO2X := ( -D - SQRT( SQR(D) - 4*A*C ) ) / (2*A);
236   3           HYPO1Y := M*HYPO1X + B;
237   3           HYPO2Y := M*HYPO2X + B;
238   3
238   3
238   3           IF DISTANCE(X2,Y2,HYPO1X,HYPO1Y) > DISTANCE(X2,Y2,HYPO2X,HYPO2Y)
239   3               THEN BEGIN X:=HYPO1X; Y:=HYPO1Y; END
243   3               ELSE BEGIN X:=HYPO2X; Y:=HYPO2Y; END;
247   3
247   3           END;
248   2   END;
249   1   (*$L+ *)
```

```
File Stmt Level
518  249  1   (* ****************************************************************************** *)
519  249  1   PROCEDURE CALIB;
520  249  1   (* MAKE THIS AT LEVEL OF CROSS (I.E. ABOVE BLACK PLANE) *)
521  249  1   (* THIS PROC SETS THE GLOBAL VARIABLES: CORX,CORY *)
522  249  1   VAR
523  249  2       L1X,L1Y,  L2X,L2Y: INTEGER;
524  249  2       L: ARRAY[1..6,1..6] OF REAL;
525  249  2       M1,B1,  M2,B2: REAL;
526  249  2       I,J: INTEGER;
527  249  2       C: CHAR;
528  249  2
529  249  2   BEGIN
530  249  2   HOME; WRITE('ENTERING CALIBRER:  TYPE CR WHEN SATISFIED WITH PIC');
531  251  2   PARAM1.KEY := 0;
532  252  2   WHILE PARAM1.KEY = 0 DO GETSHOW;
533  254  2
534  254  2   PARAM1.KEY := 0;
535  255  2   HOME;
536  256  2
537  256  2   DETDOTS(DOT,FR2);
538  257  2   I := 1;
539  258  2   WHILE DOT[I].ROW <> -1 DO
540  259  3       BEGIN
541  260  4       BLINK(DOT[I]);
542  261  4       I := I + 1;
543  262  4       END;
544  263  2   WRITELN(I-1:2,' DOTS');
545  264  2
546  264  2   L[1,2] := 71.0;     L[1,3] := 37.0;
547  266  2
548  266  2          L1X := ABS( DOT[2].COL-DOT[1].COL );
549  267  2          L1Y := ABS( DOT[2].ROW-DOT[1].ROW );
550  268  2          L2X := ABS( DOT[1].COL-DOT[3].COL );
551  269  2          L2Y := ABS( DOT[1].ROW-DOT[3].ROW );
552  270  2
553  270  2          M1 := -SQR(L1X/L1Y);
554  271  2          M2 := -SQR(L2X/L2Y);
555  272  2          B1 :=  SQR(L[1,2]/L1Y);
556  273  2          B2 :=  SQR(L[1,3]/L2Y);
557  274  2
558  274  2          CORX := (B2-B1)/(M1-M2);
559  275  2          CORY := SQRT( (M1*CORX) + B1 );
560  276  2          CORX := SQRT(CORX);
561  277  2
562  277  2   WRITELN('CORRECTION FACTOR ALONG X= ',CORX:12:10, ' cm/pix');
563  278  2   WRITELN('CORRECTION FACTOR ALONG Y= ',CORY:12:10, ' cm/pix');
564  279  2   WRITE('LEAVING CALIBRER, TYPE CR '); READLN(C); HOME;
565  282  2   END;
566  283  1   (*$L+ *)
```

```
(* ************************************************************************* *)
PROCEDURE MOVEARM(MOTOR: MOTTYPE; STEPCNT: INTEGER);
(* THIS WILL MOVE THE SPECIFIED MOTOR FOR THE SPECIFIED NUMBER OF STEPS
   (+/-). IT USES/UPDATES THE GLOBAL VARIABLES MSTATE AND MPULSE, AND USES
   GLOBAL MOTHOLD.  IT RETURNS PARAM.KEYIN=0 IF NO KEY WERE PRESSED
   DURING THE ACTIVATION OR IT RETURNS PARAM.KEYIN= KEY IN LSB.

                 MOTOR #     DESCRIPTION            + DIR        - DIR
                 -----------------------------------------------------
                    0: ALL MOTORS DISABLED
                    1: HEAD ROTATE SEEN FROM TOP...   CW          CCW
                    2: ARM  TOWARD................. FLOOR       SEILING
                    3: WRIST ROTATE FACING IT......   CW          CCW
                    4: EXTEND......................  OUT           IN
                    5: GRIPPER..................... CLOSE         OPEN
                    6: WRIST PIVOT TOWARD.......... FLOOR       SEILING

OUTPUT BYTE:    BITS
                  7: IF 1 ENABLES MOTORS 1 TO 3
                  6: IF 1 ENABLES MOTORS 4 TO 6
                  5 AND 4: CONTROL MOTOR 3 IF BIT 7 = 1
                           CONTROL MOTOR 6 IF BIT 6 = 1
                  3 AND 2: CONTROL MOTOR 2 IF BIT 7 = 1
                           CONTROL MOTOR 5 IF BIT 6 = 1
                  1 AND 0: CONTROL MOTOR 1 IF BIT 7 = 1
                           CONTROL MOTOR 4 IF BIT 6 = 1

              TO OBTAIN + MOVES( FORWARD) THE FOLLOWING SEQUENCE
           MUST BE SENT TO THE 2 BITS CONTROLLING THE ADDRESSED MOTOR

                      DECIMAL        BINARY
                                 MS BIT  LS BIT
                      ---------------------------
                         0          0       0
                         2          1       0
                         3          1       1
                         1          0       1

           NOTES: ASSUMING THE 2 BITS ARE IN STATE 1 (01) THE
                  ABOVE SEQUENCE WOULD COMMAND 4 STEPS FORWARD:
                  1->0->2->3->1
                  A PATTERN IS HELD MOTHOLD MSEC.
                  TO OBTAIN A - (BACKWARD) MOVE, IN ANY STATE,
                  THE INVERSE SEQUENCE MUST BE ASSERTED. FOR EX
                  3->2->0->1->3->2 WOULD COMMAND 5 STEPS BACKWARD,
                  ASSUMING THE STATE IS 3 BEFORE THE ASSERTION OF THE
                  SEQUENCE.  *)
(*$L+ *)
```

```
Stmt Level
283   1   VAR
283   2       CURRENT: STATYPE;
283   2       OUTBYTE: BYTE;    I,KEY: INTEGER;
283   2   BEGIN
283   2   KEY := 0;
284   2   IF MOTOR = 0 THEN
285   2       BEGIN
286   3       PARAM.CTRLOUT := 0;
287   3       PARAM.HOLDTIME := MOTHOLD[0];
288   3       ARMOUT(PARAM);
289   3       IF PARAM.KEYIN <> 0 THEN KEY := PARAM.KEYIN;
291   3       END
292   2   ELSE
292   2       FOR I:= 1 TO ABS(STEPCNT) DO
293   3           BEGIN
294   4           CURRENT := MSTATE[MOTOR];
295   4           IF STEPCNT>0 THEN
296   4               BEGIN
297   5               MPULSE[MOTOR] := MPULSE[MOTOR] + 1;
298   5               CASE CURRENT OF
299   5                   0: MSTATE[MOTOR] := 2;
300   5                   2: MSTATE[MOTOR] := 3;
301   5                   3: MSTATE[MOTOR] := 1;
302   5                   1: MSTATE[MOTOR] := 0;
303   5                   END;
303   5               END
304   4           ELSE
304   4               BEGIN
305   5               MPULSE[MOTOR] := MPULSE[MOTOR] - 1;
306   5               CASE CURRENT OF
307   5                   1: MSTATE[MOTOR] := 3;
308   5                   3: MSTATE[MOTOR] := 2;
309   5                   2: MSTATE[MOTOR] := 0;
310   5                   0: MSTATE[MOTOR] := 1;
311   5                   END;
311   5               END;
312   4
312   4           IF MOTOR <= 3 THEN
313   4               OUTBYTE := 128 + MSTATE[1] + MSTATE[2]*4 + MSTATE[3]*16
314   4           ELSE
314   4               OUTBYTE := 64  + MSTATE[4] + MSTATE[5]*4 + MSTATE[6]*16;
315   4           PARAM.CTRLOUT := OUTBYTE;
316   4           PARAM.HOLDTIME := MOTHOLD[MOTOR];
317   4           ARMOUT(PARAM);
318   4           IF PARAM.KEYIN <> 0 THEN KEY := PARAM.KEYIN;
320   4           END;
321   2   PARAM.KEYIN := KEY;
322   2   END;
323   1   (*$L+ *)
```

```
Stmt Level
323  1   (* ********************************************************************** *)
323  1   PROCEDURE INITMOT;
323  1   BEGIN
323  2   FOR I := 1 TO 6 DO MSTATE[I] := 0; (* ALL MOTORS IN STATE 0 *)
325  2   FOR I := 1 TO 6 DO MPULSE[I] := 0; (* ALL MOTORS HAD 0 PULSES *)
327  2
327  2   PARAM.CTRLOUT := 128;  (* ENABLE MOTORS 1,2 AND 3 AND PUT IN STATE 0 *)
328  2   PARAM.HOLDTIME := MOTHOLD[0];
329  2   ARMOUT(PARAM);        (* ACTUAL WRITING TO I/O PORT *)
330  2   PARAM.CTRLOUT :=  64;  (* ENABLE MOTORS 4,5 AND 6 AND PUT IN STATE 0 *)
331  2   PARAM.HOLDTIME := MOTHOLD[0];
332  2   ARMOUT(PARAM);        (* ACTUAL WRITING TO I/O PORT *)
333  2   PARAM.CTRLOUT := 0;
334  2   PARAM.HOLDTIME := MOTHOLD[0];
335  2   ARMOUT(PARAM);        (* DISABLE ALL MOTORS *)
336  2   END;
337  1
337  1
337  1
337  1   (* ********************************************************************** *)
337  1   FUNCTION DIST(X1,Y1,X2,Y2,LENGTH,TOL: REAL): BOOLEAN;
337  1   (* DETERMINE IF POINTS 1, 2 ARE AT DISTANCE LENGTH +/- TOL.  *)
337  1   VAR L: REAL;
337  2   BEGIN
337  2   L := SQRT( SQR(X1-X2) + SQR(Y1-Y2) );
338  2   IF ABS(L-LENGTH) <= TOL THEN DIST := TRUE
340  2   ELSE DIST := FALSE;
341  2   END;
342  1   (*$L+ *)
```

```
Stmt Level
342   1    (* ************************************************************************* *)
342   1    PROCEDURE FINDPOS(VAR CXO,CYO,XOTIP,YOTIP: REAL);
342   1    VAR
342   2        NODOT,DOTCNT,I,J: INTEGER;
342   2        X,Y,U,V: REALARRAY;
342   2        EXIST: ARRAY[1..MAXDOT] OF BOOLEAN;
342   2        GESSX,GESSY,MINDIST,CURDIST: REAL;
342   2        MINIDX: INTEGER;
342   2        ONEMOVE: INTEGER;
342   2
342   2    BEGIN
342   2    ONEMOVE := 320; (* # OF PULSES TO MOVE APPROX. 80 DEGREES *)
343   2               HOME; WRITE('DETERMINING CURRENT POSITION');
345   2               (* GRIPPER CLOSED, NORMAL HEIGHT *)
345   2               MOVEARM(3,-2*ONEMOVE);
346   2               NODOT := 0;
347   2               WHILE NODOT <> 3 DO
348   3                  BEGIN
349   4                  DOTCNT := MAXINT;
350   4                  WHILE DOTCNT > 12 DO
351   5                     BEGIN
352   6                     GETSHOW;
353   6                     DETDOTS(DOT,FR2); DOTCNT := 1;
355   6                     WHILE DOT[DOTCNT].ROW <> -1 DO
356   7                        BEGIN
357   8                        TEMP := DOT[DOTCNT].COL * CORX; U[DOTCNT] := TEMP;
359   8                        TEMP := -DOT[DOTCNT].ROW * CORY; V[DOTCNT] := TEMP;
361   8                        TRANSLAT(U[DOTCNT],V[DOTCNT]);
362   8                        DOTCNT := DOTCNT + 1;
363   8                        END;
364   6                     DOTCNT := DOTCNT - 1;
365   6                     END;
366   4
366   4                  FOR I := 1 TO DOTCNT DO EXIST[I] := TRUE;
368   4                  FOR I := 1 TO DOTCNT-1 DO
369   5                     FOR J := I+1 TO DOTCNT DO
370   6                        IF EXIST[I] AND EXIST[J] THEN
371   6                           IF DISTANCE(U[I],V[I],U[J],V[J])< DBLDOT THEN
372   6                              BEGIN
373   7                              EXIST[J] := FALSE;
374   7                              GETMID(U[I],V[I],U[J],V[J],U[I],V[I]);
375   7                              END;
376   4                  J := 0;
377   4                  FOR I := 1 TO DOTCNT DO
378   5                     IF EXIST[I] THEN
379   5                        BEGIN
380   6                        J := J + 1;
381   6                        U[J] := U[I];
382   6                        V[J] := V[I];
383   6                        END;
384   4    (*$L+ *)
```

```
File Stmt Level
 744  384   4                  DOTCNT := J; (* J MUST BE 1 *)
 745  385   4                  IF DOTCNT > 1 THEN
 746  386   4                     (* MORE THAN 1 DOT; DETERMINE MOST PLAUSIBLE *)
 747  386   4                     BEGIN
 748  387   5                     GESSPOS(MPULSE[1],MPULSE[3],MPULSE[4],GESSX,GESSY);
 749  388   5                     MINDIST := MAXINT; MINIDX := 0;
 750  390   5                     FOR I:=1 TO DOTCNT DO
 751  391   6                        BEGIN
 752  392   7                        CURDIST := DISTANCE(GESSX,GESSY,U[I],V[I]);
 753  393   7                        IF CURDIST < MINDIST THEN
 754  394   7                           BEGIN MINDIST := CURDIST; MINIDX := I; END;
 755  398   7                        END;
 756  399   5                     U[1] := U[MINIDX];  V[1] := V[MINIDX];
 757  401   5                     END;
 758  402   4
 759  402   4                  NODOT := NODOT + 1;
 760  403   4                  X[NODOT] := U[1]; Y[NODOT] := V[1];
 761  405   4                  IF NODOT <> 3 THEN MOVEARM(3,ONEMOVE);
 762  407   4                  END; (* WHILE NODOT... *)
 763  408   2
 764  408   2               XOTIP := X[3]; YOTIP := Y[3];
 765  410   2               CIRCLE(X[1],Y[1],X[2],Y[2],X[3],Y[3],CXO,CYO,TEMP);
 766  411   2               HOME;
 767  412   2         END; (* FINDPOS *)
 768  413   1         (*$L+ *)
```

```
Line File Stmt Level
 769  769  413   1    (* ****************************************************************************** *)
 770  770  413   1    PROCEDURE LOADPARM;
 771  771  413   1    (* INSTALL PARAMETERS FROM FILE *)
 772  772  413   1    VAR I: INTEGER;
 773  773  413   2    BEGIN
 774  774  413   2    HOME;
 775  775  414   2    WRITE('LOADING B:CAMPAS.DAT ');
 776  776  415   2    RESET('B:CAMPAS.DAT',PARMF);
 777  777  416   2    READLN(PARMF,CORX);
 778  778  417   2    READLN(PARMF,CORY);
 779  779  418   2    READLN(PARMF,X0);
 780  780  419   2    READLN(PARMF,Y0);
 781  781  420   2    READLN(PARMF,NIDM1);
 782  782  421   2    READLN(PARMF,RAD3);
 783  783  422   2    READLN(PARMF,NIDM3);
 784  784  423   2    READLN(PARMF,SHOULDER);
 785  785  424   2    READLN(PARMF,NICM4);
 786  786  425   2    FOR I := 0 TO 6 DO READLN(PARMF,MOTHOLD[I]);
 787  787  427   2    READLN(PARMF,XSTART);
 788  788  428   2    READLN(PARMF,YSTART);
 789  789  429   2    READLN(PARMF,WASTART);
 790  790  430   2    READLN(PARMF,XOTIP);
 791  791  431   2    READLN(PARMF,YOTIP);
 792  792  432   2    READLN(PARMF,UPM2);
 793  793  433   2    READLN(PARMF,PICKM2);
 794  794  434   2    READLN(PARMF,NORMM2);
 795  795  435   2    READLN(PARMF,CLOSE5);
 796  796  436   2    READLN(PARMF,GRIP3);
 797  797  437   2    FOR I:= 1 TO 6 DO READLN(PARMF,MINPUL[I]);
 798  798  439   2    FOR I:= 1 TO 6 DO READLN(PARMF,MAXPUL[I]);
 799  799  441   2    READLN(PARMF,FAR);
 800  800  442   2    READLN(PARMF,TOLFAR);
 801  801  443   2    READLN(PARMF,NEAR);
 802  802  444   2    READLN(PARMF,TOLNEAR);
 803  803  445   2    READLN(PARMF,STANDBY);
 804  804  446   2    READLN(PARMF,DROPCROSS);
 805  805  447   2    HOME;
 806  806  448   2    END;
 807  807  449   1    (*$L+ *)
```

```
Stmt Level
 449   1    (* ****************************************************************************** *)
 449   1    PROCEDURE STORPARM;
 449   1    (* LOAD PARAMETERS FROM FILES *)
 449   1    VAR I: INTEGER;
 449   2    BEGIN
 449   2    WRITE('SAVING B:CAMPAS.DAT ');
 450   2    REWRITE('B:CAMPAS.DAT',PARMF);
 451   2    WRITELN(PARMF,CORX);
 452   2    WRITELN(PARMF,CORY);
 453   2    WRITELN(PARMF,X0);
 454   2    WRITELN(PARMF,Y0);
 455   2    WRITELN(PARMF,NIDM1);
 456   2    WRITELN(PARMF,RAD3);
 457   2    WRITELN(PARMF,NIDM3);
 458   2    WRITELN(PARMF,SHOULDER);
 459   2    WRITELN(PARMF,NICM4);
 460   2    FOR I := 0 TO 6 DO WRITELN(PARMF,MOTHOLD[I]);
 462   2    WRITELN(PARMF,XSTART);
 463   2    WRITELN(PARMF,YSTART);
 464   2    WRITELN(PARMF,WASTART);
 465   2    WRITELN(PARMF,XOTIP);
 466   2    WRITELN(PARMF,YOTIP);
 467   2    WRITELN(PARMF,UPM2);
 468   2    WRITELN(PARMF,PICKM2);
 469   2    WRITELN(PARMF,NORMM2);
 470   2    WRITELN(PARMF,CLOSE5);
 471   2    WRITELN(PARMF,GRIP3);
 472   2    FOR I:= 1 TO 6 DO WRITELN(PARMF,MINPUL[I]);
 474   2    FOR I:= 1 TO 6 DO WRITELN(PARMF,MAXPUL[I]);
 476   2    WRITELN(PARMF,FAR);
 477   2    WRITELN(PARMF,TOLFAR);
 478   2    WRITELN(PARMF,NEAR);
 479   2    WRITELN(PARMF,TOLNEAR);
 480   2    WRITELN(PARMF,STANDBY);
 481   2    WRITELN(PARMF,DROPCROSS);
 482   2    RESET('CON:',PARMF);
 483   2    END;
 484   1    (*$L+ *)
```

```
Stmt Level
484   1    (* ******************************************************************************* *)
484   1    PROCEDURE WAITCROSS(VAR CROSSX,CROSSY,EX,EY,WRISTA: REALARRAY);
484   1    (* WAIT FOR CROSS TO APPEAR IN CAMERA FIELD AND RETURN 4 DOTS ON CROSS +
484   1       4 ACCESS POINTS + 4 CORRESPONDING WRIST ANGLES *)
484   1    VAR
484   2        I,J,K: INTEGER;
484   2        X,Y,U,V: REALARRAY;
484   2        POSSI,EXIST: ARRAY[1..MAXDOT] OF BOOLEAN;
484   2        FAR1,NEAR1,POSSCNT,DOTCNT: INTEGER;
484   2    BEGIN
484   2
484   2                POSSCNT := 0;
485   2                WHILE POSSCNT < 4 DO
486   3                   BEGIN
487   4                   DOTCNT := 0;
488   4                   WHILE (DOTCNT<4) OR (DOTCNT>12) DO
489   5                      BEGIN
490   6                      GETSHOW;
491   6                      DETDOTS(DOT,FR2); DOTCNT := 1;
493   6                      WHILE DOT[DOTCNT].ROW <> -1 DO
494   7                         BEGIN
495   8                         TEMP := DOT[DOTCNT].COL * CORX; X[DOTCNT] := TEMP;
497   8                         TEMP := -DOT[DOTCNT].ROW * CORY; Y[DOTCNT] := TEMP;
499   8                         TRANSLAT(X[DOTCNT],Y[DOTCNT]);
500   8                         DOTCNT := DOTCNT + 1;
501   8                         END;
502   6                      DOTCNT := DOTCNT - 1;
503   6                      END;
504   4                   FOR I := 1 TO DOTCNT DO EXIST[I] := TRUE;
506   4                   FOR I := 1 TO DOTCNT-1 DO
507   5                      FOR J := I+1 TO DOTCNT DO
508   6                         IF EXIST[I] AND EXIST[J] THEN
509   6                            IF DISTANCE(X[I],Y[I],X[J],Y[J]) < DBLDOT THEN
510   6                               BEGIN
511   7                               EXIST[J] := FALSE;
512   7                               GETMID(X[I],Y[I],X[J],Y[J],X[I],Y[I]);
513   7                               END;
514   4                   J := 0;
515   4                   FOR I := 1 TO DOTCNT DO
516   5                      IF EXIST[I] THEN
517   5                         BEGIN
518   6                         J := J + 1;
519   6                         X[J] := X[I];
520   6                         Y[J] := Y[I];
521   6                         END;
522   4                   DOTCNT := J;
523   4
523   4    (*$L+ *)
```

```
Line File Stmt Level
 894  894  523   4           FOR I:= 1 TO DOTCNT DO
 895  895  524   5              BEGIN
 896  896  525   6              POSSI[I] := TRUE;
 897  897  526   6              FAR1 := 0; NEAR1 := 0;
 898  898  528   6              FOR J := 1 TO DOTCNT DO
 899  899  529   7                 BEGIN
 900  900  530   8                 IF DIST(X[I],Y[I],X[J],Y[J],FAR,TOLFAR) THEN
 901  901  531   8                    FAR1 := FAR1 + 1
 902  902  532   8                 ELSE
 903  903  532   8                    IF DIST(X[I],Y[I],X[J],Y[J],NEAR,TOLNEAR)
 904  904  533   8                       THEN NEAR1 := NEAR1 + 1;
 905  905  534   8                 END;
 906  906  535   6              IF (FAR1=0) OR (NEAR1 < 2) THEN
 907  907  536   6                 POSSI[I] := FALSE;
 908  908  537   6              END;
 909  909  538   4           POSSCNT := 0;
 910  910  539   4           FOR J := 1 TO DOTCNT DO
 911  911  540   5              IF POSSI[J] THEN POSSCNT := POSSCNT + 1;
 912  912  542   4           END; (* WHILE POSSCNT < ... *)
 913  913  543   2        J := 1;
 914  914  544   2        FOR I := 1 TO DOTCNT DO
 915  915  545   3           IF POSSI[I] THEN
 916  916  546   3              BEGIN
 917  917  547   4              X[J] := X[I];
 918  918  548   4              Y[J] := Y[I];
 919  919  549   4              J := J + 1;
 920  920  550   4              END;
 921  921  551   2        FOR I := 2 TO POSSCNT DO
 922  922  552   3           IF DIST(X[1],Y[1],X[I],Y[I],FAR,TOLFAR) THEN
 923  923  553   3              BEGIN
 924  924  554   4              U[1] := X[1]; V[1] := Y[1];
 925  925  556   4              U[2] := X[I]; V[2] := Y[I];
 926  926  558   4              K := I;
 927  927  559   4              END;
 928  928  560   2        FOR I := 2 TO POSSCNT DO
 929  929  561   3           FOR J := I+1 TO POSSCNT DO
 930  930  562   4              IF DIST(X[I],Y[I],X[J],Y[J],FAR,TOLFAR) AND
 931  931  563   4                 (I<>1) AND (I<>K) AND (J<>1) AND (J<>K) THEN
 932  932  563   4                    BEGIN
 933  933  564   5                    U[3] := X[I]; V[3] := Y[I];
 934  934  566   5                    U[4] := X[J]; V[4] := Y[J];
 935  935  568   5                    END;
 936  936  [illegible]   2
 937  937  [illegible]   2  (*$L+ *)
```

```
Line File Stmt Level
 938  938  569    2            FOR I:= 1 TO 4 DO
 939  939  570    3              BEGIN
 940  940  571    4              CROSSX[I] := U[I];  CROSSY[I] := V[I];
 941  941  573    4              IF ODD(I) THEN
 942  942  574    4                BEGIN
 943  943  575    5                GETEXT(U[I],V[I],U[I+1],V[I+1],RAD3-GRIP3,
 944  944  576    5                                                 EX[I],EY[I]);
 945  945  576    5                WRISTA[I] := WRISTANG(EX[I],EY[I],U[I+1],V[I+1]);
 946  946  577    5                END
 947  947  578    4              ELSE
 948  948  578    4                BEGIN
 949  949  579    5                GETEXT(U[I],V[I],U[I-1],V[I-1],RAD3-GRIP3,
 950  950  580    5                                                 EX[I],EY[I]);
 951  951  580    5                WRISTA[I] := WRISTANG(EX[I],EY[I],U[I-1],V[I-1]);
 952  952  581    5                END;
 953  953  582    4              END;
 954  954  583    2      END;
 955  955  584    1      (*$L+ *)
```

```
Stmt Level
584   1    (* ****************************************************************************** *)
584   1    PROCEDURE ARMMOVE;
584   1    (* MAIN PROCEDURE PERMITTING SETTINGS OF PROGRAM PARAMETERS AND IN WHICH
584   1       THE PICK AND PLACE (CROSS) TASK IS ACOMPLISHED *)
584   1
584   1    VAR
584   2        CENTERX,CENTERY: REAL;  (* CENTER OF ROTATION *)
584   2        RADIUS: REAL;
584   2        ONEOR3: CHAR;
584   2        S1,S2,S3,SINE,COSINE,TANG,ANGLE: REAL;
584   2        NIMPD: REAL;
584   2        STEPCNT,NOPUL1,NOPUL3,NOPUL4: INTEGER;
584   2        I,J,K: INTEGER;
584   2        X,Y,U,V: REALARRAY;
584   2        XI,YI: ARRAY[1..MAXDOT] OF INTEGER;
584   2        XNOW,YNOW,XTIPNOW,YTIPNOW,WANOW: REAL;
584   2        MUSTSHOW: BOOLEAN;
584   2        DOTCNT: INTEGER;
584   2        MOTORNO: MOTTYPE;
584   2        CHARIN: CHAR;
584   2
584   2        MINIDX: INTEGER;
584   2        WANGLES: REALARRAY;
584   2
584   2        MINDIST,DIST: REAL;
584   2        FOUND,NOCROSS: BOOLEAN;
584   2
584   2        XBAR,YBAR,SLOPE,SUMXY,SUMX2,INTX,INTY: REAL;
584   2    BEGIN
584   2    HOME; WRITELN('ENTERING ARM MODE');
586   2    WRITE('GET PARMS FROM FILE? Y/N? '); READLN(C);
588   2    IF C = 'Y' THEN LOADPARM
590   2    ELSE
590   2        BEGIN
591   3        XO := 0.0; YO := 0.0;
593   3        HOME;
594   3        END;
595   2
595   2
595   2    (*$L+ *)
```

```
Stmt Level
 595   2   MOTORNO := 0;
 596   2   MUSTSHOW := TRUE;
 597   2   CHARIN := ' ';
 598   2   WHILE CHARIN <> 'Z' DO
 599   3       BEGIN
 600   4       MOVEARM(MOTORNO,STEPCNT);
 601   4       IF ( (MOTORNO=0) AND MUSTSHOW ) THEN
 602   4    BEGIN
 603   5           GETSHOW;
 604   5           MUSTSHOW := FALSE;
 605   5   END;
 606   4
 606   4       CHARIN := CHR(PARAM.KEYIN MOD 256);
 607   4       CASE CHARIN OF
 608   4           '=':
 608   4               BEGIN
 609   5               HOME; WRITE('NIDM3? '); READLN(NIDM3);
 612   5               HOME;
 613   5               END;
 614   4           'P': (* DISPLAY THE NUMBER OF PULSES EACH MOTOR HAS RECEIVED *)
 614   4               BEGIN
 615   5               HOME; WRITELN('PULSES ON EACH MOTOR:');
 617   5               FOR I := 1 TO 6 DO
 618   6                   WRITE('     ',I:2,'     ');
 619   5               WRITELN;
 620   5               FOR I := 1 TO 6 DO
 621   6                   WRITE(MPULSE[I]:10);
 622   5               WRITELN; WRITELN;
 624   5               WRITE('TYPE CR '); READLN(C); HOME;
 627   5               END;
 628   4           '>':
 628   4               BEGIN
 629   5               HOME; WRITELN('RESETTING ORIGIN TO 0,0 ');
 631   5               X0 := 0.0;   Y0 := 0.0;
 633   5               WRITE('TYPE CR '); READLN(C); HOME;
 636   5               END;
 637   4           'Y': (* SET PARAMETERS STANDBY AND DROPCROSS *)
 637   4               BEGIN
 638   5               HOME; WRITE('STANDBY DROPCROSS? ');
 640   5               READLN(STANDBY,DROPCROSS);
 641   5               HOME;
 642   5               END;
 643   4           '[':
 643   4               BEGIN
 644   5               HOME; WRITE('SHOULDER? '); READLN(SHOULDER); HOME;
 648   5               END;
 649   4   (*$L+ *)
```

```
Stat Level
649  4      '?': (* TEST PROCEDURE GESSPOS *)
649  4         BEGIN
650  5         GESSPOS(MPULSE[1],MPULSE[3],MPULSE[4],X[1],Y[1]);
651  5         BLINK2(X[1],Y[1]);
652  5         END;
653  4      '+': (* TEST PROCEDURES WAITCROSS AND GETEXT *)
653  4         BEGIN
654  5         HOME; WRITE('WAITING FOR CROSS');
656  5         WAITCROSS(U,V,X,Y,WANGLES); HOME;
658  5         FOR I := 1 TO 4 DO
659  6             BEGIN
660  7             HOME; WRITE('EXTERNAL DOT ',I:3);
662  7             WRITE('     WRISTA= ',WANGLES[I]:8:3);
663  7             BLINK2(X[I],Y[I]);
664  7             END;
665  5         HOME;
666  5         END;
667  4      'T': (* SET CAMERA EXPOSURE TIME *)
667  4         BEGIN
668  5         HOME;
669  5         WRITELN('CURRENT EXP TIME IN MSEC=',EXPTIME);
670  5         WRITE('ENTER EXP TIME IN MSEC=');
671  5         READLN(EXPTIME);
672  5         HOME;
673  5         END;
674  4      'C': CALIB; (* DETERMINE CORX AND CORY *)
675  4      'S': (* STOP MOTOR AND GET PICTURE *)
675  4         BEGIN
676  5         MOTORNO := 0;
677  5         MUSTSHOW := TRUE;
678  5         END;
679  4      'H': (* STEP MOTOR HOLD TIME *)
679  4         BEGIN
680  5         HOME; FOR I:=0 TO 6 DO WRITE(MOTHOLD[I]:3); WRITELN;
684  5         WRITE('MOTOR # 0..6? '); READLN(I);
686  5         WRITE('HOLD TIME (MSEC)? '); READLN(MOTHOLD[I]);
688  5         WRITELN;
689  5         FOR I:=0 TO 6 DO WRITE(MOTHOLD[I]:3); READLN(C);
692  5         HOME;
693  5         END;
694  4      'I': (* INIT COUNTERS, USED ONLY DURING CALIBRATION *)
694  4         BEGIN
695  5         HOME; INITMOT; WRITE('INITMOT, TYPE CR '); READLN(C);
699  5         DOTCNT := 1;
700  5         HOME;
701  5         END;
702  4  (*$L+ *)
```

```
Line File Stmt Level
1090 1090  702   4         '1','2','3','4','5','6': (* MOVE A MOTOR IN POSITIVE DIRECTION *)
1091 1091  702   4           BEGIN
1092 1092  703   5           MOTORNO := ORD(CHARIN) - 48;
1093 1093  704   5           STEPCNT := +1;
1094 1094  705   5           END;
1095 1095  706   4          '!','@','#','$','%','^': (* MOVE A MOTOR IN NEGATIVE DIRECTION *)
1096 1096  706   4           BEGIN
1097 1097  707   5           STEPCNT := -1;
1098 1098  708   5           IF CHARIN = '!' THEN MOTORNO := 1;
1099 1099  710   5           IF CHARIN = '@' THEN MOTORNO := 2;
1100 1100  712   5           IF CHARIN = '#' THEN MOTORNO := 3;
1101 1101  714   5           IF CHARIN = '$' THEN MOTORNO := 4;
1102 1102  716   5           IF CHARIN = '%' THEN MOTORNO := 5;
1103 1103  718   5           IF CHARIN = '^' THEN MOTORNO := 6;
1104 1104  720   5           END;
1105 1105  721   4          'R': (* INIT DOT COUNTER *)
1106 1106  721   4           BEGIN
1107 1107  722   5           HOME; WRITE('DOTCNT := 1;  TYPE CR '); READLN(C);
1108 1108  725   5           DOTCNT := 1;
1109 1109  726   5           HOME;
1110 1110  727   5           END;
1111 1111  728   4          '.': (* REGISTER A SINGLE DOT *)
1112 1112  728   4           BEGIN
1113 1113  729   5           HOME;
1114 1114  730   5           DETDOTS(DOT,FR2);
1115 1115  731   5           COORD.ROW := DOT[1].ROW; COORD.COL := DOT[1].COL;
1116 1116  733   5           I := COORD.COL; J := COORD.ROW;
1117 1117  735   5           TEMP := I*CORX;
1118 1118  736   5           X[DOTCNT] := TEMP;
1119 1119  737   5           TEMP := -J*CORY;
1120 1120  738   5           Y[DOTCNT] := TEMP;
1121 1121  739   5           TRANSLAT(X[DOTCNT],Y[DOTCNT]);
1122 1122  740   5           WRITE('DOT ',DOTCNT:2,' BLINKING  ');
1123 1123  741   5           WRITE('   (',X[DOTCNT]:8:3,',',Y[DOTCNT]:8:3,')');
1124 1124  742   5           BLINK2(X[DOTCNT],Y[DOTCNT]);
1125 1125  743   5           DOTCNT := DOTCNT+1;
1126 1126  744   5           GETSHOW;
1127 1127  745   5           HOME;
1128 1128  746   5           END;
1129 1129  747   4     (*$L+ *)
```

```
Stat Level
747  4    'E': (* DETERMINE X0,Y0 AND NIDM1 OR RAD3 AND NIDM3 *)
747  4         BEGIN
748  5         HOME; WRITELN('COMPUTING CIRCLE EQUATION ');
750  5         CIRCLE(X[1],Y[1],X[2],Y[2],X[3],Y[3],CENTERX,CENTERY,RADIUS);
751  5
751  5         WRITE(' CENTER AT (',CENTERX:8:4,',',CENTERY:8:4,')      ' );
752  5         WRITELN(' RADIUS= ',RADIUS:8:4);
753  5
753  5         S1 := SQRT( SQR(X[1]-X[3]) + SQR(Y[1]-Y[3]) );
754  5         S2 := SQRT( SQR(X[1]-CENTERX) + SQR(Y[1]-CENTERY) );
755  5         S3 := SQRT( SQR(X[3]-CENTERX) + SQR(Y[3]-CENTERY) );
756  5         COSINE := (S1*S1 - S2*S2 - S3*S3)/ (-2*S2*S3);
757  5                 (* COS OF ANGLE BETHWEEN S2 AND S3 *)
757  5         SINE := SQRT(1 -COSINE*COSINE);
758  5         TANG := SINE/COSINE;
759  5         ANGLE := RADDEG * ARCTAN(TANG); (* DEGREE *)
760  5         IF ANGLE < 0 THEN ANGLE := ANGLE + 180;
762  5         WRITELN('ANGLE COVERED=',ANGLE:8:4,' DEGREES    ');
763  5
763  5         WRITE('PARMS FOR MOTOR 1 OR 3? '); READLN(ONEOR3);
765  5         IF ONEOR3 = '1' THEN
766  5            BEGIN
767  6            X0 := CENTERX;
768  6            Y0 := CENTERY;
769  6            NIDM1 := ABS(NPULSE[1] / ANGLE);
770  6            END
771  5         ELSE
771  5            BEGIN
772  6            RAD3 := RADIUS;
773  6            NIDM3 := ABS(NPULSE[3] / ANGLE);
774  6            END;
775  5
775  5         HOME;
776  5         END;
777  4    (*$L+ *)
```

```
Line File Stmt Level
1165 1165  777   4          '\': (* COMPUTE SHOULDER *)
1166 1166  777   4             BEGIN
1167 1167  778   5             HOME; WRITELN('COMPUTING SHOULDER WITH 2 DOTS ');
1168 1168  780   5             SLOPE := (Y[1]-Y[2]) / (X[1]-X[2]);
1169 1169  781   5             YBAR := Y[1] - (SLOPE*X[1]); (* ORD A L'ORIGINE *)
1170 1170  782   5             (* -1/SLOPE IS SLOPE OF LINE PERPENDICULAR PASSING BY
1171 1171  782   5                ORIGIN *)
1172 1172  782   5             WRITE('YBAR,SLOPE=');
1173 1173  783   5             WRITELN(YBAR:8:5,SLOPE:8:5);
1174 1174  784   5
1175 1175  784   5             (* POINT OF INTERCEPTION *)
1176 1176  784   5             INTX := (-SLOPE*YBAR) / (1 + SLOPE*SLOPE);
1177 1177  785   5             INTY := -INTX / SLOPE;
1178 1178  786   5
1179 1179  786   5             WRITE('INTX,INTY=',INTX:8:5,INTY:8:5);
1180 1180  787   5             SHOULDER := SQR(INTX) + SQR(INTY);
1181 1181  788   5             WRITELN('  SHOULDER= ',SHOULDER:8:4);
1182 1182  789   5
1183 1183  789   5             S1 := SQRT( SQR(X[1]-X[2]) + SQR(Y[1]-Y[2]) );
1184 1184  790   5             NICM4 := ABS(MPULSE[4] / S1);
1185 1185  791   5             WRITE('DIST. COVERED=',S1:8:4,' cm     ');
1186 1186  792   5             WRITELN('NO OF IMPUL/cm  =',NICM4:8:4);
1187 1187  793   5
1188 1188  793   5             WRITE('TYPE CR'); READLN(C); HOME;
1189 1189  796   5             END;
1190 1190  797   4          '0': (* SET INITIAL POS: XSTART,YSTART,WASTART,XOTIP,YOTIP *)
1191 1191  797   4             BEGIN
1192 1192  798   5             FINDPOS(XSTART,YSTART,XOTIP,YOTIP);
1193 1193  799   5             WASTART := WRISTANG(XSTART,YSTART,XOTIP,YOTIP);
1194 1194  800   5             INITMOT;
1195 1195  801   5             WRITELN('CENTER OF ROTATION(MOTOR 3) BLINKING ');
1196 1196  802   5             BLINK2(XSTART,YSTART);
1197 1197  803   5             WRITELN('TIP BLINKING');
1198 1198  804   5             BLINK2(XOTIP,YOTIP);
1199 1199  805   5             WRITE('WRISTANGLE= ',WASTART:6:2,'    TYPE CR '); READLN(C);
1200 1200  807   5             HOME;
1201 1201  808   5             GETSHOW;
1202 1202  809   5             END;
1203 1203  810   4          '*': (* DETERMINE CURRENT CUR POS AND MOVE TO INITIAL POS *)
1204 1204  810   4             BEGIN
1205 1205  811   5             FINDPOS(XNOW,YNOW,XTIPNOW,YTIPNOW);
1206 1206  812   5             WANOW := WRISTANG(XNOW,YNOW,XTIPNOW,YTIPNOW);
1207 1207  813   5             GIVEPUL(XNOW,YNOW,WANOW, XSTART,YSTART,WASTART,
1208 1208  814   5                           NOPUL1,NOPUL3,NOPUL4);
1209 1209  814   5             MOVEARM(1,NOPUL1); MOVEARM(3,NOPUL3); MOVEARM(4,NOPUL4);
1210 1210  817   5             INITMOT;
1211 1211  818   5             HOME;
1212 1212  819   5             END;
1213 1213  820   4    (*L+ *)
```

```
Line File Stmt Level
1214 1214  820    4          'M': (* SET LIMITS (FROM INITIAL POS) FOR MOVES *)
1215 1215  820    4             BEGIN
1216 1216  821    5             HOME;
1217 1217  822    5             WRITE('MINPUL[1..6]=   ');
1218 1218  823    5             FOR I := 1 TO 6 DO
1219 1219  824    6                WRITE(MINPUL[I]:5);
1220 1220  825    5             WRITELN;
1221 1221  826    5             WRITE('MAXPUL[1..6]=   ');
1222 1222  827    5             FOR I := 1 TO 6 DO
1223 1223  828    6                WRITE(MAXPUL[I]:5);
1224 1224  829    5             WRITELN; WRITELN;
1225 1225  831    5             WRITE('IS IT FOR MIN (M) OR MAX (X)? '); READLN(C);
1226 1226  833    5             WRITE('FOR WHICH MOTOR? 1..6 '); READLN(I);
1227 1227  835    5             IF C = 'M' THEN MINPUL[I] := MPULSE[I]
1228 1228  837    5             ELSE MAXPUL[I] := MPULSE[I];
1229 1229  838    5             HOME;
1230 1230  839    5             END;
1231 1231  840    4          'L': (* LENGTHS FOR CROSS SEARCH *)
1232 1232  840    4             BEGIN
1233 1233  841    5             HOME;
1234 1234  842    5             WRITE('FAR TOLFAR NEAR TOLNEAR? (cm)  ');
1235 1235  843    5             READLN(FAR, TOLFAR, NEAR, TOLNEAR);
1236 1236  844    5             HOME;
1237 1237  845    5             END;
1238 1238  846    4    (*$L+ *)
```

```
Line File Stmt Level
1239 1239  846   4           'V': (* DISPLAY PARAMETER VALUES *)
1240 1240  846   4               BEGIN
1241 1241  847   5               HOME;
1242 1242  848   5               WRITELN('CORX=',CORX:7:5,'  CORY=',CORY:7:5);
1243 1243  849   5               WRITELN('X0=',X0:7:2,'   Y0=',Y0:7:2,'   NIDM1=',NIDM1:6:2);
1244 1244  850   5               WRITELN('RAD3=',RAD3:7:2,'   NIDM3=',NIDM3:6:2);
1245 1245  851   5               WRITELN('SHOULDER=',SHOULDER:7:2,'   NICM4=',NICM4:8:2);
1246 1246  852   5               WRITE('TYPE CR FOR MORE '); READLN(C);
1247 1247  854   5
1248 1248  854   5               HOME;
1249 1249  855   5               WRITE('MOTHOLD[0..6]=');
1250 1250  856   5               FOR I := 0 TO 6 DO WRITE(MOTHOLD[I]:4,' '); WRITELN;
1251 1251  859   5               WRITE('XSTART=',XSTART:7:2,'   YSTART=',YSTART:7:2);
1252 1252  860   5               WRITELN('   WASTART=',WASTART:7:2);
1253 1253  861   5               WRITELN('XOTIP=',XOTIP:7:2,'   YOTIP=',YOTIP:7:2);
1254 1254  862   5               WRITE('TYPE CR FOR MORE '); READLN(C);
1255 1255  864   5
1256 1256  864   5               HOME;
1257 1257  865   5               WRITE('UPM2=',UPM2:5,'  PICKM2=',PICKM2:5);
1258 1258  866   5               WRITELN('  NORMM2=',NORMM2:5);
1259 1259  867   5               WRITELN('CLOSE5=',CLOSE5:5,'   GRIP3=',GRIP3:6:4);
1260 1260  868   5               WRITE('MINPUL[1..6]=');
1261 1261  869   5               FOR I:= 1 TO 6 DO WRITE(MINPUL[I]:6,' '); WRITELN;
1262 1262  872   5               WRITE('MAXPUL[1..6]=');
1263 1263  873   5               FOR I:= 1 TO 6 DO WRITE(MAXPUL[I]:6,' '); WRITELN;
1264 1264  876   5               WRITE('TYPE CR FOR MORE '); READLN(C);
1265 1265  878   5
1266 1266  878   5               HOME;
1267 1267  879   5               WRITE('FAR=',FAR:6:3,'   TOLFAR=',TOLFAR:6:3);
1268 1268  880   5               WRITELN('  NEAR=',NEAR:6:3,'   TOLNEAR=',TOLNEAR:6:3);
1269 1269  881   5               WRITELN('STANDBY=',STANDBY:6,'   DROPCROSS=',DROPCROSS:6);
1270 1270  882   5
1271 1271  882   5               WRITE('TYPE CR'); READLN(C); HOME;
1272 1272  885   5               END;
1273 1273  886   4           'U':
1274 1274  886   4               BEGIN
1275 1275  887   5               HOME; WRITELN('UP- / DOWN+ MOTOR 2');
1276 1276  889   5               C := 'N';
1277 1277  890   5               WHILE C <> 'Y' DO
1278 1278  891   6                   BEGIN
1279 1279  892   7                   WRITE('ENTER NO OF PULSES '); READLN(I);
1280 1280  894   7                   MOVEARM(2,I);
1281 1281  895   7                   HOME; WRITE('OK? Y/N '); READLN(C); HOME;
1282 1282  899   7                   END;
1283 1283  900   5               WRITE('UPM2, PICKM2, NORMM2? ');
1284 1284  901   5               READLN(UPM2,PICKM2,NORMM2);
1285 1285  902   5               HOME;
1286 1286  903   5               END;
1287 1287  904   4      (*$L+ *)
```

```
Line File Stmt Level
1288 1288  904    4          'G': (* SET PARAMETERS CLOSE5 AND GRIP3 *).
1289 1289  904    4             BEGIN
1290 1290  905    5             HOME; WRITELN('CLOSE+ / OPEN- MOTOR 5 ');
1291 1291  907    5             C := 'N';
1292 1292  908    5             WHILE C <> 'Y' DO
1293 1293  909    6                BEGIN
1294 1294  910    7                WRITE('ENTER NO OF PULSES '); READLN(I);
1295 1295  912    7                MOVEARM(5,I);
1296 1296  913    7                HOME; WRITE('OK? Y/N '); READLN(C); HOME;
1297 1297  917    7                END;
1298 1298  918    5             WRITE('CLOSE5, GRIP3? ');
1299 1299  919    5             READLN(CLOSE5,GRIP3);
1300 1300  920    5             HOME;
1301 1301  921    5             END;
1302 1302  922    4    (*$L+ *)
```

```
Stmt Level
 922   4            'X': (* WAIT FOR CROSS AND PICK IT *)
 922   4               BEGIN
 923   5               MOVEARM(1,-MPULSE[1]);
 924   5               MOVEARM(3,-MPULSE[3]);
 925   5               MOVEARM(4,-MPULSE[4]);
 926   5               FINDPOS(XSTART,YSTART,XOTIP,YOTIP);
 927   5               WASTART := WRISTANG(XSTART,YSTART,XOTIP,YOTIP);
 928   5               INITMOT;
 929   5
 929   5
 929   5
 929   5
 929   5               MOVEARM(2,UPM2);
 930   5               MOVEARM(1,STANDBY);
 931   5               MOVEARM(5,-CLOSE5);
 932   5               MOVEARM(0,0);
 933   5
 933   5
 933   5               NOCROSS := TRUE;
 934   5               WHILE NOCROSS DO
 935   6                  BEGIN
 936   7                  HOME; WRITE('STANDBY, WAITING FOR CROSS ');
 938   7                  WAITCROSS(U,V,X,Y,WANGLES);
 939   7                  HOME; WRITE('CROSS DETECTED');
 941   7
 941   7                  FOUND := FALSE; I := 0;
 943   7                  WHILE (NOT FOUND) AND (I < 4) DO
 944   8                     BEGIN
 945   9                     MINIDX := 0; MINDIST := MAXINT;
 947   9                     FOR J := 1 TO 4 DO
 948  10                        BEGIN
 949  11                        DIST := DISTANCE(0,0,X[J],Y[J]);
 950  11                        IF DIST < MINDIST THEN
 951  11                           BEGIN MINDIST := DIST; MINIDX := J; END;
 955  11                        END;
 956   9                     GIVEPUL(XSTART,YSTART,WASTART,X[MINIDX],Y[MINIDX],
 957   9                             WANGLES[MINIDX],NOPUL1,NOPUL3,NOPUL4);
 957   9                     IF (NOPUL1>=MINPUL[1]) AND (NOPUL1<=MAXPUL[1]) AND
 958   9                        (NOPUL3>=MINPUL[3]) AND (NOPUL3<=MAXPUL[3]) AND
 958   9                        (NOPUL4>=MINPUL[4]) AND (NOPUL4<=MAXPUL[4]) THEN
 958   9   (*$L+ *)
```

```
Line File Stmt Level
1344 1344  958   9          (* TST START *) BEGIN
1345 1345  959  10                            HOME; BLINK2(U[MINIDX],V[MINIDX]);
1346 1346  961  10                            WRITE('ACCESS CROSS BY THIS SIDE? Y/N ');
1347 1347  962  10                            READLN(C);
1348 1348  963  10                            IF C = 'Y' THEN FOUND := TRUE
1349 1349  965  10                            ELSE
1350 1350  965  10                              BEGIN
1351 1351  966  11                              X[MINIDX] := MAXINT;
1352 1352  967  11                              Y[MINIDX] := MAXINT;
1353 1353  968  11                              END;
1354 1354  969  10                            HOME;
1355 1355  970  10                            END      (* TST END **)
1356 1356  971   9             (* NON TEST  FOUND := TRUE  *)
1357 1357  971   9                        ELSE
1358 1358  971   9                            BEGIN
1359 1359  972  10                            X[MINIDX] := MAXINT;
1360 1360  973  10                            Y[MINIDX] := MAXINT;
1361 1361  974  10                            END;
1362 1362  975   9                        I := I + 1;
1363 1363  976   9                        END;
1364 1364  977   7                   IF (NOT FOUND) THEN
1365 1365  978   7                        BEGIN
1366 1366  979   8                        HOME;
1367 1367  980   8                        WRITELN('UNREACHABLE! ');
1368 1368  981   8                        WRITE('REPOSITION CROSS AND TYPE Y     OR N TO EXIT ');
1369 1369  982   8                        READLN(C);
1370 1370  983   8                        IF C = 'Y' THEN NOCROSS := TRUE
1371 1371  985   8                        ELSE
1372 1372  985   8                            BEGIN
1373 1373  986   9                            MOVEARM(5,CLOSE5);
1374 1374  987   9                            MOVEARM(2,PICKM2+NORMM2);
1375 1375  988   9                            NOCROSS := FALSE;
1376 1376  989   9                            END;
1377 1377  990   8                        HOME;
1378 1378  991   8                        END
1379 1379  992   7                   ELSE
1380 1380  992   7                        BEGIN
1381 1381  993   8                        HOME; WRITE('REACHABLE');
1382 1382  995   8                        NOCROSS := FALSE;
1383 1383  996   8
1384 1384  996   8                        MOVEARM(1,NOPUL1-STANDBY);
1385 1385  997   8                        MOVEARM(4,NOPUL4);
1386 1386  998   8                        MOVEARM(3,NOPUL3);
1387 1387  999   8                        MOVEARM(2,PICKM2);
1388 1388 1000   8
1389 1389 1000   8   (* TST *)            HOME; WRITE('TYPE RETURN TO CONTINU '); READLN(C);
1390 1390 1003   8                        HOME;
1391 1391 1004   8
1392 1392 1004   8   (*$L+ *)
```

```
Line File Stmt Level
1393 1393 1004   8                        MOVEARM(5,CLOSE5+15);
1394 1394 1005   8
1395 1395 1005   8                        MOVEARM(2,NORMM2+UPM2);
1396 1396 1006   8                        MOVEARM(1,-NOPUL1+DROPCROSS);
1397 1397 1007   8                        MOVEARM(3,-NOPUL3);
1398 1398 1008   8                        MOVEARM(4,-NOPUL4);
1399 1399 1009   8                        MOVEARM(5,-CLOSE5);
1400 1400 1010   8                        MOVEARM(5,CLOSE5+15);
1401 1401 1011   8                        MOVEARM(1,STANDBY-DROPCROSS);
1402 1402 1012   8                        MOVEARM(2,PICKM2+NORMM2);
1403 1403 1013   8                        HOME;
1404 1404 1014   8                        END;
1405 1405 1015   7                     END;  (* WHILE NOCROSS *)
1406 1406 1016   5
1407 1407 1016   5                  END  (* 'X' *)
1408 1408 1017   4
1409 1409 1017   4
1410 1410 1017   4              OTHERWISE  (* DO NOTHING *) ;
1411 1411 1018   4              END; (* CASE *)
1412 1412 1018   4          END; (* WHILE *)
1413 1413 1019   2
1414 1414 1019   2     HOME;
1415 1415 10[illegible]   2     WRITE('LEAVING ARMMODE, UPDATE PARM FILE? Y/N? '); READLN(C);
1416 1416 1022   2     IF C = 'Y' THEN STORPARM;
1417 1417 1024   2
1418 1418 1024   2     PARAM.CTRLOUT := 0;
1419 1419 1025   2     PARAM.HOLDTIME := MOTHOLD[0];
1420 1420 1026   2     ARMOUT(PARAM);
1421 1421 1027   2     HOME;
1422 1422 1028   2     END; (* OF ARMMOVE *)
1423 1423 1029   1     (*$L+ *)
```

```
Line File Stmt Level
1424 1424 1029   1   (* ************************************************************************** *)
1425 1425 1029   1   (* ************************************************************************** *)
1426 1426 1029   1   (* ------------------------- BEGIN MAIN ------------------------------ *)
1427 1427 1029   1   BEGIN
1428 1428 1029   1   FOR I:=0 TO 6 DO MOTHOLD[I] := 10;
1429 1429 1031   1   INITMOT; (* DISABLE ALL 6 MOTORS *)
1430 1430 1032   1   PI := 4.0 * ARCTAN(1.0);
1431 1431 1033   1   RADDEG := 180/PI;
1432 1432 1034   1   CORX := 1.0;  CORY := 1.0; X0 := 0.0; Y0 := 0.0;
1433 1433 1038   1   VIDMODE := 6;    (* 640X200 PIXELS, BLACK AND WHITE *)
1434 1434 1039   1   GMODE(VIDMODE);
1435 1435 1040   1   EXPTIME := 1100;
1436 1436 1041   1
1437 1437 1041   1
1438 1438 1041   1   PARAM2.FIRST_BYTE := 0;
1439 1439 1042   1   PARAM2.SCREEN_START := 0;
1440 1440 1043   1   PARAM2.SCR_ROWCT := 128;
1441 1441 1044   1   PARAM2.SCR_COLCT := 80;
1442 1442 1045   1
1443 1443 1045   1   C := 'G';
1444 1444 1046   1   WHILE C <> 'Z' DO
1445 1445 1047   2       BEGIN
1446 1446 1048   3       PARAM1.KEY := 0;
1447 1447 1049   3       WHILE PARAM1.KEY = 0 DO GETSHOW;
1448 1448 1051   3       C := CHR(PARAM1.KEY MOD 256);
1449 1449 1052   3       CASE C OF
1450 1450 1053   3       'A': ARMMOVE;
1451 1451 1054   3   (*$L+ *)
```

```
Stmt Level
1054  3        'L': (* LOAD A PICTURE FROM FILE, DISPLAY AND OPTIONNALY PRINT IT *)
1054  3           BEGIN
1055  4           HOME;
1056  4           WRITELN('NO. OF BYTES= ',PARAM1.BYTECNT);
1057  4           WRITELN('KEY PRESSED = ',C);
1058  4           WRITELN('LOAD WHAT FILE?  ');  READLN(FILENAME);
1060  4           HOME;
1061  4           RESET(FILENAME,PICFILE);
1062  4           FOR I := 1 TO 128 DO
1063  5              FOR J := 1 TO 80 DO
1064  6                 BEGIN
1065  7                 READ(PICFILE,PIC8);
1066  7                 FR2[I,J] := PIC8;
1067  7                 END;
1068  4           CADRE;
1069  4           MOVESCR(PARAM2,FR2);
1070  4           WRITE('FILE LOADED, WANT TO PRINT PICTURE? Y/N  ');
1071  4           READLN(C);
1072  4           IF C = 'Y' THEN
1073  4              BEGIN
1074  5              HOME;
1075  5              WRITELN('CONNECT PRINTER TO // PORT.');
1076  5              WRITELN('THE GRAPHICS COMMAND MUST HAVE BEEN EXECUTED.');
1077  5              WRITELN;
1078  5              WRITE('TYPE Y TO PROCEED, N TO CANCEL '); READLN(C);
1080  5              HOME;
1081  5              IF C = 'Y' THEN
1082  5                 PRINTSCR;
1083  5              END;
1084  4           HOME;
1085  4           WRITE('TYPE ANY KEY TO CONTINUE '); READ(C);
1087  4           HOME;
1088  4           END;
1089  3        'S': (* SAVE CURRENT PICTURE TO FILE *)
1089  3           BEGIN
1090  4           HOME;
1091  4           WRITELN('NO. OF BYTES= ',PARAM1.BYTECNT);
1092  4           WRITELN('KEY PRESSED = ',C);
1093  4           WRITELN('SAVE WHAT FILE?  ');  READLN(FILENAME);
1095  4           REWRITE(FILENAME,PICFILE);
1096  4           FOR I := 1 TO 128 DO
1097  5              FOR J := 1 TO 80 DO
1098  6                 BEGIN
1099  7                 PIC8 := FR2[I,J];
1100  7                 WRITE(PICFILE,PIC8);
1101  7                 END;
1102  4           RESET( 'CON:',PICFILE);
1103  4           WRITELN('FILE SAVED, TYPE ANY KEY TO CONTINUE '); READ(C);
1105  4           HOME;
1106  4           END;
1107  3     (*$L+ *)
```

```
Line File Stmt Level
1503 1503 1107    3        'P':
1504 1504 1107    3            BEGIN
1505 1505 1108    4            HOME;
1506 1506 1109    4            WRITELN('NO. OF BYTES= ',PARAM1.BYTECNT);
1507 1507 1110    4            WRITELN('KEY PRESSED = ',C);
1508 1508 1111    4            WRITE('CURSOR POS? ROW=0..127, COL=0..639 ');
1509 1509 1112    4            READLN(ROW,COL);
1510 1510 1113    4            COORD.ROW := ROW; COORD.COL := COL;
1511 1511 1115    4            BLINK(COORD);
1512 1512 1116    4            HOME;
1513 1513 1117    4            END;
1514 1514 1118    3        'T':
1515 1515 1118    3            BEGIN
1516 1516 1119    4            HOME;
1517 1517 1120    4            WRITELN('NO. OF BYTES= ',PARAM1.BYTECNT);
1518 1518 1121    4            WRITELN('KEY PRESSED = ',C);
1519 1519 1122    4            WRITELN('CURRENT EXP TIME IN MSEC=',EXPTIME);
1520 1520 1123    4            WRITE('ENTER EXP TIME IN MSEC=');
1521 1521 1124    4            READLN(EXPTIME);
1522 1522 1125    4            HOME;
1523 1523 1126    4            END
1524 1524 1127    3
1525 1525 1127    3        ELSE :
1526 1526 1127    3            BEGIN
1527 1527 1128    4            HOME;
1528 1528 1129    4            WRITELN('NO. OF BYTES= ',PARAM1.BYTECNT);
1529 1529 1130    4            WRITELN('KEY PRESSED = ',C);
1530 1530 1131    4            WRITELN('SUM OF ALL INPUT BYTES= ', PARAM1.SUMB);
1531 1531 1132    4            WRITE('PRESS ANY KEY TO CONTINUE '); READ(C);
1532 1532 1134    4            HOME;
1533 1533 1135    4            END;
1534 1534 1136    3            END; (* CASE *)
1535 1535 1136    3        END; (* WHILE *)
1536 1536 1137    1    GMODE(VIDMODE);
1537 1537 1138    1    END.
0 compilation error(s).
```

```
                PAGE
COMMENT !
SET OF ASSEMBLER SUBROUTINES CALLABLE FROM PASCAL PROGRAM CAMPAS.
AUTHORS: BERNARD BROCHU
         and The Micr?int Inc., Cedarhurst, NY.

Port 0319H controls the picture format:  (- means active low; + active high)
    bit7:   -send/+nosend
    bit6:   -refresh/+soak
    bit5:   -onearray/+2array
    bit4:   -7bits/+8bits
    bit3:   -videpix/+novidepix
    bit2:   -altbit/+noaltbit
    bit1:   1  (don't care)
    bit0:   1  (don't care)
note: bit 7 here correspond to bit 7 of the IBM data bus but correspond to
      bit 0 in the ACIA registers. Bit 0 here corrsepond to bit 7 of ACIA
      registers. same rule for all other bits (6,5,4,3,2,1).
!
```

```
                                PAGE
                                ;***************************************************************************
                                COMMENT !
                                        SCREEN_START -- The byte position on the screen page at which the
                                                picture should start.  This position is calculated as:
                                                        (ROW*80)/2 + (COLUMN/8).
                                                Row must be an even number between 0 and 134. Column must
                                                be between 0 and 512 and divisible by 8.

                                        KEY_VALUE -- At the beginning of each FRAMEGRAB this variable is set
                                                to zero.  If during the FRAMEGRAB a key is pressed, then the
                                                ASCII value of the key is placed in the LSB of key_value.
                                        !

0000                            DATA    SEGMENT  PUBLIC
0000  ????                      MAPADR          DW      ?       ; USED BY FRGRAB TO SAVE ADDR. OF DEST ARRAY
0002  0000                      BITMAP          DW      0
0004  0000                      WORKMAP         DW      0
0006  0000                      BYTE_CNT        DW      0
0008  0000                      KEY_VALUE       DW      0
000A  0000                      SCREEN_START    DW      0
000C  0080                      SCR_ROWCT       DW      128
000E  0040                      SCR_COLCT       DW      64
0010  0318                      CONTROL         DW      318H            ; ACIA control
0012  0318                      STATUS          DW      318H            ; ACIA status
0014  0319                      DATAIN          DW      319H            ; received pixels from camera
0016  0319                      DATAOUT         DW      319H    ; to control camera formats
0018  012C                      EXPOSE_TIME     DW      300     ; to store value passed by calling PGM
001A  1F                        COMMAND         DB      1FH
001B  ????                      ROWCTR          DW      ?
001D  ????                      COLCTR          DW      ?
001F  ????                      SUMB            DW      ?
0021  28                        DIVISOR         DB      40
0022  08                        DIV8            DB      8
0023  02                        DIV2            DB      2
0024  50                        MULT80          DB      80
0025  ????                      LMOST           DW      ?
0027  ????                      RMOST           DW      ?
0029  ????                      BMOST           DW      ?
002B  ????                      TMOST           DW      ?
002D  ????                      LMOSTOLD        DW      ?
002F  ????                      RMOSTOLD        DW      ?
0031  ????                      BMOSTOLD        DW      ?
0033  ????                      TMOSTOLD        DW      ?
0035                            DATA    ENDS
```

```
                                PAGE
                                PUBLIC CLRCOMM,CALLDOS,CALLBIOS,GMODE,MOVESCR,FRGRAB,ENHANCE2
                                PUBLIC DETDOTS,ARMOUT,BLINK
0000                    CODE    SEGMENT BYTE PUBLIC 'PASCAL'
                                ASSUME CS:CODE,DS:DATA

0000                    CALLDOS PROC NEAR
0000  5F                        POP DI
0001  5E                        POP SI
0002  8B 04                     MOV AX,[SI]
0004  8B 5C 02                  MOV BX,[SI+2]
0007  8B 4C 04                  MOV CX,[SI+4]
000A  8B 54 06                  MOV DX,[SI+6]

000D  57                        PUSH DI
000E  56                        PUSH SI

000F  55                        PUSH BP
0010  06                        PUSH ES
0011  1E                        PUSH DS
0012  CD 21                     INT 21H
0014  1F                        POP DS
0015  07                        POP ES
0016  5D                        POP BP

0017  5E                        POP SI
0018  89 54 06                  MOV [SI+6],DX
001B  89 4C 04                  MOV [SI+4],CX
001E  89 5C 02                  MOV [SI+2],BX
0021  89 04                     MOV [SI],AX

0023  C3                        RET
0024                    CALLDOS ENDP
```

```
                                    PAGE
0024                 CALLBIOS PROC NEAR
0024  5F                      POP DI          ; RETURN ADDR.
0025  5E                      POP SI          ; ADDR. OF RECORD
0026  57                      PUSH DI         ; INSTALL RETURN ADDR.

0027  8B 44 0E                MOV AX,[SI+14]     ; INT NUMBER IN AL    00..1F
002A  2E: A2 0044 R           MOV CS:INTNO,AL
002E  2E: C6 06 0043 R CD     MOV CS:SWINT,0CDH  ; OPCODE FOR INT n  n<>3
      90

0035  8B 04                   MOV AX,[SI]
0037  8B 5C 02                MOV BX,[SI+2]
003A  8B 4C 04                MOV CX,[SI+4]
003D  8B 54 06                MOV DX,[SI+6]

0040  56                      PUSH SI
0041  55                      PUSH BP
0042  1E                      PUSH DS
0043  ??             SWINT    DB  ?           ; OPCODE FOR INT INSTRUCTION
0044  ??             INTNO    DB  ?           ; HOLD INT NUMBER <> 3
0045  1F                      POP DS
0046  5D                      POP BP
0047  5E                      POP SI

0048  89 54 06                MOV [SI+6],DX
004B  89 4C 04                MOV [SI+4],CX
004E  89 5C 02                MOV [SI+2],BX
0051  89 04                   MOV [SI],AX
0053  C3                      RET
0054                 CALLBIOS ENDP
```

```
                                        PAGE
0054                    GMODE   PROC    NEAR
0054  5F                        POP DI
0055  5E                        POP SI
0056  5E                        POP SI
0057  57                        PUSH DI

0058  56                        PUSH SI
0059  57                        PUSH DI
005A  55                        PUSH BP
005B  B4 0F                     MOV     AH,0FH
005D  CD 10                     INT     10H
005F  5D                        POP BP
0060  5F                        POP  DI
0061  5E                        POP  SI
                                ; AL HOLDS CURRENT VIDEO MODE
0062  8B D0                     MOV DX,AX      ;SAVE CURRENT MODE (AL)
0064  8B 04                     MOV AX,[SI]    ;GET DESIRED MODE
0066  8B D8                     MOV BX,AX      ;SAVE DESIRED MODE
0068  8B C2                     MOV AX,DX      ;AX <- CURRENT MODE
006A  B4 00                     MOV AH,0       ;MAKE SURE THAT ONLY AL HOLDS A VALUE
006C  89 04                     MOV [SI],AX    ;RETURN CURR MODE VALUE
006E  8B C3                     MOV AX,BX      ;DESIRED MODE INTO AL

0070  56                        PUSH SI
0071  57                        PUSH DI
0072  55                        PUSH BP
0073  B4 00                     MOV     AH,0
0075  CD 10                     INT     10H
0077  5D                        POP BP
0078  5F                        POP  DI
0079  5E                        POP  SI


007A  C3                        RET
007B                    GMODE   ENDP
```

```
                                          PAGE
007B                        CLRCOMM PROC NEAR
007B  55                            PUSH BP
007C  06                            PUSH ES

007D  B8 B800                       MOV AX,0B800H   ; ES AT
0080  8E C0                         MOV ES,AX       ;        BEGINNING OF VIDEO RAM
0082  B3 24                         MOV BL,36       ; 72 ROWS BUT WE CLEAR EVEN AND ODD IN ONE LOOP EXECUT:
0084  BF 1400                       MOV DI,5120     ; DISPLACEMENT IN VIDEO RAM OF LINE 16 (OUT OF 0..24)
0087  B8 0000                       MOV AX,0
008A  B9 0028               AGAIN:  MOV CX,40       ; # OF WORDS TO CLEAR FOR ONE ROW
008D  FC                            CLD
008E  F3/ AB                        REP STOSW ;UNTIL CX=0
0090  83 EF 50                      SUB DI,80       ; TO START OF EVEN ROW
0093  81 C7 2000                    ADD DI,2000H    ; TO START OF CORRESPONDING ODD ROW
0097  B9 0028                       MOV CX,40
009A  F3/ AB                        REP STOSW ;UNTIL CX=0
009C  81 EF 2000                    SUB DI,2000H    ; TO START OF NEXT EVEN ROW
00A0  FE CB                         DEC BL
00A2  75 E6                         JNZ AGAIN

00A4  07                            POP ES
00A5  5D                            POP BP
00A6  C3                            RET
00A7                        CLRCOMM ENDP
```

```
                                PAGE
                                ;Move picture to graphics screen
00A7                 MOVESCR PROC    NEAR
00A7  5F                     POP     DI              ; DI <- RETURN ADDRESS
00A8  5E                     POP     SI              ; SI <- ADDRESS OF M130X80
00A9  58                     POP AX                  ; AX <- ADDR OF MISC REC
00AA  57                     PUSH    DI              ; INSTALL RETURN ADDRESS
00AB  06                     PUSH    ES
00AC  8B F8                  MOV DI,AX
00AE  8B 05                  MOV AX,[DI]
00B0  03 F0                  ADD     SI,AX           ; SI <- ADDR. OF 1ST BYTE IN MATRIX
                                                     ;   I.E. 1ST ROW TO BE DISPLAYED
00B2  8B 45 02               MOV AX,[DI+2]
00B5  A3 000A R              MOV SCREEN_START,AX
00B8  8B 45 04               MOV AX,[DI+4]
00BB  A3 000C R              MOV SCR_ROWCT,AX
00BE  8B 45 06               MOV AX,[DI+6]
00C1  A3 000E R              MOV SCR_COLCT,AX


00C4  FC                     CLD                     ;make sure direction flag is forward
00C5  8B 3E 000A R           MOV     DI,SCREEN_START ;point destination register
00C9  B8 B800                MOV     AX,0B800H       ;  at desired offset on the
00CC  8E C0                  MOV     ES,AX           ;  graphics screen
00CE  8B 16 000C R           MOV     DX,SCR_ROWCT    ;DX is set to the number of rows/2
00D2  D1 EA                  SHR     DX,1            ;  because we handle two rows at a time
00D4  8B 1E 000E R           MOV     BX,SCR_COLCT    ;BX is set to the number of bytes/row
00D8  8B CB          nxtrow: MOV     CX,BX           ;CX is the number of words per row
00DA  D1 E9                  SHR     CX,1
00DC  F3/ A5                 REP     MOVSW           ;move the entire even row
00DE  2B FB                  SUB     DI,BX           ;set up for odd row which is 2000H away
00E0  81 C7 2000             ADD     DI,2000H
00E4  8B CB                  MOV     CX,BX
00E6  D1 E9                  SHR     CX,1
00E8  F3/ A5                 REP     MOVSW           ;move the entire odd row
00EA  81 EF 1FB0             SUB     DI,1FB0H        ;set up for next even row
00EE  2B FB                  SUB     DI,BX
00F0  4A                     DEC     DX              ;continue until we've done every row
00F1  75 E5                  JNZ     NXTROW
00F3  07                     POP     ES
00F4  C3                     RET
00F5                 MOVESCR ENDP
```

```
                                        PAGE
                                                  gets image from Micro Dcam
                          ;
00F5                      FRGRAB PROC  NEAR
00F5  58                         POP AX             ; AX <- RETURN ADDRESS
00F6  5F                         POP DI             ; DI <- ADDR. OF M128X80 PASSED BY CALLING PGM
00F7  5E                         POP SI             ; ADDR OF MISC RECORD
00F8  50                         PUSH AX            ; INSTALL RETURN ADDRESS
00F9  06                         PUSH ES
00FA  56                         PUSH SI


00FB  89 3E 0000 R               MOV MAPADR,DI
00FF  8B 04                      MOV AX,[SI]        ; ADDR FOR EXPOSE TIME
0101  A3 0018 R                  MOV EXPOSE_TIME,AX
0104  C7 06 0006 R 0000          MOV BYTE_CNT,0
010A  C7 06 001F R 0000          MOV SUMB,0

0110  8B 44 04                   MOV AX,[SI+4]
0113  A3 0010 R                  MOV CONTROL,AX
0116  A3 0012 R                  MOV STATUS,AX
0119  40                         INC AX
011A  A3 0014 R                  MOV DATAIN,AX
011D  A3 0016 R                  MOV DATAOUT,AX

0120  E8 0208 R                  CALL ACIACLR       ; INIT ACIA

0123  8A 26 001A R               MOV     AH,COMMAND     ;tell Micro Dcam to soak w/o send
0127  C7 06 0008 R 0000          MOV     KEY_VALUE,[illegible] ;zero keyvalue
012D  80 CC C0                   OR      AH,0C0H        ;NOSEND AND SOAK
0130  E8 0213 R                  CALL    SENDCMD
0133  E8 0224 R                  CALL    SOAK           ;soak for specified expose time
0136  E8 01A9 R                  CALL    KEYCHK         ;check for key before disabling interrupts
0139  8B DF                      MOV     BX,DI          ;save start address of buffer for compare
013B  8A 26 001A R               MOV     AH,COMMAND     ;tell Micro Dcam to send picture (w/o soak)
013F  FA                         CLI                    ;disable interrupts during grab
0140  E8 0213 R                  CALL    SENDCMD
0143  8B 16 0012 R               MOV     DX,STATUS
0147  8C D8                      MOV     AX,DS          ;equate extra segment and data segment
0149  8E C0                      MOV     ES,AX
014B  FC                         CLD                    ;set direction reg to forward movement
014C  B9 000F             nfby   MOV     CX,15          ;set up timeout register for char receipt
014F  EC                  RECHK: IN      AL,DX          ;if character not available after 15
0150  D0 E0                      SHL     AL,1           ;  checks then we assume the Micro Dcam is
0152  73 10                      JNC     DNCHK          ;  done sending
0154  42                         INC     DX             ;when character has come we point DX to
0155  EC                         IN      AL,DX          ;  DATAIN and get the character and then
```

```
0156  B4 00                     MOV    AH,0
0158  01 06 001F R              ADD    SUMB,AX
015C  4A                        DEC    DX           ;  repoint DX at the status register
015D  AA                        STOSB               ;put the byte in the buffer
015E  FF 06 0006 R              INC    BYTE_CNT
0162  EB E8                     JMP    NFBYT        ;go back and try and get another byte
0164  E2 E9             dnchk:  LOOP   RECHK        ;this is the timeout decrementer
0166  8B C7                     MOV    AX,DI        ;if we have timed out then we check and
0168  2B C3                     SUB    AX,BX        ;  see if we got as many bytes as we had
016A  3D 0FFF                   CMP    AX,4095      ;  hoped to get
016D  7F 03                     JG     NTO          ;if not enough bytes received then we
016F  E8 0234 R                 CALL   BEEP         ;  beep to show our disgust
0172  FB                nto:    STI                 ;reenable interrupts
0173  8A 26 001A R              MOV    AH,COMMAND   ;tell Micro Dcam to refresh w/o send
0177  80 CC 80                  OR     AH,80H
017A  E8 0213 R                 CALL   SENDCMD
017D  E8 01BE R                 CALL   KEYCLR       ;clear keyboard buffer
0180  80 3E 001A R 3F           CMP COMMAND,3FH
0185  75 0E                     JNE NTA
0187  8B 3E 0000 R              MOV DI,MAPADR
018B  8B F7                     MOV SI,DI
018D  83 C6 20                  ADD SI,32
0190  B9 0800                   MOV CX,2048
0193  F3/ A5                    REP MOVSW

0195  5F                NTA:    POP DI
0196  A1 0008 R                 MOV AX,KEY_VALUE
0199  89 05                     MOV [DI],AX
019B  A1 0006 R                 MOV AX,BYTE_CNT
019E  89 45 02                  MOV [DI+2],AX
01A1  A1 001F R                 MOV AX,SUMB
01A4  89 45 06                  MOV [DI+6],AX

01A7  07                        POP ES
01A8  C3                        RET


01A9  50                keychk: PUSH AX
01AA  B4 01                     MOV    AH,1         ;if key is available from keyboard buffer then
01AC  CD 16                     INT    16H          ;  get the key and scancode, put in key_value,
01AE  74 0C                     JZ     NOKEY        ;  and set ZF=0
01B0  B4 00                     MOV    AH,0         ;otherwise
01B2  CD 16                     INT    16H          ;  just set ZF=1
01B4  A3 0008 R                 MOV    KEY_VALUE,AX
01B7  A1 0008 R                 MOV    AX,KEY_VALUE
01BA  0B C0                     OR     AX,AX
01BC  58                nokey:  POP AX
01BD  C3                        RET
```

```
01BE  E8 01A9 R        keyclr: CALL   KEYCHK     ;clears the system keyboard buffer because we
01C1  75 FB                    JNZ    KEYCLR     ; can't trust the IBM to get the keyboard
01C3  C3                       RET               ; decoded properly when interrupts were off so
                               ;                 ; long.
01C4                   FRGRAB ENDP
```

```
                                        PAGE
01C4                    ARMOUT     PROC NEAR
01C4  5F                           POP DI            ; DI <- RETURN ADDRESS
01C5  5E                           POP SI            ; SI <- ADDR. OF RECORD
01C6  57                           PUSH DI           ; INSTALL RETURN ADDR.
01C7  56                           PUSH SI           ; SAVE

01C8  8B 44 02                     MOV AX,[SI+2]     ; AL <- BYTE TO OUTPUT, AH <- 0
01CB  BA 0378                      MOV DX,0378H      ; // PORT TO CONTROL ROBOT ARM

01CE  EE                           OUT DX,AL

01CF  C7 06 0008 R 0000            MOV KEY_VALUE,0

01D5  8B 4C 04                     MOV  CX,[SI+4]     ; NUMBER OF MILLISECONDS
01D8  E8 01E8 R                    CALL DELAY
01DB  E8 01A9 R                    CALL KEYCHK
01DE  E8 01BE R                    CALL KEYCLR

01E1  5F                           POP DI            ; DI <- ADDR. OF RECORD
01E2  A1 0008 R                    MOV AX,KEY_VALUE
01E5  89 05                        MOV [DI],AX
01E7  C3                           RET
01E8                    ARMOUT     ENDP

                        ;******************************************************************************
01E8                    DELAY      PROC NEAR
01E8  51                           PUSH CX
01E9  E3 09                        JCXZ OUTDELAY
01EB  51                DEL1:      PUSH CX
01EC  B9 0106                      MOV CX,262
01EF  E2 FE             DEL2:      LOOP DEL2
01F1  59                           POP CX
01F2  E2 F7                        LOOP DEL1
01F4  59                OUTDELAY:  POP CX
01F5  C3                           RET
```

```
                                        PAGE
                                ; CLEAR THE DESTINATION ARRAY FOR ENHANCE2
01F6                            CLEARW  PROC NEAR
01F6  57                                PUSH DI
01F7  FC                                CLD
01F8  B9 28A0                           MOV  CX,10400       ; (128+2)X80= # OF BYTES TO CLEAR (128X640 PIX)
                                                            ; 2 EXTRA ROWS BECAUSE ENHANCE 2 WRITES THERE.
01FB  D1 E9                             SHR  CX,1           ; /2  TO OBTAIN # OF WORDS
01FD  8C D8                             MOV  AX,DS
01FF  8E C0                             MOV ES,AX
0201  B8 0000                           MOV AX,0
0204  F3/ AB                            REP STOSW
0206  5F                                POP DI

0207  C3                                RET
0208                            CLEARW  ENDP

0208                            ACIACLR PROC    NEAR
0208  8B 16 0010 R                      MOV     DX,CONTROL
020C  B0 C0                             MOV     AL,0C0H             ;send master reset to camera
020E  EE                                OUT     DX,AL
020F  B0 28                             MOV     AL,28H              ;send camera protocol of
0211  EE                                OUT     DX,AL               ;  1 start, 8 data, 1 stop bits
0212  C3                                RET
0213                            ACIACLR ENDP
                                ;
0213                            SENDCMD PROC    NEAR
0213  8B 16 0012 R                      MOV     DX,STATUS
0217  EC                                IN      AL,DX               ;get status of camera
0218  A8 40                             TEST    AL,40H              ;see if command can be sent
021A  74 F7                             JZ      SENDCMD             ;loop until ready
021C  8A C4                             MOV     AL,AH               ;set up command
021E  8B 16 0016 R                      MOV     DX,DATAOUT
0222  EE                                OUT     DX,AL               ;send camera command
0223  C3                                RET
0224                            SENDCMD ENDP
                                ;
0224                            SOAK    PROC    NEAR
0224  8B 0E 0018 R                      MOV     CX,EXPOSE_TIME  ;soaktime = number msec delay
0228  E3 09                             JCXZ    NOSOAK
022A  51                        S1:     PUSH    CX
022B  B9 0106                           MOV     CX,262          ;set up loop for 1 msec
022E  E2 FE                     S2:     LOOP    S2
0230  59                                POP     CX
0231  E2 F7                             LOOP    S1
0233  C3                        NOSOAK: RET
0234                            SOAK    ENDP
                                ;
```

```
0234                BEEP   PROC   NEAR
0234  B4 02                MOV    AH,2          ; DOS call to sound bell
0236  B2 07                MOV    DL,7
0238  CD 21                INT    21H
023A  C3                   RET
023B                BEEP   ENDP
```

```
                                PAGE
                        ;       ENHANCE ROUTINE for 640 x 128 picture
                        ; INPUT: ADDR OF A 128X256 PIXELS FRAME (SOURCE); ADDR. OF A 128X640 PIXELS
                        ;     FRAME (DEST).
023B                    ENHANCE2 PROC   NEAR
023B  58                        POP AX                  ; RETURN ADDRESS
023C  5F                        POP DI                  ; ADDR OF DESTINATION ARRAY
023D  5E                        POP SI                  ; ADDR OF SOURCE ARRAY
023E  50                        PUSH AX                 ; INSTALL RETURN ADDR
023F  55                        PUSH BP                 ; ONLY BP,ES HAS TO BE SAVED FOR PASCAL PGM
0240  57                        PUSH DI                 ; SAVE FOR LATER USE HERE IN ENHANCE2

0241  83 C6 06                  ADD SI,6                ; TO CENTER THINGS: COL 0..47 AND 208..255
                                                        ; OF SOURCE ARE NOT PROCESSED. WE KEEP ONLY
                                                        ; THE CENTER 160 COL WHICH AFTER ENHANCE2
                                                        ; WILL RESULT IN 640 COL IN THE DEST PIC.


0244  FC                        CLD
0245  E8 01F6 R                 CALL    CLEARW          ; CLEAR DESTINATION AREA
0248  BB 0080                   MOV     BX,128          ; # OF ROWS
024B  D1 EB                     SHR     BX,1            ; # OF ROWS/2 BECAUSE PROCESS 2 ROWS AT A TIME
024D  B2 0A                     MOV     DL,10           ; SOURCE HAS 16 WORDS/ROW PROCESS ONLY 10
                                                        ;    TO GET A RESULT OF 640 PIXELS WIDE
                                ;EVEN ROW PROCESSING
024F  8A F2             nrow2:  MOV     DH,DL
0251  AD                nbe2:   LODSW                   ; AX <- SOURCE WORD
0252  86 C4                     XCHG    AL,AH           ; LEFTMOST PIXELS -> AH;  RIGHTMOST -> AL
0254  B9 0008                   MOV     CX,8            ; PROCESS EACH SOURCE WORD 2 BITS AT A TIME
                                                        ;    => 8 PROCESSINGS/WORD => 8 BYTES CREATED
                                                        ;    FOR EACH WORD OF THE SOURCE.
0257  D1 E0             nxt2:   SHL     AX,1
0259  73 04                     JNC     NOCA
025B  80 4D 50 03               OR      BYTE PTR 80[DI],3
025F  D1 E0             noca:   SHL     AX,1
0261  73 03                     JNC     NOCB
0263  80 0D 0C                  OR      BYTE PTR [DI],0CH
0266  47                nocb:   INC     DI
0267  E2 EE                     LOOP    NXT2
0269  FE CE                     DEC     DH
026B  75 E4                     JNZ     NBE2

                                ;ODD ROW PROCESSING
026D  8A F2                     MOV     DH,DL
026F  83 C6 0C                  ADD     SI,12           ;skip to start of next row
0272  AD                nbo2:   LODSW
0273  86 C4                     XCHG    AL,AH
0275  B9 0008                   MOV     CX,8
0278  D1 E0             nxt3:   SHL     AX,1
```

```
027A  73 03                    JNC     NOCC
027C  80 0D C0                 OR      BYTE PTR [DI],0C0H
027F  D1 E0           nocc:    SHL     AX,1
0281  73 04                    JNC     NOCD
0283  80 4D 51 30              OR      BYTE PTR 81[DI],30H
0287  47              nocd:    INC     DI
0288  E2 EE                    LOOP    NXT3
028A  FE CE                    DEC     DH
028C  75 E4                    JNZ     NB02
028E  83 C6 0C                 ADD     SI,12
0291  4B                       DEC     BX
0292  75 BB                    JNZ     NROW2

                               ; NOW FILL THE HOLES IN THE DESTINATION ARRAY
0294  5E                       POP SI               ; GET THE ADDR OF DEST ARRAY (128X640)
0295  56                       PUSH SI
0296  BB 0080                  MOV     BX,128       ;as discussed in the topology section and
0299  D1 EB                    SHR     BX,1         ;  elsewhere, the 256 x 128 array needs to
029B  89 1E 00[illegible]      MOV     ROWCTR,BX    ;  be descrambled and filled in.  This takes
029F  BA 0050                  MOV     DX,80        ;  care of the fillin when expanding the
02A2  D1 EA                    SHR     DX,1         ;  256 x 128 to 1024 x 128 (clipped to
02A4  89 16 001D R             MOV     COLCTR,DX    ;  640 x 128 because of screen limitations.
02A8  8B 04           nebyt:   MOV     AX,[SI]      ;to do the fillin we look at the bytes one
02AA  8B D8                    MOV     BX,AX        ;  row back and one row forward.  Where there
02AC  8B 8C 00A0               MOV     CX,160[SI]   ;  are 'holes' they occur in groups of two.
02B0  0B C1                    OR      AX,CX        ;the rightmost hole of the two is the result of
02B2  25 1414                  AND     AX,1414H     ;  OR'ing the bits one row back and one row
02B5  23 D9                    AND     BX,CX        ;  ahead
02B7  81 E3 2828               AND     BX,2828H     ;the leftmost hole of the two is the result of
02BB  0B C3                    OR      AX,BX        ;  AND'ing the bits one row back and one row
02BD  09 44 50                 OR      80[SI],AX    ;  ahead
02C0  83 C6 02                 ADD     SI,2
02C3  4A                       DEC     DX
02C4  75 E2                    JNZ     NEBYT
02C6  8B 16 001D R             MOV     DX,COLCTR
02CA  8B 04           nobyt:   MOV     AX,[SI]      ;when we work with the odd rows we do exactly
02CC  8B D8                    MOV     BX,AX        ;  as we did on the even rows.  the only
02CE  8B 8C 00A0               MOV     CX,160[SI]   ;  difference is where the holes fall.
02D2  0B C1                    OR      AX,CX        ;by using a slightly different mask we can
02D4  25 4141                  AND     AX,4141H     ;  accomplish the task
02D7  23 D9                    AND     BX,CX
02D9  81 E3 8282               AND     BX,8282H
02DD  0B C3                    OR      AX,BX
02DF  09 44 50                 OR      80[SI],AX
02E2  83 C6 02                 ADD     SI,2
02E5  4A                       DEC     DX
02E6  75 E2                    JNZ     NOBYT
02E8  8B 16 001D R             MOV     DX,COLCTR
02EC  FF 0E 001B R             DEC     ROWCTR
```

```
02F0  75 B6                     JNZ     NEBYT


02F2  5E                        POP SI
02F3  06                        PUSH ES
02F4  8C D8                     MOV AX,DS
02F6  8E C0                     MOV ES,AX

                                ; WHITEN 1ST 2 ROWS OF DESTINATION
02F8  B8 FFFF                   MOV AX,0FFFFH
02FB  B9 0050                   MOV CX,80    ; 40 WORDS / ROW,  WE BLANK 2 ROWS
02FE  8B FE                     MOV DI,SI
0300  FC                        CLD
0301  F3/ AB                    REP STOSW
                                ; WHITEN 1ST 4 COLUMNS
0303  B9 0080                   MOV CX,128
0306  8B FE                     MOV DI,SI
0308  81 0D 00F0      NXTCLR1:  OR WORD PTR [DI],00F0H
030C  83 C7 50                  ADD DI,80
030F  E2 F7                     LOOP NXTCLR1
                                ; WHITEN LAST ROW
0311  83 EF 50                  SUB DI,80
0314  B9 0028                   MOV CX,40
0317  FC                        CLD
0318  F3/ AB                    REP STOSW
                                ; WHITEN LAST COLUMN
031A  4F                        DEC DI
031B  4F                        DEC DI
031C  B9 0080                   MOV CX,128
031F  81 0D 0100      NXTCLR2:  OR WORD PTR [DI],0100H
0323  83 EF 50                  SUB DI,80
0326  E2 F7                     LOOP NXTCLR2

0328  07                        POP ES


0329  5D                        POP BP
032A  C3                        RET
032B                  ENHANCE2 ENDP
```

```
                                        PAGE
0328                    DETDOTS PROC NEAR
0328  58                        POP AX          ; RETURN ADDR.
032C  5E                        POP SI          ; ADDR. OF ARRAY TO BE SEARCH FOR A DOT
032D  5F                        POP DI          ; ADDR. OF RECORD FOR RETURN COORDINATES
032E  50                        PUSH AX         ; INSTALL RETURN ADDR.
032F  06                        PUSH ES

0330  57                        PUSH DI         ; SAVE
0331  8C D8                     MOV AX,DS
0333  8E C0                     MOV ES,AX
                                ; BLANK 1ST 2 ROWS
0335  B8 0000                   MOV AX,0
0338  B9 0050                   MOV CX,80       ; 40 WORDS / ROW
033B  8B FE                     MOV DI,SI
033D  FC                        CLD
033E  F3/ AB                    REP STOSW
                                ; BLANK 1ST 4 COLUMNS
0340  B9 0080                   MOV CX,128
0343  8B FE                     MOV DI,SI
0345  81 25 FF0F        NXTAND1: AND WORD PTR [DI],0FF0FH
0349  83 C7 50                  ADD DI,80
034C  E2 F7                     LOOP NXTAND1
                                ; BLANK LAST ROW
034E  83 EF 50                  SUB DI,80
0351  B9 0028                   MOV CX,40
0354  FC                        CLD
0355  F3/ AB                    REP STOSW
                                ; BLANK LAST COLUMN
0357  4F                        DEC DI
0358  4F                        DEC DI
0359  B9 0080                   MOV CX,128
035C  81 25 FEFF        NXTAND2: AND WORD PTR [DI],0FEFFH
0360  83 EF 50                  SUB DI,80
0363  E2 F7                     LOOP NXTAND2
0365  5F                        POP DI
0366  E8 0378 R         NXTDOT: CALL DETDOT
0369  89 1D                     MOV [DI],BX
036B  89 4D 02                  MOV [DI+2],CX
036E  83 C7 04                  ADD DI,4
0371  83 FB FF                  CMP BX,-1
0374  75 F0                     JNZ NXTDOT

0376  07                        POP ES
0377  C3                        RET
0378                    DETDOTS ENDP
```

```
                              PAGE
                        ; DETDOT IS GIVEN THE ADDR OF A PICTURE 640 COL X 128 ROWS, IT RETURNS THE
                        ; COORDINATES OF THE FIRST NON-ZERO PIXEL. COL: 0..639; ROW: 0..127.
                        ; SEARCH START AT TOPMOST,LEFTMOST PIXEL (ROW 0, COL 0) AND GOES ROW MAJOR.
                        ; IT RETURNS -1 FOR BOTH COORDINATES IF NOT FOUND.
0378                    DETDOT PROC NEAR
0378  50                        PUSH AX
0379  52                        PUSH DX
037A  06                        PUSH ES
037B  56                        PUSH SI
037C  57                        PUSH DI

037D  8B FE                     MOV DI,SI
037F  8B DF                     MOV BX,DI
0381  B9 1400                   MOV CX,40*128
0384  83 3D 00          NXTCMP: CMP WORD PTR [DI],0
0387  75 0F                     JNE FOUNDD
0389  83 C7 02                  ADD DI,2
038C  49                        DEC CX
038D  75 F5                     JNZ NXTCMP
038F  BB FFFF                   MOV BX,-1       ; RETURN -1 IF NOT FOUND
0392  B9 FFFF                   MOV CX,-1
0395  E9 0499 R                 JMP FINDOT
0398  8B C7             FOUNDD: MOV AX,DI       ; AX <- ADDR OF NON ZERO WORD
039A  2B C3                     SUB AX,BX       ; FOUND - START = DISPLACEMENT
039C  D1 E8                     SHR AX,1        ; AX <- WORD # (0..128*40-1)
039E  F6 36 0021 R              DIV DIVISOR     ; AL <- ROW # 0..127; AH <- WORD # 0..39
03A2  8B D0                     MOV DX,AX       ; DX <-
03A4  B6 00                     MOV DH,0        ;       ROW # 0..127
03A6  B9 0000                   MOV CX,0        ; WILL HOLD BIT POSITION MS=#1  LS=#15
03A9  8A 3D                     MOV BH,[DI]     ; LEFTMOST 8 BITS OF PICTURE IN BH
03AB  8A 5D 01                  MOV BL,[DI+1]   ; RIGHTMOST 8 BITS OF PICTURE IN BL
03AE  D1 E3             SHLNXT: SHL BX,1        ; EXAMINE BIT
03B0  72 03                     JC BITOK        ; OK NON ZERO BIT FOUND

03B2  41                        INC CX
03B3  EB F9                     JMP SHLNXT
03B5  8A C4             BITOK:  MOV AL,AH       ; AX <-
03B7  B4 00                     MOV AH,0        ;       WORD # 0..39
03B9  D1 E0                     SHL AX,1        ; MULTIPLY BY 16 TO OBTAIN # OF BITS
03BB  D1 E0                     SHL AX,1        ;       BEFORE NON-ZERO WORD
03BD  D1 E0                     SHL AX,1
03BF  D1 E0                     SHL AX,1
03C1  03 C1                     ADD AX,CX       ; AX <- COLUMN (0..639)

03C3  8B DA                     MOV BX,DX       ;BX <- ROW #; AX HOLDS COL #
03C5  8B FE                     MOV DI,SI       ;DI <- START OF FRAME
03C7  B9 0000                   MOV CX,0
```

```
03CA 41              NXTPATH: INC CX
03CB E8 049F R                CALL TSTPATH
03CE 75 FA                    JNZ NXTPATH
                              ;HERE CX HOLDS PATH #. I.E. 1ST PATH WHICH IS 0, AROUND PIXEL

                              ;NOW DETERMINE THE DOT BOUNDARIES: TMOST,BMOST,LMOST,RMOST.
                              ; HERE AX=COL#; BX=ROW# OF 1ST NON ZERO PIXEL ENCONTERED
03D0 8B D1                    MOV DX,CX        ; SAVE PATH #.

03D2 8B D0                    MOV DX,AX
03D4 2B D1                    SUB DX,CX
03D6 89 16 0025 R             MOV LMOST,DX     ;INIT LMOST

03DA 8B D0                    MOV DX,AX
03DC 03 D1                    ADD DX,CX
03DE 89 16 0027 R             MOV RMOST,DX     ;INIT RMOST

03E2 8B D3                    MOV DX,BX
03E4 03 D1                    ADD DX,CX
03E6 89 16 0029 R             MOV BMOST,DX     ;INIT BMOST

03EA 89 1E 002B R             MOV TMOST,BX     ;INIT TMOST

                              ;NOW COMPUTE NEW LMOST,...TMOST,  IF = OLD THEN FINISHED
03EE 8B 16 0025 R    MOSTNF:  MOV DX,LMOST
03F2 89 16 002D R             MOV LMOSTOLD,DX
03F6 8B 16 0027 R             MOV DX,RMOST
03FA 89 16 002F R             MOV RMOSTOLD,DX
03FE 8B 16 002B R             MOV DX,TMOST
0402 89 16 0033 R             MOV TMOSTOLD,DX
0406 8B 16 0029 R             MOV DX,BMOST
040A 89 16 0031 R             MOV BMOSTOLD,DX


040E 8B D0                    MOV DX,AX
0410 4A              LCOLNZ:  DEC DX
0411 E8 04CC R                CALL TSTCOL2
0414 75 FA                    JNZ LCOLNZ
0416 42                       INC DX
0417 89 16 0025 R             MOV LMOST,DX

041B 8B D0                    MOV DX,AX
041D 42              RCOLNZ:  INC DX
041E E8 04CC R                CALL TSTCOL2
0421 75 FA                    JNZ RCOLNZ
0423 4A                       DEC DX
0424 89 16 0027 R             MOV RMOST,DX
```

```
0428  8B D3                     MOV DX,BX
042A  42               BLINNZ:  INC DX
042B  E8 04E8 R                 CALL TSTLINE2
042E  75 FA                     JNZ BLINNZ
0430  4A                        DEC DX
0431  89 16 0029 R              MOV BMOST,DX

                                ;NOW COMPARE OLD AND NEW BMOST,TMOST ...
0435  8B 16 0027 R              MOV DX,RMOST
0439  39 16 002F R              CMP RMOSTOLD,DX
043D  75 AF                     JNZ MOSTNF      ; MOSTS NOT FOUND
043F  8B 16 0025 R              MOV DX,LMOST
0443  39 16 002D R              CMP LMOSTOLD,DX
0447  75 A5                     JNZ MOSTNF      ; MOSTS NOT FOUND
0449  8B 16 002B R              MOV DX,TMOST
044D  39 16 0033 R              CMP TMOSTOLD,DX
0451  75 9B                     JNZ MOSTNF      ; MOSTS NOT FOUND
0453  8B 16 0029 R              MOV DX,BMOST
0457  39 16 0031 R              CMP BMOSTOLD,DX
045B  75 91                     JNZ MOSTNF      ; MOSTS NOT FOUND

                                ;NOW DETERMINE CENTER OF DOT FROM TMOST,BMOST,LMOST,RMOST.
045D  A1 0027 R                 MOV AX,RMOST
0460  2B 06 0025 R              SUB AX,LMOST
0464  F6 36 0023 R              DIV DIV2
0468  8B 0E 0025 R              MOV CX,LMOST
046C  B4 00                     MOV AH,0
046E  03 C8                     ADD CX,AX     ; CX IS COL # OF CENTER OF DOT

0470  A1 0029 R                 MOV AX,BMOST
0473  2B C3                     SUB AX,BX
0475  F6 36 0023 R              DIV DIV2
0479  B4 00                     MOV AH,0
047B  03 D8                     ADD BX,AX     ; BX IS ROW # OF CENTER OF DOT


                                ;NOW CLEAR DOT DELIMITED BY TMOST,BMOST,LMOST,RMOST.
047D  53                        PUSH BX      ; BX AND CX MUST BE PRESERVED (RETURN VALUES).
047E  8B 1E 002B R              MOV BX,TMOST
0482  4B                        DEC BX
0483  43               NXTRCLR: INC BX
0484  A1 0025 R                 MOV AX,LMOST
0487  48                        DEC AX
0488  40               NXTBCLR: INC AX
0489  E8 0556 R                 CALL CLRBIT
048C  3B 06 0027 R              CMP AX,RMOST
0490  75 F6                     JNZ NXTBCLR
```

```
0492  3B 1E 0029 R            CMP BX,BMOST
0496  75 EB                   JNZ NXTRCLR
0498  5B                      POP BX

                                          ; BX=RETURN ROW # (0..127) OR -1 IF NOT FOUND
                                          ; CX=RETURN COL # (0..639) OR -1 IF NOT FOUND
0499  5F             FINDOT:  POP DI
049A  5E                      POP SI
049B  07                      POP ES
049C  5A                      POP DX
049D  58                      POP AX
049E  C3                      RET
049F                 DETDOT   ENDP
```

```
                                        PAGE
049F                    TSTPATH PROC NEAR
049F  51                        PUSH CX
04A0  F8                        CLC
04A1  E8 04B1 R                 CALL TSTCOL
04A4  75 09                     JNZ PATHNZ
04A6  F9                        STC
04A7  E8 04B1 R                 CALL TSTCOL
04AA  75 03                     JNZ PATHNZ
04AC  E8 0503 R                 CALL TSTLINE
04AF  59                PATHNZ: POP CX
04B0  C3                        RET
04B1                    TSTPATH ENDP


04B1                    TSTCOL  PROC NEAR
04B1  50                        PUSH AX
04B2  53                        PUSH BX
04B3  51                        PUSH CX
04B4  73 06                     JNC LFTCOL
04B6  03 C1                     ADD AX,CX
04B8  41                        INC CX
04B9  EB 04 90                  JMP GOTSTB
04BC  2B C1             LFTCOL: SUB AX,CX
04BE  41                        INC CX
04BF  E8 051A R         GOTSTB: CALL TSTBIT
04C2  75 04                     JNZ OUTCOL
04C4  43                        INC BX
04C5  49                        DEC CX
04C6  75 F7                     JNZ GOTSTB
04C8  59                OUTCOL: POP CX
04C9  5B                        POP BX
04CA  58                        POP AX
04CB  C3                        RET
04CC                    TSTCOL  ENDP

                        ;THIS WILL DETERMINE IF COL DX BETWEEN ROWS TMOST AND BMOST IS ZERO.
04CC                    TSTCOL2 PROC NEAR
04CC  50                        PUSH AX
04CD  53                        PUSH BX
04CE  51                        PUSH CX
04CF  52                        PUSH DX

04D0  8B C2                     MOV AX,DX     ;COL TO BE TESTED
04D2  8B 1E 002B R              MOV BX,TMOST  ;START ROW FOR TEST
04D6  4B                        DEC BX
04D7  43                NXT2COL: INC BX
04D8  E8 051A R                 CALL TSTBIT
04DB  75 06                     JNZ COLNZ
```

```
04DD  3B 1E 0029 R              CMP BX,BNQST
04E1  75 F4                     JNZ NXT2COL
04E3  5A               COLNZ:   POP DX
04E4  59                        POP CX
04E5  5B                        POP BX
04E6  58                        POP AX
04E7  C3                        RET
04E8                   TSTCOL2  ENDP
```

```
                                    PAGE
                              ;THIS WILL DETERMINE IF ROW DX BETWEEN COLS LMOST AND RMOST IS ZERO.
04E8                          TSTLINE2 PROC NEAR
04E8  50                             PUSH AX
04E9  53                             PUSH BX
04EA  51                             PUSH CX
04EB  52                             PUSH DX

04EC  A1 0025 R                      MOV AX,LMOST ;START COL FOR TEST
04EF  8B DA                          MOV BX,DX    ;ROW TO BE TESTED
04F1  48                             DEC AX
04F2  40                      NXT2ROW: INC AX
04F3  E8 051A R                      CALL TSTBIT
04F6  75 06                          JNZ ROWNZ
04F8  3B 06 0027 R                   CMP AX,RMOST
04FC  75 F4                          JNZ NXT2ROW
04FE  5A                      ROWNZ:  POP DX
04FF  59                             POP CX
0500  5B                             POP BX
0501  58                             POP AX
0502  C3                             RET
0503                          TSTLINE2 ENDP


0503                          TSTLINE PROC NEAR
0503  50                             PUSH AX
0504  53                             PUSH BX
0505  51                             PUSH CX
0506  2B C1                          SUB AX,CX
0508  03 D9                          ADD BX,CX
050A  D1 E1                          SHL CX,1
050C  41                             INC CX
050D  E8 051A R               TSTLBIT: CALL TSTBIT
0510  75 04                          JNZ OUTLINE
0512  40                             INC AX
0513  49                             DEC CX
0514  75 F7                          JNZ TSTLBIT
0516  59                      OUTLINE: POP CX
0517  5B                             POP BX
0518  58                             POP AX
0519  C3                             RET
051A                          TSTLINE ENDP
```

```
                                        PAGE
051A                    TSTBIT PROC NEAR
051A  50                        PUSH AX
051B  53                        PUSH BX
051C  51                        PUSH CX
051D  3D 027F                   CMP AX,639
0520  77 2E                     JA ITSOTST
0522  83 FB 7F                  CMP BX,127
0525  77 29                     JA ITSOTST

0527  50                        PUSH AX
0528  8A C3                     MOV AL,BL
052A  F6 26 0024 R              MUL MULT80
052E  8B D8                     MOV BX,AX
0530  58                        POP AX

0531  50                        PUSH AX
0532  F6 36 0022 R              DIV DIV8
0536  B4 00                     MOV AH,0
0538  03 D8                     ADD BX,AX
053A  58                        POP AX

053B  F6 36 0022 R              DIV DIV8
053F  B1 80                     MOV CL,80H
0541  80 FC 00          CHKODONE: CMP AH,0
0544  74 06                     JZ GOTST
0546  D0 E9                     SHR CL,1
0548  FE CC                     DEC AH
054A  EB F5                     JMP CHKODONE
054C  84 09             GOTST:  TEST [BX][DI],CL
054E  75 02                     JNZ POPALL
0550  2B C0             ITSOTST: SUB AX,AX
0552  59                POPALL: POP CX
0553  5B                        POP BX
0554  58                        POP AX
0555  C3                        RET
0556                    TSTBIT  ENDP
```

PAGE

```
0556                    CLRBIT PROC NEAR
0556  50                       PUSH AX
0557  53                       PUSH BX
0558  51                       PUSH CX
0559  3D 027F                  CMP AX,639
055C  77 2E                    JA OUTCLR
055E  83 FB 7F                 CMP BX,127
0561  77 29                    JA OUTCLR

0563  50                       PUSH AX
0564  8A C3                    MOV AL,BL
0566  F6 26 0024 R             MUL MULT80
056A  8B D8                    MOV BX,AX
056C  58                       POP AX

056D  50                       PUSH AX
056E  F6 36 0022 R             DIV DIV8
0572  B4 00                    MOV AH,0
0574  03 D8                    ADD BX,AX
0576  58                       POP AX

0577  F6 36 0022 R             DIV DIV8
057B  B1 80                    MOV CL,80H
057D  80 FC 00          MSKCLR: CMP AH,0
0580  74 06                    JZ GOCLR
0582  D0 E9                    SHR CL,1
0584  FE CC                    DEC AH
0586  EB F5                    JMP MSKCLR
0588  F6 D1             GOCLR:  NOT CL
058A  20 09                    AND [BX][DI],CL
058C  59                OUTCLR: POP CX
058D  5B                       POP BX
058E  58                       POP AX
058F  C3                       RET
0590                    CLRBIT  ENDP
```

```
                                        PAGE
0590                    BLINK  PROC NEAR
0590  58                       POP AX         ; RETURN ADDR.
0591  5E                       POP SI         ; ADDR OF RECORD
0592  50                       PUSH AX        ; INSTALL-RETURN ADDR.
0593  06                       PUSH ES        ; SAVE
0594  55                       PUSH BP

0595  8B 1C                    MOV BX,[SI]    ; ROW # 0..127
0597  8B 4C 02                 MOV CX,[SI+2]  ; COL # 0..639
059A  81 F9 027F               CMP CX,639
059E  77 08                    JA NOGOOD
05A0  83 FB 7F                 CMP BX,127
05A3  77 03                    JA NOGOOD
05A5  EB 07 90                 JMP INRANGE
05A8  BB 0000           NOGOOD: MOV BX,0
05AB  B9 0000                  MOV CX,0

05AE  F7 C3 0001        INRANGE: TEST BX,0001H
05B2  74 06                    JZ EVENROW
05B4  BE BA00                  MOV SI,0B800H+200H
05B7  EB 04 90                 JMP SETBX
05BA  BE B800           EVENROW: MOV SI,0B800H
05BD  D1 EB             SETBX:  SHR BX,1      ;DIVIDE BY 2: BX IS ROW # IN EVEN OR ODD
                                              ; ROWS OF VIDEO RAM. BX=0,1,2,3...

05BF  8A C3                    MOV AL,BL
05C1  F6 26 0024 R             MUL MULT80
05C5  8B D8                    MOV BX,AX      ;# OF BYTES ON ROWS ABOVE

05C7  8B C1                    MOV AX,CX      ;COL #
05C9  F6 36 0022 R             DIV DIV8
05CD  B4 00                    MOV AH,0       ;AX=# OF BYTES ON ROW, BEFORE BYTE CONTAINING DOT
05CF  03 D8                    ADD BX,AX      ;BX= BYTE # OF BYTE CONTAINING DOT
05D1  8B FB                    MOV DI,BX      ;DISPLACEMENT IN VIDEO RAM

05D3  8B C1                    MOV AX,CX
05D5  F6 36 0022 R             DIV DIV8       ;AH=REMAINDER=0..7
05D9  B1 80                    MOV CL,80H
05DB  80 FC 00          OKMSK:  CMP AH,0
05DE  74 06                    JZ GOBLINK
05E0  D0 E9                    SHR CL,1
05E2  FE CC                    DEC AH
05E4  EB F5                    JMP OKMSK

                               ;HERE CL HOLDS MASK, SI=BASE ADDR IN VIDEO RAM, DI=DISPLACEMENT
05E6  8A D1             GOBLINK: MOV DL,CL
05E8  E8 01BE R                CALL KEYCLR
05EB  1E                       PUSH DS
```

```
05EC  8E DE                  MOV DS,SI
05EE  8A 35                  MOV DH,[DI]      ;SAVE ORIGINAL VALUE
05F0  1F                     POP DS
05F1  1E            NXTBLK:  PUSH DS
05F2  8E DE                  MOV DS,SI
05F4  30 15                  XOR  [DI],DL
05F6  1F                     POP DS
05F7  B9 00FA                MOV CX,250        ; 0.25 SEC
05FA  E8 01E8 R              CALL DELAY
05FD  E8 01A9 R              CALL KEYCHK
0600  74 EF                  JZ NXTBLK
0602  E8 01BE R              CALL KEYCLR
0605  1E                     PUSH DS
0606  8E DE                  MOV DS,SI
0608  88 35                  MOV [DI],DH      ;RESTORE ORIGINAL VALUE
060A  1F                     POP DS

060B  5D            OUTBLINK: POP BP
060C  07                      POP ES
060D  C3                      RET
060E                BLINK    ENDP


060E                CODE      ENDS
                              END
Open procedures:

DELAY. . . . . . . . . . . . . .
```

Segments and groups:

N a m e	Size	align	combine	class
CODE	060E	BYTE	PUBLIC	'PASCAL'
DATA	0035	PARA	PUBLIC	

Symbols:

N a m e	Type	Value	Attr	
ACIACLR.	N PROC	0208	CODE	Length =000B
AGAIN.	L NEAR	008A	CODE	
ARMOUT	N PROC	01C4	CODE	Global Length =0024
BEEP	N PROC	0234	CODE	Length =0007
BITMAP	L WORD	0002	DATA	
BITOK.	L NEAR	03B5	CODE	
BLINK.	N PROC	0590	CODE	Global Length =007E
BLINNZ	L NEAR	042A	CODE	
BMOST.	L WORD	0029	DATA	
BMOSTOLD	L WORD	0031	DATA	
BYTE_CNT	L WORD	0006	DATA	
CALLBIOS	N PROC	0024	CODE	Global Length =0030
CALLDOS.	N PROC	0000	CODE	Global Length =0024
CHKOBONE	L NEAR	0541	CODE	
CLEARW	N PROC	01F6	CODE	Length =0012
CLRBIT	N PROC	0556	CODE	Length =003A
CLRCOMM.	N PROC	007B	CODE	Global Length =002C
COLCTR	L WORD	001D	DATA	
COLNZ.	L NEAR	04E3	CODE	
COMMAND.	L BYTE	001A	DATA	
CONTROL.	L WORD	0010	DATA	
DATAIN	L WORD	0014	DATA	
DATAOUT.	L WORD	0016	DATA	
DEL1	L NEAR	01EB	CODE	
DEL2	L NEAR	01EF	CODE	
DELAY.	N PROC	01E8	CODE	Length =0000
DETDOT	N PROC	0378	CODE	Length =0127
DETDOTS.	N PROC	032B	CODE	Global Length =004D
DIV2	L BYTE	0023	DATA	
DIV8	L BYTE	0022	DATA	
DIVISOR.	L BYTE	0021	DATA	
DNCHK.	L NEAR	0164	CODE	
ENHANCE2	N PROC	023B	CODE	Global Length =00F0
EVENROW.	L NEAR	05BA	CODE	
EXPOSE_TIME.	L WORD	0018	DATA	
FINDOT	L NEAR	0499	CODE	
FOUNDD	L NEAR	0398	CODE	
FRGRAB	N PROC	00F5	CODE	Global Length =00CF

```
GMODE. . . . . . . . . . . . . .N PROC  0054  CODE  Global  Length =0027
GOBLINK. . . . . . . . . . . . .L NEAR  05E6  CODE
GOCLR. . . . . . . . . . . . . .L NEAR  0588  CODE
GOTST. . . . . . . . . . . . . .L NEAR  054C  CODE
GOTSTB . . . . . . . . . . . . .L NEAR  04BF  CODE
INRANGE. . . . . . . . . . . . .L NEAR  05AE  CODE
INTNO. . . . . . . . . . . . . .L BYTE  0044  CODE
ITSOTST. . . . . . . . . . . . .L NEAR  0530  CODE
KEYCHK . . . . . . . . . . . . .L NEAR  01A9  CODE
KEYCLR . . . . . . . . . . . . .L NEAR  01BE  CODE
KEY_VALUE. . . . . . . . . . . .L WORD  0008  DATA
LCOLNZ . . . . . . . . . . . . .L NEAR  0410  CODE
LFTCOL . . . . . . . . . . . . .L NEAR  04BC  CODE
LMOST. . . . . . . . . . . . . .L WORD  0025  DATA
LMOSTOLD . . . . . . . . . . . .L WORD  002D  DATA
MAPADR . . . . . . . . . . . . .L WORD  0000  DATA
MOSTNF . . . . . . . . . . . . .L NEAR  03EE  CODE
MOVESCR. . . . . . . . . . . . .N PROC  00A7  CODE  Global  Length =004E
MSKCLR . . . . . . . . . . . . .L NEAR  057D  CODE
MULT80 . . . . . . . . . . . . .L BYTE  0024  DATA
NB02 . . . . . . . . . . . . . .L NEAR  0272  CODE
NBE2 . . . . . . . . . . . . . .L NEAR  0251  CODE
NEBYT. . . . . . . . . . . . . .L NEAR  02A8  CODE
NFBYT. . . . . . . . . . . . . .L NEAR  014C  CODE
NOBYT. . . . . . . . . . . . . .L NEAR  02CA  CODE
NOCA . . . . . . . . . . . . . .L NEAR  025F  CODE
NOCB . . . . . . . . . . . . . .L NEAR  0266  CODE
NOGC . . . . . . . . . . . . . .L NEAR  027F  CODE
NOCD . . . . . . . . . . . . . .L NEAR  0287  CODE
NOGOOD . . . . . . . . . . . . .L NEAR  05A8  CODE
NOKEY. . . . . . . . . . . . . .L NEAR  01BC  CODE
NOSOAK . . . . . . . . . . . . .L NEAR  0233  CODE
NROW2. . . . . . . . . . . . . .L NEAR  024F  CODE
NTA. . . . . . . . . . . . . . .L NEAR  0195  CODE
NTO. . . . . . . . . . . . . . .L NEAR  0172  CODE
NXT2 . . . . . . . . . . . . . .L NEAR  0257  CODE
NXT2COL. . . . . . . . . . . . .L NEAR  04D7  CODE
NXT2ROW. . . . . . . . . . . . .L NEAR  04F2  CODE
NXT3 . . . . . . . . . . . . . .L NEAR  0278  CODE
NXTAND1. . . . . . . . . . . . .L NEAR  0345  CODE
NXTAND2. . . . . . . . . . . . .L NEAR  035C  CODE
NXTBCLR. . . . . . . . . . . . .L NEAR  0488  CODE
NXTBLK . . . . . . . . . . . . .L NEAR  05F1  CODE
NXTCLR1. . . . . . . . . . . . .L NEAR  0308  CODE
NXTCLR2. . . . . . . . . . . . .L NEAR  031F  CODE
NXTCMP . . . . . . . . . . . . .L NEAR  0384  CODE
NXTDOT . . . . . . . . . . . . .L NEAR  0366  CODE
NXTPATH. . . . . . . . . . . . .L NEAR  03CA  CODE
NXTRCLR. . . . . . . . . . . . .L NEAR  0483  CODE
```

```
NXTROW . . . . . . . . . . . . .L NEAR   00D8   CODE
OKMSK. . . . . . . . . . . . . .L NEAR   05DB   CODE
OUTBLINK . . . . . . . . . . . .L NEAR   060B   CODE
OUTCLR . . . . . . . . . . . . .L NEAR   058C   CODE
OUTCOL . . . . . . . . . . . . .L NEAR   04C8   CODE
OUTDELAY . . . . . . . . . . . .L NEAR   01F4   CODE
OUTLINE. . . . . . . . . . . . .L NEAR   0516   CODE
PATHNZ . . . . . . . . . . . . .L NEAR   04AF   CODE
POPALL . . . . . . . . . . . . .L NEAR   0552   CODE
RCOLNZ . . . . . . . . . . . . .L NEAR   041D   CODE
RECHK. . . . . . . . . . . . . .L NEAR   014F   CODE
RMOST. . . . . . . . . . . . . .L WORD   0027   DATA
RMOSTOLD . . . . . . . . . . . .L WORD   002F   DATA
ROWCTR . . . . . . . . . . . . .L WORD   001B   DATA
ROWNZ. . . . . . . . . . . . . .L NEAR   04FE   CODE
S1 . . . . . . . . . . . . . . .L NEAR   022A   CODE
S2 . . . . . . . . . . . . . . .L NEAR   022E   CODE
SCREEN_START . . . . . . . . . .L WORD   000A   DATA
SCR_COLCT. . . . . . . . . . . .L WORD   000E   DATA
SCR_ROWCT. . . . . . . . . . . .L WORD   000C   DATA
SENDCMD. . . . . . . . . . . . .N PROC   0213   CODE   Length =0011
SETBX. . . . . . . . . . . . . .L NEAR   05BD   CODE
SHLNXT . . . . . . . . . . . . .L NEAR   03AE   CODE
SOAK . . . . . . . . . . . . . .N PROC   0224   CODE   Length =0010
STATUS . . . . . . . . . . . . .L WORD   0012   DATA
SUMB . . . . . . . . . . . . . .L WORD   001F   DATA
SWINT. . . . . . . . . . . . . .L BYTE   0043   CODE
TMOST. . . . . . . . . . . . . .L WORD   002B   DATA
TMOSTOLD . . . . . . . . . . . .L WORD   0033   DATA
TSTBIT . . . . . . . . . . . . .N PROC   051A   CODE   Length =003C
TSTCOL . . . . . . . . . . . . .N PROC   04B1   CODE   Length =001B
TSTCOL2. . . . . . . . . . . . .N PROC   04CC   CODE   Length =001C
TSTLBIT. . . . . . . . . . . . .L NEAR   050D   CODE
TSTLINE. . . . . . . . . . . . .N PROC   0503   CODE   Length =0017
TSTLINE2 . . . . . . . . . . . .N PROC   04EB   CODE   Length =001B
TSTPATH. . . . . . . . . . . . .N PROC   049F   CODE   Length =0012
WORKMAP. . . . . . . . . . . . .L WORD   0004   DATA

Warning Severe
Errors  Errors
0       0
```